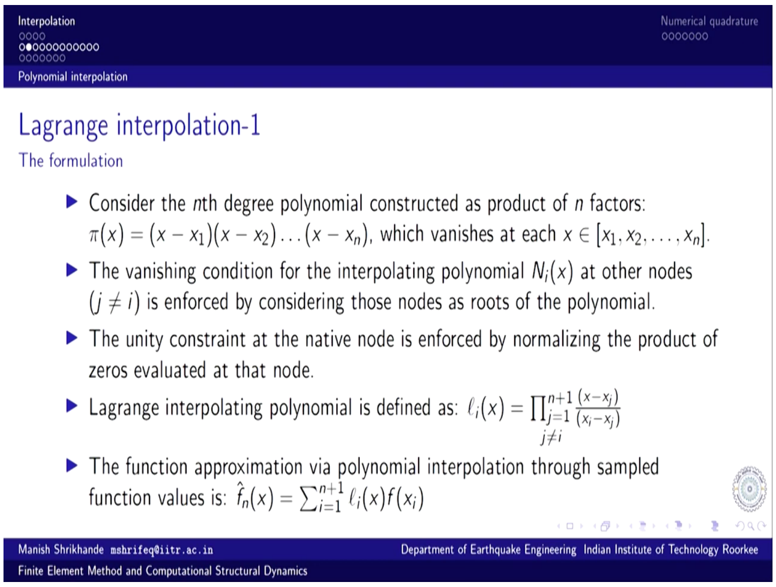
Finite Element Method and Computational Structural Dynamics

Prof. Manish Shrikhande
Department of Earthquake Engineering
Indian Institute of Technology, Roorkee

Lecture - 05
Polynomial Interpolation and Numerical Quadrature–II

(Refer Slide Time: 00:38)


Interpolation | Numerical quadrature

Polynomial interpolation

Lagrange interpolation-1

The formulation

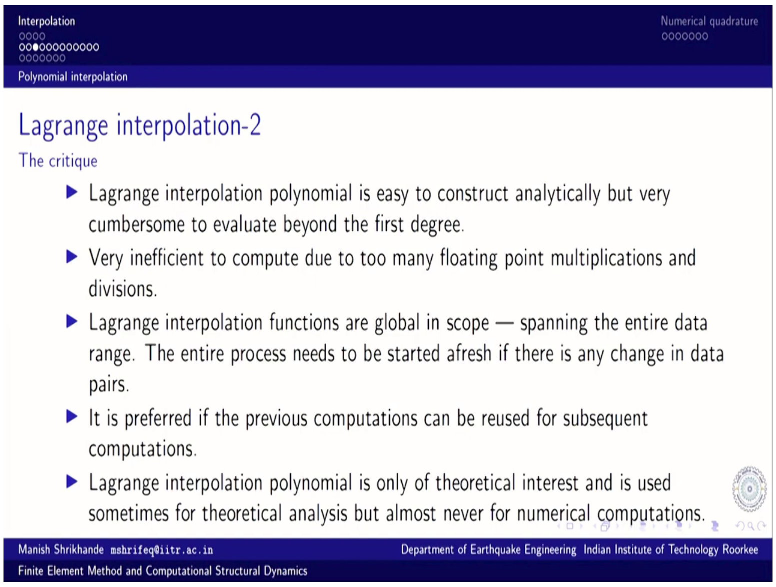
- Consider the nth degree polynomial constructed as product of n factors: $\pi(x) = (x - x_1)(x - x_2)\dots(x - x_n)$, which vanishes at each $x \in [x_1, x_2, \dots, x_n]$.
- The vanishing condition for the interpolating polynomial $N_i(x)$ at other nodes $(j \neq i)$ is enforced by considering those nodes as roots of the polynomial.
- The unity constraint at the native node is enforced by normalizing the product of zeros evaluated at that node.
- Lagrange interpolating polynomial is defined as: $\ell_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n+1} \frac{(x-x_j)}{(x_i-x_j)}$
- The function approximation via polynomial interpolation through sampled function values is: $\hat{f}_n(x) = \sum_{i=1}^{n+1} \ell_i(x) f(x_i)$

Manish Shrikhande mshrifeq@iitr.ac.in | Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

Hello. So, yesterday we discussed this formulation of Lagrange interpolation as normalised product of 0s of all other nodes evaluated at the node for which we are developing the interpolation function. So, the product is essentially multiplied by equation of roots (x - x $_i$), (x- x $_j$) and then this is evaluated at the i th node for which we are deriving the interpolation function and that is normalised with respect to that value. So both the conditions of interpolation that is N $_i$ evaluated at x $_j$ should be equal to δ $_{i j}$ .It should evaluate the unity at x $_i$, and it should vanish at all other nodes when i is not equal to j. And once we have this interpolation function, then it can be combined with the function values that we have at specified at these nodes and just to express it as a linear combination. So, f $_i$ is the function approximation through interpolation model polynomial interpolation. So, nth degree polynomial interpolation using Lagrange's interpolation model.

(Refer Slide Time: 01:59)


Interpolation

Numerical quadrature

Polynomial interpolation

Lagrange interpolation-2

The critique

- Lagrange interpolation polynomial is easy to construct analytically but very cumbersome to evaluate beyond the first degree.
- Very inefficient to compute due to too many floating point multiplications and divisions.
- Lagrange interpolation functions are global in scope — spanning the entire data range. The entire process needs to be started afresh if there is any change in data pairs.
- It is preferred if the previous computations can be reused for subsequent computations.
- Lagrange interpolation polynomial is only of theoretical interest and is used sometimes for theoretical analysis but almost never for numerical computations.

Manish Shrikhande mshrifeq@iitr.ac.in Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

Now, Lagrange interpolation as I said it is very easy to construct analytically, but it is very cumbersome to evaluate beyond the first degree. As you can see it is constructed by accumulating the products of each term. So, nth degree polynomial will require n number of evaluations, and then of course, the normalisation that is division as well.

So, it is very inefficient to compute because there are far too many floating-point multiplications and divisions. And as we had already seen earlier every single floating point operation is a source of round of error. So, the key to reliable and robust computation floating point computation is to do with as few floating point operations as possible and that will also increase the efficiency of any algorithm.

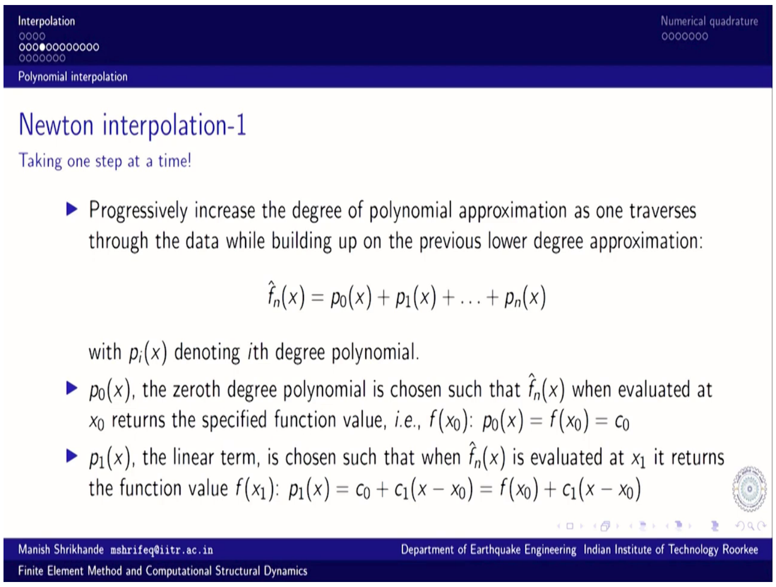
Another problem with Lagrange interpolation is that they are global in scope. We just develop entire approximation over the entire range. And if something goes wrong or if there is any change in data or if we change the node a little bit here and there some somewhere in between, then the entire exercise has to be started a fresh there is no way we can make use of already computed results if there are any.

So, the entire process for even for one change in data point the whole process has to be repeated completely from first step. And of course, when we talk of numerical computations and efficiency it is of course preferred if previous computations can be reused for subsequent computations.

So, the Lagrange interpolation as I said it is a polynomial nth degree polynomial is only of theoretical interests and it is very easy to explain how it works. But because the polynomial interpolation any polynomial nth degree polynomial passing through n + 1 number of points is a unique polynomial. So, no matter which form of interpolation I use. As long as it is a polynomial interpolating through function values n + 1 number of function values, then it is going to be the same polynomial irrespective of which way I arrive at it. So, Lagrange interpolation because of its simplicity and elegance in construction is often used for theoretical analysis and error bounds etcetera, but never almost never for any actual numerical computations.

So, to build upon that error the concern that it is a global polynomial, global scope and everything has to be constructed afresh. We have improvement on that by use using Newton interpolation model wherein we increase the degree of polynomial; we increase the degree of interpolation. So, the first term is a constant term – p_0 it is a constant term; p_1 is a first degree polynomial; p_2 would be second degree polynomial and so on. And so that the nth term is the nth degree polynomial and that is how we construct this approximation.

(Refer Slide Time: 05:35)



Interpolation

Polynomial interpolation

Numerical quadrature

Newton interpolation-1

Taking one step at a time!

- Progressively increase the degree of polynomial approximation as one traverses through the data while building up on the previous lower degree approximation:

$$\hat{f}_n(x) = p_0(x) + p_1(x) + \ldots + p_n(x)$$

with $p_i(x)$ denoting ith degree polynomial.

- $p_0(x)$, the zeroth degree polynomial is chosen such that $\hat{f}_n(x)$ when evaluated at x_0 returns the specified function value, *i.e.*, $f(x_0)$: $p_0(x) = f(x_0) = c_0$
- $p_1(x)$, the linear term, is chosen such that when $\hat{f}_n(x)$ is evaluated at x_1 it returns the function value $f(x_1)$: $p_1(x) = c_0 + c_1(x - x_0) = f(x_0) + c_1(x - x_0)$

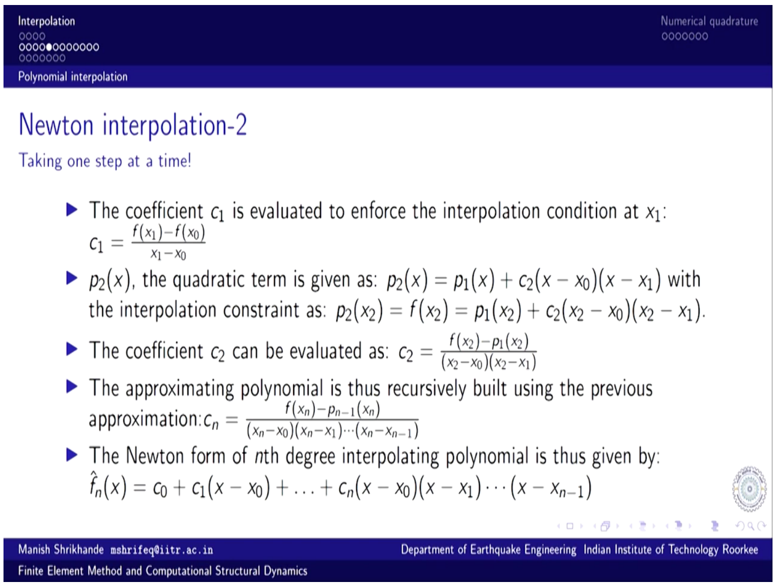
Manish Shrikhande mshrifeq@iitr.ac.in Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

And gradually so that we can make use of previous results and if any intermediate data is left out or it has to be changed then from that point onwards it has to be modified. So, p 0 , the 0 th degree polynomial is chosen such that approximation function $\hat{f}$ when it is evaluated at x $_0$ returns the specified values.

So, it is just the function value that specified at x_0, so that becomes just a constant term c_0. Let us call it c_0. Now, considering that c_0 is already known, then we define next term p $_{1\ x}$ such that p $_{1\ x}$ returns the value of f of x $_0$ when evaluated at x $_0$, and it returns the value of f of x $_1$ when it is evaluated at x $_1$. So, we impose this condition. And the way to do that would be to enforce that the coefficient linear term should vanish at x $_0$. So, it is just x. So, the polynomial model that works out is c $_0$ + c $_1$ that c $_1$ is still unknown, c $_1$ (x - x_0). So, this first-degree polynomial of course, vanishes at x $_0$, and because c $_0$ has been evaluated such that it returns it is same as the value of the function specified function value at x $_0$. So, it meets that criteria. And now we impose the condition that when p $_{1\ x}$ is evaluated at x $_1$, it has to return the value of f $_{x\ 1}$ f of x $_1$ that is specified. And when we impose that condition that allows us to find out what is the coefficient c $_1$ going to be and that is what you get.

(Refer Slide Time: 07:48)



Interpolation — Numerical quadrature

Polynomial interpolation

Newton interpolation-2

Taking one step at a time!

- The coefficient c_1 is evaluated to enforce the interpolation condition at x_1: $c_1 = \frac{f(x_1)-f(x_0)}{x_1-x_0}$
- $p_2(x)$, the quadratic term is given as: $p_2(x) = p_1(x) + c_2(x-x_0)(x-x_1)$ with the interpolation constraint as: $p_2(x_2) = f(x_2) = p_1(x_2) + c_2(x_2-x_0)(x_2-x_1)$.
- The coefficient c_2 can be evaluated as: $c_2 = \frac{f(x_2)-p_1(x_2)}{(x_2-x_0)(x_2-x_1)}$
- The approximating polynomial is thus recursively built using the previous approximation: $c_n = \frac{f(x_n)-p_{n-1}(x_n)}{(x_n-x_0)(x_n-x_1)\cdots(x_n-x_{n-1})}$
- The Newton form of nth degree interpolating polynomial is thus given by: $\hat{f}_n(x) = c_0 + c_1(x-x_0) + \ldots + c_n(x-x_0)(x-x_1)\cdots(x-x_{n-1})$

Manish Shrikhande mshrifeq@iitr.ac.in — Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

So, by enforcing the interpolation condition at x $_1$, we can evaluate what is the unknown coefficient c $_1$ going to be. And you can see the pattern here now you can look at it as a difference formula. It looks like a difference formula and it is a difference of formula, so that gives us another way of developing the of computing the coefficients of Newton interpolation model by using divided differences.

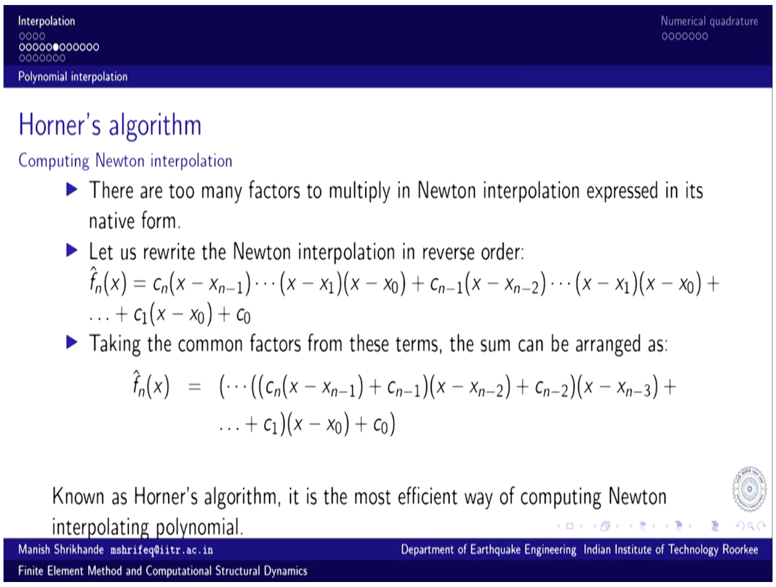
So, what you see here, c_1 evaluated as difference in the function values divided by the difference in the independent variable. So, $f(x_1) - f(x_0) / (x_1 - x_0)$. And then next term quadratic term is again imposed with the same addition on, I mean we add modification whatever quadratic term $(c_{2x} - x_0)(x - x_1)$ on top of p_{1x}, so the first degree polynomial, first degree interpolation that we had earlier found.

And then again the interpolation constraint is applied that this second degree polynomial p_{2x} has to evaluate to $f(x\ 2)$ at x_2. And when we impose this condition, then you get the basic coefficient c_2 evaluated at $f(x\ 2)$ minus that first degree polynomial evaluated at $x\ 2$ and divided by this $(x_2 - x_0)$ and multiplied by $(x_2 - x_1)$. So, this is how it is recursively built using previous approximations. So, n^{th} degree term coefficient of n^{th} degree term would be evaluated by $f(x_n)$ minus value of the previous degree of interpolation polynomial evaluated at x_n and divided by the normalising factor.

So, the Newton form of nth degree interpolation polynomial is given by progressively increasing degree of polynomial term in the series. So, starts with a constant term, then you add a linear term, then you add a quadratic term, then you add a cubic term and so on until you build n^{th} degree term, so that is what the formulation or the conceptually Newton interpolation model talks about. But as you can see here still there are far too many floating point multiplications if we evaluate it according to this definition the way the Newton form is defined. But if we look at this pattern very closely, we can see that the factors are common for example, $x - x_0$ is a common factor for all terms starting from the second term all the way to $n + 1^{th}$ term. Similarly, $x - x_1$ is a common factor for all term starting from third term to $n + 1^{th}$ term and so on.

So, we can actually minimise the floating point multiplication by recognising this and rearranging the expression backwards instead of starting from constant term and going all the way to nth term, we can start writing the expression from nth term and taking the common factors and working our way back to 0^{th} term or the constant term.

(Refer Slide Time: 11:56)



Horner's algorithm

Computing Newton interpolation

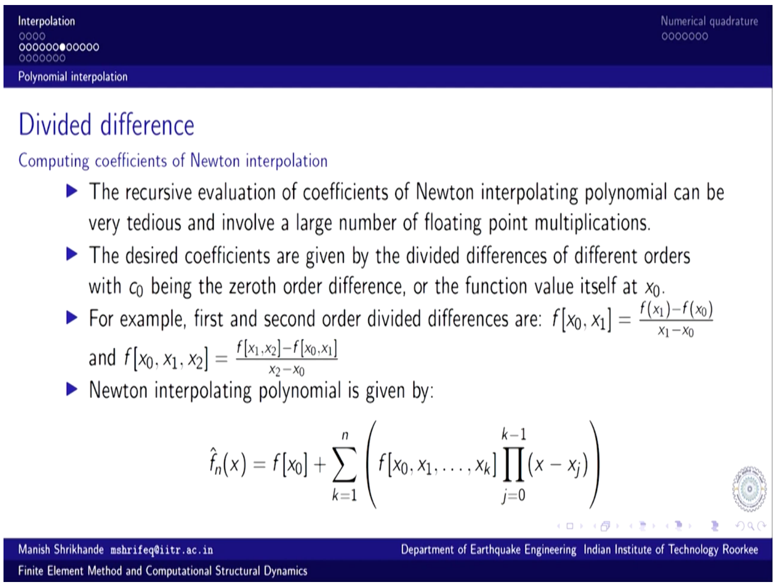
- There are too many factors to multiply in Newton interpolation expressed in its native form.
- Let us rewrite the Newton interpolation in reverse order: $\hat{f}_n(x) = c_n(x - x_{n-1})\cdots(x - x_1)(x - x_0) + c_{n-1}(x - x_{n-2})\cdots(x - x_1)(x - x_0) + \ldots + c_1(x - x_0) + c_0$
- Taking the common factors from these terms, the sum can be arranged as:

$$\begin{aligned}\hat{f}_n(x) &= (\cdots((c_n(x - x_{n-1}) + c_{n-1})(x - x_{n-2}) + c_{n-2})(x - x_{n-3}) + \\ &\quad \ldots + c_1)(x - x_0) + c_0)\end{aligned}$$

Known as Horner's algorithm, it is the most efficient way of computing Newton interpolating polynomial.

So, that is what we do here. Write the expression in the reverse order. And then once we realise that there are too many common factors it is easy to reorganise this by taking the common factors out and its number of multiplications drop down dramatically. And this is the most efficient way of computing Newton interpolation polynomial. And this is called Horner's algorithm.

(Refer Slide Time: 12:30)



Divided difference

Computing coefficients of Newton interpolation

- The recursive evaluation of coefficients of Newton interpolating polynomial can be very tedious and involve a large number of floating point multiplications.
- The desired coefficients are given by the divided differences of different orders with c_0 being the zeroth order difference, or the function value itself at x_0.
- For example, first and second order divided differences are: $f[x_0, x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0}$ and $f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$
- Newton interpolating polynomial is given by:

$$\hat{f}_n(x) = f[x_0] + \sum_{k=1}^{n} \left(f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_j) \right)$$

Another way of computing recursive evaluation I mean this is the evaluation of the function itself Newton interpolation polynomial, but we still need to find out what is the value of these coefficients, what are these coefficients c_0, c_1, c_2, etcetera.

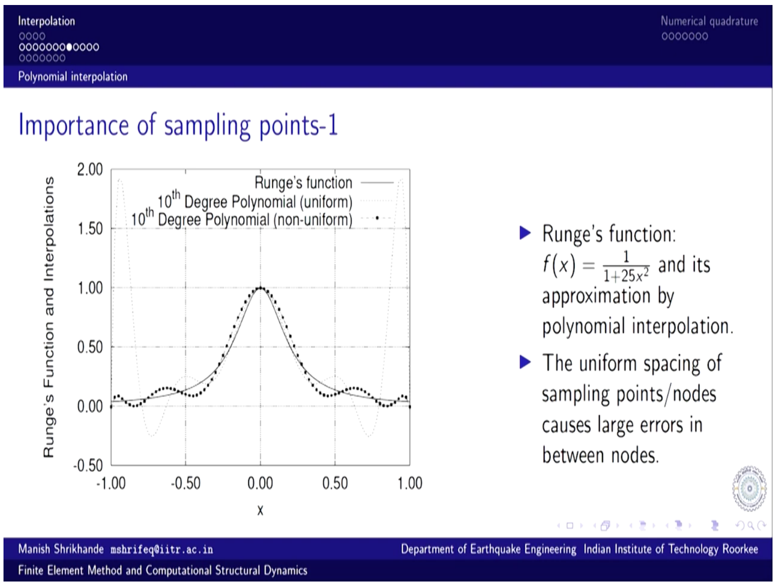
So, this as I said earlier see the computation of c_1 provided as a hint that we can actually compute these coefficients by way of divided differences. So, that is possible by using recursive valuation of coefficients by using the divided difference of different orders and c_0 being the 0^{th} order difference. So, that is the just the function value at x_0. Then c_1 is the first order difference, and c_2 is the second order difference and so on.

So, if we arrange the values, then it is very easy to compute and coefficients can be computed very quickly in an automated almost an in an automated fashion. And then the Newton interpolation polynomial can be simply expressed in terms of divided differences.

So, here within square brackets f within square brackets x 0, these are the these denote the divided differences. So, one term that is the 0^{th} order divided difference, and this one is the k $+1^{th}$ order divided difference, so coefficients. So, with this we can this way we can compute the interpolation polynomial and evaluated it very efficiently for any particular value of x that we may need by using Horner's algorithm. And we can go ahead with our computations. Now, in the beginning when we were discussing about the Weierstrass approximation theorem, which said that it is possible to approximate any function no matter what its origin by a polynomial approximation over a finite range.

And the approximation can be as close as desired that is theoretically true because polynomials are an infinite family, I mean the basis infinite basis functions can be a constructed. So, theoretically you should we should be able to find any function which is finite without any singularities and over a finite region. So, we should be able to approximate that function using sufficient number of polynomial terms, but the problem as I said earlier it is not specified the Weierstrass approximation theorem does not tell us about how to construct or what is the criteria for imposing the constraints of polynomial or closeness of approximation, how it is to be measured, and that actually has a difference impact on the quality of approximation as we will see just now.

(Refer Slide Time: 16:06)



Importance of sampling points-1

- Runge's function: $f(x) = \frac{1}{1+25x^2}$ and its approximation by polynomial interpolation.
- The uniform spacing of sampling points/nodes causes large errors in between nodes.

So, importance of sampling points. So, where do I choose the sampling points? So the nodes. If I have a liberty to choose the nodes of interpolation, I mean I can choose the nodes of interpolation there are large number of tabulated data points and if I have the liberty to pick the nodes according to my wishes, according to it is left to us, then it might be worthwhile to see which nodes to choose, so that for a given degree of interpolation, so that we may be able to derive a better approximation.

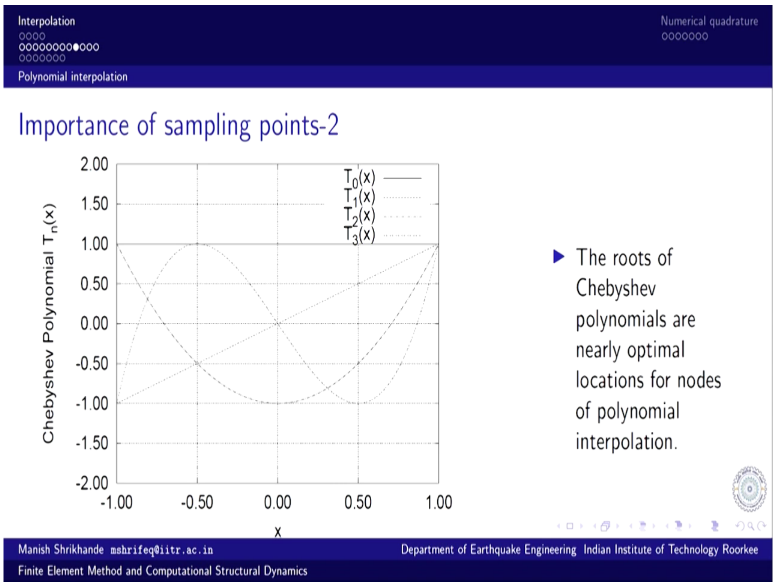
And you may like to look at this particular thing, I mean this is called Runge's function that is a rational polynomial. So, it is a function which is a ratio of I mean constant term divided by some polynomial term in the denominator, so that it blows and then it decays as the denominator increases. So, the solid line actually describes the Runge's function exactly the in analytical form, and the dotted line I mean the dashed line represents 10th degree polynomial approximation. I mean when I when we say when we try to approximate we would obviously inclined to divide uniformly any interval that we are given. So, for example, in this case we are we have this interval from -1 to + 1. So, if I am asked to interpolate between these I mean 10th degree polynomial, so I would just pick up eleven points which are uniformly spaced in this range that is the first impulse that anybody would do.

Now it so happens that this is not a very sound strategy although it is easier for us to do that and hope that everything will work out very well. But it so happens that it does not always work out so well. So, the dashed line you can see is the 10th degree interpolation polynomial based on for which the nodes were derived on the basis of equidistant approach. So, you can see that the function value exactly matches at regular intervals. So, the Runge's function and the 10th degree interpolation polynomial they match exactly at the nodes of interpolation, but in between the nodes of interpolation particularly I would draw your attention to the end interval. There are wild fluctuations between the between two nodes and that is because of the constraint that has been imposed. It is a very high degree polynomial, 10th degree polynomial and 10th degree polynomial if it has to agree with the higher degree polynomials, obviously have large curvatures and all. And once they have to agree to certain function values, then there are going to be wild oscillations above the points.

So, there is a limit and that is why the virtue of using only lower order polynomials for interpolation. Higher order polynomials, higher degree polynomials as an interpolation function for approx construction of function approximation are not necessarily a good thing. So, this is particular example I picked up just to highlight this point that more the higher degree polynomial interpolation is not necessarily the best thing, very soon we hit the law of diminishing returns.

So, the Runge's function that I used here is $1/(1+25x^2)$, and we approximated its values at by polynomial interpolation. So, uniform spacing of sampling points or nodes of interpolation, they cause huge errors between the nodes. At the nodes obviously, they agree because the polynomial interpolation constraint has been imposed. So, there is no error, there is no problem with that. But in between nodes of interpolation there are large oscillations, and with no way we can consider that interpolation function or interpolation model approximation is anyway close to the original function although they match exactly at the nodes.

(Refer Slide Time: 21:15)


Interpolation
Numerical quadrature
Polynomial interpolation
$T_0(x)$
$T_1(x)$
$T_2(x)$
$T_3(x)$
Chebyshev Polynomial $T_n(x)$
x
Manish Shrikhande mshrifeq@iitr.ac.in
Department of Earthquake Engineering Indian Institute of Technology Roorkee
Finite Element Method and Computational Structural Dynamics

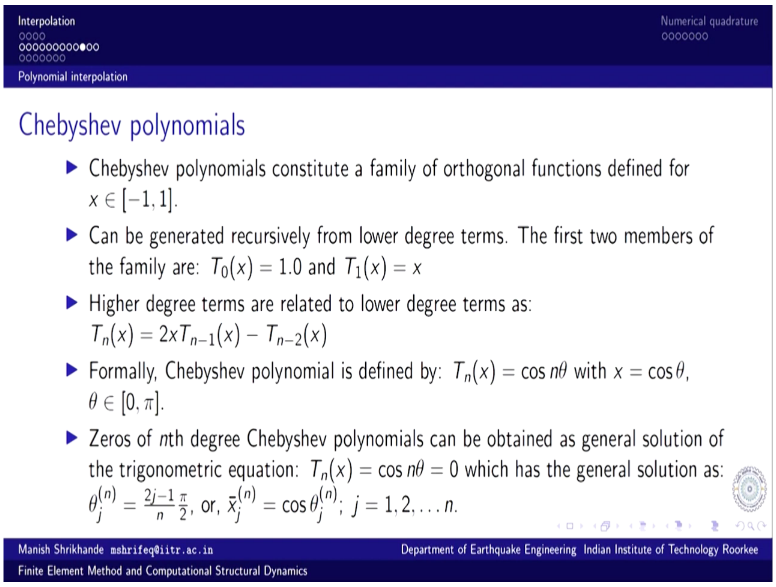
Importance of sampling points-2

- The roots of Chebyshev polynomials are nearly optimal locations for nodes of polynomial interpolation.

So, although the Weierstrass theorem and by basic linear algebra, we know that it is possible to approximate as closely as possible any finite function over a finite range by using polynomial suitable polynomial approximation, but the details are not known, and that we still remains an unsolved problem how to choose the nodes of interpolation, how to choose the nodes for developing polynomial interpolation that will be the best possible optimal location of points. So, we do not know the exact solution yet. But we do have some idea of what should be a better solution, what would be a better solution rather instead of using indiscriminate uniform sampling of data points?

And it so, happens that very interesting thing that it so happens that they roots of Chebyshev polynomials, they constitute a merely optimal set of points which serve as excellent location for the nodes of interpolation. So, these are the Chebyshev polynomials of a respective degrees or increasing degrees. So, T_0 is a constant line, constant line at 1. So, that is the first degree 0th degree constant part term. T_1 is a linear term, and T_2 is a quadratic, T_3 is cubic and so on. And they have very smooth, very nice looking functions, but they also have very interesting property that the roots of these Chebyshev polynomials are nearly optimal locations for the nodes of polynomial interpolation. So, for a 10th degree polynomial, I need 11 nodes. So, I would just pick up the roots of 11th degree Chebyshev polynomial and locate that.

(Refer Slide Time: 23:43)


Interpolation

Numerical quadrature

Polynomial interpolation

Chebyshev polynomials

- Chebyshev polynomials constitute a family of orthogonal functions defined for $x \in [-1, 1]$.
- Can be generated recursively from lower degree terms. The first two members of the family are: $T_0(x) = 1.0$ and $T_1(x) = x$
- Higher degree terms are related to lower degree terms as: $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$
- Formally, Chebyshev polynomial is defined by: $T_n(x) = \cos n\theta$ with $x = \cos\theta$, $\theta \in [0, \pi]$.
- Zeros of nth degree Chebyshev polynomials can be obtained as general solution of the trigonometric equation: $T_n(x) = \cos n\theta = 0$ which has the general solution as: $\theta_j^{(n)} = \frac{2j-1}{n}\frac{\pi}{2}$, or, $\bar{x}_j^{(n)} = \cos\theta_j^{(n)}$; $j = 1, 2, \ldots n$.

Manish Shrikhande mshrifeq@iitr.ac.in Department of Earthquake Engineering Indian Institute of Technology Roorkee

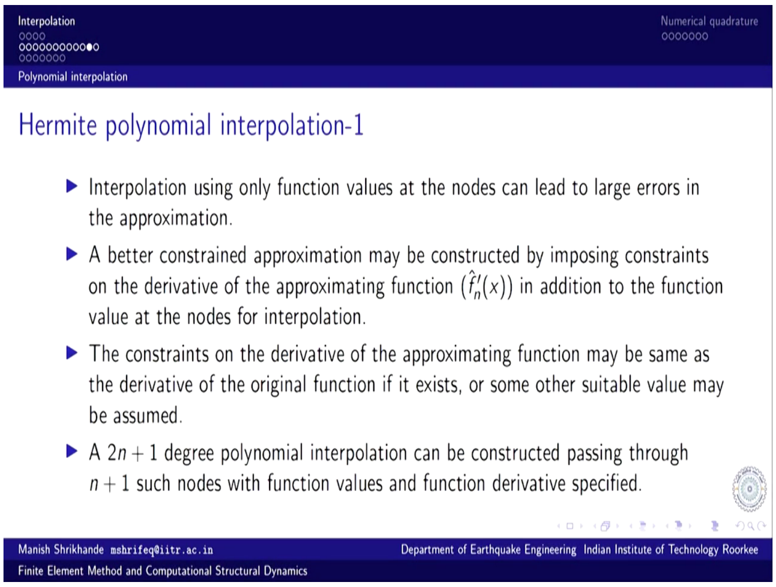
Finite Element Method and Computational Structural Dynamics

So, what are Chebyshev polynomials? Chebyshev polynomials are a family of orthogonal functions those are defined for x between minus 1 to plus 1 bounded range. Now, that can be scaled I mean these functions can be scaled and the roots can be accordingly scaled up outside the range. And they can be generated recursively from the lower degree terms. So, first two members of the family are simply 1 and x. So, T_0 the constant term 1, and T_1 is a simple x term. And higher degree terms are related to two preceding lower degree terms as T_n nth degree Chebyshev polynomial is derived from n +1th degree polynomial and n - 2th degree polynomial as twice T_{n-1} - T_{n-2}.

So, that is a Chebyshev polynomial has its own theory and very interesting development, but that is just not much of interest for our purpose as far as nodes of a location of nodes of interpolation are concerned. So, coming back to the Runge's function and approximation that you see here, so the other function non-uniform 10th degree polynomial approximation, non-uniform sampling points those non-uniform sampling points correspond to the location of roots of Chebyshev polynomial. And if I use those roots of Chebyshev polynomials, you can see that the polynomial interpolation 10th degree polynomial of interpolation is reasonably well behaved. And it can be and certainly pass off as a reasonably good approximation for Runge's function. And the same 10th degree polynomial using uniform sampling points.

And the sampling points are not very different I mean they are this very slight shift in sampling points. And the effect that they have on the quality of approximation is just amazing. So, the location of nodes of interpolation is very, very crucial. And the same thing , I would urge all of you to please keep this picture in mind. And when we construct finite element approximation, we will be discussing about nodes, although we may not be devoting too much time on the mesh generation and what is the best good mesh or how to generate the good quality mesh. You can keep this particular aspect in mind. The location of nodes has a very important implication on the quality of approximation that we develop. So, that is for the polynomial interpolation using only function values.

(Refer Slide Time: 26:57)



Interpolation | Numerical quadrature

Polynomial interpolation

Hermite polynomial interpolation-1

- Interpolation using only function values at the nodes can lead to large errors in the approximation.
- A better constrained approximation may be constructed by imposing constraints on the derivative of the approximating function $(\hat{f}_n'(x))$ in addition to the function value at the nodes for interpolation.
- The constraints on the derivative of the approximating function may be same as the derivative of the original function if it exists, or some other suitable value may be assumed.
- A $2n+1$ degree polynomial interpolation can be constructed passing through $n+1$ such nodes with function values and function derivative specified.

Manish Shrikhande mshrifeq@iitr.ac.in | Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

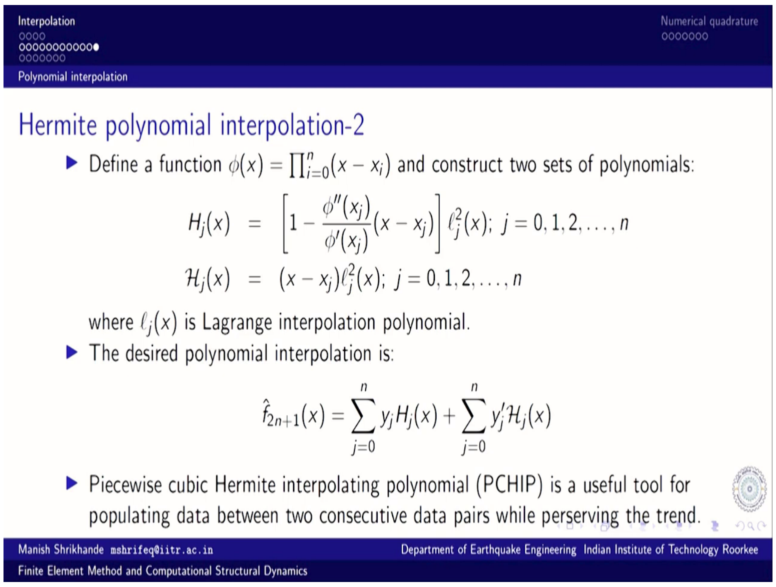
So, as we saw in the approximation of Runge’s function, if we concentrate only on the function values, it can we can possibly have a very large errors in between the nodes because there is no constraint on the derivative. There is no constraint on how the approximation function approaches the function value at the nodes.

So, if we improve try to improve this situation by constraining not only the function values, but also the derivative functions then that model is referred to as Hermite polynomial interpolation. So, a better constraint approximation can be constructed by imposing constraints on the derivatives of the approximation function in addition to the functional values or function value at the nodes for interpolation.

So, at the nodes, we look at not only the function values, but also the derivative of the functions. So, slopes of the function. So, constraints on the derivative of the approximating function may be same as the derivative of the original function if it exists or some other suitable value may be assumed. So, if the derivative is already defined, then it can be used or we can impose some other we may derive we may impose something which may be able to provide a better constraint on the approximation.

Now we look at this situation if there are n number of nodes, then if there are function values and the derivative that is available at each node, then we are looking at 2n number of terms 2n number of constraints conditions are available. And if there are n +1 number of nodes, then I am looking at 2n + 2, a number of constraints conditions that are available. And with these 2 n + 2 number of conditions available, I can evaluate 2 n plus 2 number of coefficients and 2 n + 1 or 2 n + 2 number of coefficients would be sufficient to uniquely define a 2 n + 1 degree polynomial interpolation and that is how we work out. And obviously, the formulation is little bit more tedious more involved compared to simple function interpolation, but it is very easy to formulate and for construction.

(Refer Slide Time: 29:40)



Interpolation | Numerical quadrature

Polynomial interpolation

Hermite polynomial interpolation-2

- Define a function $\phi(x) = \prod_{i=0}^{n}(x - x_i)$ and construct two sets of polynomials:

$$H_j(x) = \left[1 - \frac{\phi''(x_j)}{\phi'(x_j)}(x - x_j)\right] \ell_j^2(x); \; j = 0, 1, 2, \ldots, n$$

$$\mathcal{H}_j(x) = (x - x_j)\ell_j^2(x); \; j = 0, 1, 2, \ldots, n$$

where $\ell_j(x)$ is Lagrange interpolation polynomial.

- The desired polynomial interpolation is:

$$\hat{f}_{2n+1}(x) = \sum_{j=0}^{n} y_j H_j(x) + \sum_{j=0}^{n} y_j' \mathcal{H}_j(x)$$

- Piecewise cubic Hermite interpolating polynomial (PCHIP) is a useful tool for populating data between two consecutive data pairs while perserving the trend.

Manish Shrikhande mshrifeq@iitr.ac.in | Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

And we can define a function ϕ as a similar to what we have as the roots of the nodes $\phi(x)$ as a function n + 1th degree polynomial $x - x_i$ product of $x - x_i$ terms. And then construct two sets of polynomials based on the derivatives and the corresponding Lagrangian interpolation.

And then we can combine these and we can have the suitable polynomial interpolation based on the linear combination of these two sets of polynomial functions. And again based on interpolation of first term provides the interpolation of the function values. Second summation provides the interpolation of derivative values at the nodes.

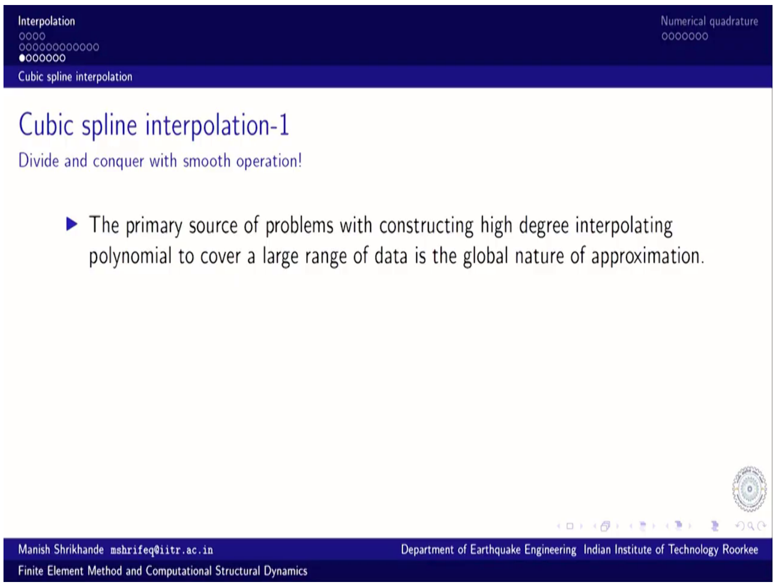
And together this complete interpolation model provides a better model for function approximation which is relatively smoother than the interpolation model derived only on the basis of function values. So, you can see that if I use a two point to two node model, the Lagrangian interpolation or simple polynomial interpolation would fetch me only a first degree polynomial interpolation.

Whereas, if I use Hermite polynomial, then I am looking at four constraints for two nodes and then I can actually look at a cubic interpolation polynomial between two nodes. So, the higher degree of smoothness of the function approximation is obviously, provides a advantage in certain cases.

And of course, it comes at the cost of increased computations. So, wherever it is justified it is a good model to adopt. So, piece wise cubic Hermite interpolating polynomial is a useful tool for populating data between two consecutive data points, two consecutive data pairs while preserving the trend that is another interesting property of piece wise cubic Hermite interpolation.

Cubic Hermite interpolation as I said between two nodes. So, we can have a cubic polynomial. Now, piece wise cubic polynomial has this very useful property that it does not change the trend of data between the interpolation, so that has a very useful property, and it can be a harnessed for wherever the need be.

(Refer Slide Time: 32:58)


Interpolation

Numerical quadrature

Cubic spline interpolation

Cubic spline interpolation-1

Divide and conquer with smooth operation!

- The primary source of problems with constructing high degree interpolating polynomial to cover a large range of data is the global nature of approximation.

Manish Shrikhande mshrifeq@iitr.ac.in Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

So, another way of interpolation is the cubic spline interpolation, and that we will discuss in our next lecture.

Thank you.