Lecture 7: Newton Interpolation

Instructor: Professor Amos Ron Scribes: Yunpeng Li, Mark Cowlishaw, Nathanael Fillmore

1 Motivation for Newton interpolation

Recall the polynomial interpolation problem we have been studying for the last few lectures. Given $\vec{X} = (x_0, x_1, \cdots, x_n)$ and the value of f at the given points $\vec{F} = (f(x_0), f(x_1), \cdots, f(x_n))$, find the polynomial $p \in \Pi_n$ so that $p|_{\vec{X}} = \vec{F}$. We proved in the previous lecture that the solution to this problem always exists and is unique.

We introduced two methods for solving the polynomial interpolation problem, based on two different representations of the polynomial solution:

1. We can represent the solution as a linear combination of monomials.

$$P(t) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

   Using this representation, the problem is equivalent to solving a system of linear equations

$$V\vec{a} = \vec{F}$$

   Where V is the Vandermonde matrix, $\vec{a}$ is the column vector of (unknown) polynomial coeffcients, and $\vec{F}$ is the column vector of function values at the interpolation points.

   As we have seen, solving this system is not a good method for solving the polynomial interpolation problem. However, this representation is important, since we were able to use it to conclude that the solution to a polynomial interpolation problem is unique.

2. We can represent the solution using Lagrange polynomials.

$$P(t) = f(x_0) \cdot \ell_0(t) + f(x_1) \cdot \ell_1(t) + \cdots + f(x_n) \cdot \ell_n(t)$$

   Using this representation, we showed that it is easy to find a solution p, and this led to the important conclusion that the polynomial interpolation problem always has a soluton. However, a solution written in Lagrange form may be difficult to use.

1.1 Problems with the Lagrange Representation

Our goal when solving a polynomial interpolation problem is to find a polynomial function p that approximates the function f, which we may know nothing about, other than its value at a few points. Once we have this polynomial function p, we may want to use it to study f in various ways.

Evaluation We may want to use p to approximate f at points outside the set of interpolation points. The Lagrange representation may be problematic for this purpose if we have interpolation points (x_a, x_b) that are very close together. Recall the formula for a Lagrange polynomial $\ell_i(t)$.

$$\ell_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{(t - x_j)}{(x_i - x_j)} \right)$$

Those Lagrange polynomials that include $(x_a - x_b)$ or $(x_b - x_a)$ in the denominator will be quite large, and may suffer from loss of significance.

Evaluation of Derivative The symbolic differentiation of a Lagrange polynomial, using the product rule, will have n separate terms, each involving $n - 1$ multiplications. Sine the solution to the polynomial interpolation problem p contains $n + 1$ different Lagrange poynomials, evaluating the derivative could be very inefficient.

Clearly, the Lagrange representation is not ideal for some uses of the polynomial p.

Additionally, there is another problem with the Lagrange representation, it doesn't take advantage of the inductive structure of a polynomial interpolation problem. Polynomial interpolation is often an iterative process where we add points one at a time until we reach an acceptable solution. Given a particular solution p to a polynomial interpolation problem with n points, we would like to add an additional point and find a new solution p' without too much additional cost. In other words, we would like to reuse p to help us find p'.

It is easy to see that the Lagrange representation will not meet this requirement. Using the Lagrange representation, we can find a solution to a polynomial interpolation problem with n points $X = (x_0, x_1, \ldots, x_{n-2}, x_{n-1})$ easily, but, if we add a single additional point x_n, we have to throw out our previous solution and recalculate the solution to the new problem from scratch. This is because every interpolation point is used in every term of the Lagrange representation.

We shall now discuss a polynomial representation that makes use of the inductive structure of the problem. However, note that, despite these shortcomings, the Lagrange representation is not a bad way to represent polynomials.

2 Newton Polynomials

We introduce Newton polynomials with a series of examples.

EXAMPLE 2.1. *Given a set of points* $\vec{X} = (1, 3, 0)$ *and the corresponding function values* $F = (2, 7, -8)$*, find a quadratic polynomial that interpolates this data.*

Step 1 The order of the points does not matter (as long as each point is matched to the corresponding function value), but we will consider the first point $X_0 = 1, F_0 = 2$, and find a zero order polynomial p_0 that interpolates this point.

$$N_0(t) = 1$$
$$p_0 = 2 \cdot N_0(t)$$

We call N_0 the zero order Newton polynomial.

Step 2 We use our previous solution p_0 and the first order Newton polynomial N_1 to interpolate the first two points $(1, 2)$ and $(3, 7)$.

$$N_1(t) = (t - 1)$$
$$p_1(t) = p_0(t) + \square \cdot N_1(t)$$

We need to find the value of $\square$ given the two constraints that $p_1(1) = 2$ and $p_1(3) = 7$.

$$\begin{aligned} p_1(1) &= p_0(1) + \square \cdot (1 - 1) = 2 + 0 = 2 \\ p_1(3) &= p_0(3) + \square \cdot (3 - 1) \\ 7 &= 2 + \square \cdot 2 \\ \square &= 5/2 \end{aligned}$$

Thus, $p_1(t) = p_0(t) + 5/2 \cdot N_1(t)$.

Step 3 We use our previous solution p_1 and the second order Newton polynomial N_2 to interpolate all three points.

$$\begin{aligned} N_2(t) &= (t - 1) \cdot (t - 3) \\ P_2(t) &= p_1(t) + \square \cdot N_2(t) \end{aligned}$$

Solving for the three constraints $(1, 2)$, $(3, 7)$, and $(0, -8)$, we obtain:

$$\begin{aligned} p_2(1) &= p_1(1) + \square \cdot 0 \check{} \\ p_2(3) &= p_1(3) + \square \cdot 0 \check{} \\ p_2(0) &= p_1(0) + \square \cdot 3 \\ -8 &= -1/2 + \square \cdot 3 \\ \square &= -5/2 \end{aligned}$$

Yielding the final solution, $p_2(t) = -5/2 \cdot N_2(t)$

To solve polynomial interpolation problems using this iterative method, we need the polynomials $N_i(t)$. We call these *Newton polynomials.*

Definition 2.1 (Newton Polynomials). *Given a set of $n + 1$ input points $\vec{X} = (X_0, X_1, \ldots, X_n)$, we define the $n + 1$ Newton polynomials.*

$$\begin{aligned} N_0(t) &= 1 \\ N_1(t) &= t - X_0 \\ \vdots &\quad \vdots \\ N_n(t) &= (t - X_0) \cdot (t - X_1) \cdot \cdots \cdot (t - X_{n-1}) \end{aligned}$$

Note that the Newton polynomials for a polynomial interpolation depend only on the input points $\vec{X} = (X_0, X_1, \ldots, x_n)$, and not on the associated function values $\vec{F} = (f(X_0), f(X_1), \ldots, f(X_n)$. Given these Newton polynomials, we can define a recurrence for $p_n(t)$:

$$p_n(t) = p_{n-1}(t) + \square \cdot N_n(t) \tag{1}$$

At first, the basis of Newton polynomials doesn't look any better than Lagrange in terms of computation: $N_n(t)$ has n factors, similar to the number of factors in a Lagrange polynomial. This is a valid concern; however, as we will see in future lectures, it turns out that there are very efficient linear-time algorithms to both evaluate and find derivatives of polynomials in the Newton basis.

We can find the coefficient $\square$ for each Newton polynomial using the method of *divided differences*.

3 Divided Differences

Definition 3.1 (Divided Differences). *Given a set of $n+1$ input points $\vec{X} = (X_0, X_1, \ldots, X_n)$, and the corresponding function values $\vec{F} = (f(X_0), f(X_1), \ldots, f(X_n))$, and the Newton representation of the polynomial interpolant $p_n(t) = a_0N_0(t) + a_1N_1(t) + \cdots + a_nN_n(t)$, the coefficient a_n of the nth Newton polynomial $N_n(t)$ is called the divided difference (or D.D.) of f at $(X_0, X_1, \ldots, X_n)$ and is denoted by $f[X_0, X_1, \ldots, X_n]$*

Example 3.1. *Given $\vec{X} = (1, 3, 0)$ and $\vec{F} = (2, 7, -8)$, then*

$$p_0(t) = f[1] \cdot N_0(t) \tag{2}$$
$$p_1(t) = p_0(t) + f[1, 3] \cdot N_1(t) \tag{3}$$
$$p_2(t) = p_1(t) + f[1, 3, 0] \cdot N_2(t) \tag{4}$$

Note that a polynomial interpolation problem is invariant over the ordering of the input points. If we rewrote $P_2(t)$ in equation 4 in monomial form, the coefficient of t^2 would be the divided difference $f[1, 3, 0]$. If we were to reorder the input points as $(3, 0, 1)$ and solve the problem using Newton polynomials, the coefficient of t^2 would be $f[3, 0, 1]$. Since these problems are equivalent, this means that $f[1, 3, 0] = f[3, 0, 1]$. In general, the final coefficnet in the Newton representation of a polynomial interpolant is always the same under any ordering of the input points. This means that the divided difference $f[x_0, x_1, \ldots, x_n]$ is invariant over the ordering of the points $(x_0, x_1, \ldots, x_n)$.

Finally, we define the order of a divided difference.

Definition 3.2. *The order of a divided difference $f[X_0, X_1, \ldots, X_n]$ is one less then the number of points in the divided difference.*

A divided difference of order k, $f(X_0, X_1, \ldots, X_k)$ is the coefficient of the final (order k) Newton polynomial in the interpolant of $(X_0, X_1, \ldots, X_k)$.

3.1 Computing Divided Differences

Given a polynomial interpolation problem $\vec{X} = (X_0, X_1, \ldots, X_n)$, $\vec{F} = (f(X_0), f(X_1), \ldots, f(X_n))$, we compute the divided differences by constructing a special half-table, with the rows indexed by the input points $\vec{X}$ and the columns indexed by the order of the divided difference. The values in

the first column (column 0) are simply the function values $\vec{F}$. Each successive value $D[i,j]$ (the ith row and jth column of the table) is calculated as a quotient. The numerator is the difference of the values immediately left ($D[i,j-1]$) and immediately on the downward left diagonal ($D[i+1,j-1]$). The denominator is the difference between the input value x_i labeling the ith row, and the input value x_{i+j} labeling the $i+j$th row. More formally, given a polynomial interpolation problem with $n+1$ input points, we define the following table D:

x_0	$D[0,0]$	$D[0,1]$	$\dots$	$D[0,N]$
x_1	$D[1,0]$	$\dots$	$D[1,N-1]$	
	$\vdots$			
x_N	$D[N,0]$			

we can calculate the individual values $D[i,j]$ in the table D as follows.

$$f[x_i, x_{i+j}] = D[i,j] = \begin{cases} f(x_i) & \text{if } j = 0 \\ \frac{D[i,j-1]-D[i+1,j-1]]}{x_i - x_{i+j}} & \text{Otherwise} \end{cases} \tag{5}$$

For example, if we are given $\vec{X} = (2,6,7,0)$ and $\vec{F} = (1,-1,0,2)$, we generate the divided difference array D as

2	1	$\frac{1-(-1)}{2-6} = -0.5$	$\frac{3}{10}$	$\frac{3}{70}$
6	-1	$\frac{-1-0}{6-7} = 1$	$\frac{3}{14}$	
7	0	$\frac{0-2}{7-0} = \frac{-2}{7}$		
0	2			

Note that the first row is $f[2], f[2,6], f[2,6,7]$ and $f[2,6,7,0]$. In general, we only need the first row to construct the Newton representation of the polynomial interpolant.

EXAMPLE 3.2. *For $\vec{X} = (1,0,-1)$ and $\vec{F} = (2,4,8)$, we create the divided difference table as*

1	*2*	*-2*	*1*
0	*4*	*-4*	
-1	*8*		

and we have

$$p(t) = 2 \cdot 1 - 2 \cdot (t-1) + 1 \cdot (t-1)(t-0)$$

as the solution.

Now when we add one more point and value, say, $\vec{X} = (1,0,-1,2)$ and $\vec{F} = (2,4,8,2)$, we construct a new array, with the above array as the upper left corner.

1	2	-2	1
0	4	-4	
-1	8		

$\Rightarrow$

1	2	-2	1	-2
0	4	-4	-1	
-1	8	-2		
2	-2			