Lecture 8: Polynomial Interpolation: Using Newton Polynomials and Error Analysis

Instructor: Professor Amos Ron Scribes: Giordano Fusco, Mark Cowlishaw, Nathanael Fillmore

1 Review of Newton Polynomials

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 showed in the previous lecture that the solution to this problem can be written as a linear combination of Newton polynomials.

$$p_n(t) = f[x_0] \cdot N_0(t) + f[x_0, x_1] \cdot N_1(t) + \cdots + f[x_0, x_1, \ldots, x_n] \cdot N_n(t)$$

Where the polynomials $N_i(t)$ are the Newton polynomials.

$$N_i(t) = \prod_{j=0}^{i-1} (t - x_j)$$

Note that here, the empty product, corresponding to $N_0(t)$ is 1. Recall that the divided differences $f[x_0, x_1, \ldots, x_i]$ are defined using a table.

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

Where the first row of the table corresponds to the divided differences $f[x_0], f[x_0, x_1], \ldots, f[x_0, \ldots, x_n]$, and each cell of the table is computed as follows.

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

Note that the diagonal in this table, corresponding to the last entry of each row ($D[i, n-i]$ for $i = 0, 1, \ldots, n$) computes the divided differences that we would calculate if the interpolation points $\vec{X} = (x_0, x_1, \ldots, x_n)$ were given in *reverse* order (i.e. $(x_n, x_{n-1}, \ldots, x_1, x_0)$). As always, the final divided difference (the coefficient of the nth order Newton polynomial) is the same no matter the order of the points.

$$f[x_0, x_1, \ldots, x_{n-1}, x_n] = f[x_n, x_1, \ldots, x_{n-1}, x_n] = D[0, n]$$

(In general, $D[i,j] = f[x_i, x_{i+1}, \ldots, x_{i+j}]$ is the coefficient of the highest degree term of any jth degree polynomial that interpolates the points $(x_i, f(x_i)), (x_{i+1}, f(x_{i+1})), \ldots, (x_{i+j}, f(x_{i+j}))$.) To illustrate, consider the following example.

EXAMPLE 1.1. *Given the input points* $\vec{X} = (0, 1, 2)$ *and corresponding function values* $\vec{F} = (2, -2, 0)$ *find the polynomial interpolant* $p_2 \in \Pi_2$.

To solve this problem using Newton polynomials, we build the following divided difference table.

0	2	-4	3
1	-2	2	
2	0		

The first row corresponds to the divided differences $f[0], f[0,1], f[0,1,2]$, which are the divided differences we would calculate when building the solution in the order $\vec{X} = (0, 1, 2)$.

$$p_2(t) \quad = 2 \cdot 1 - 4 \cdot (t - 0) + 3 \cdot (t - 0)(t - 1)$$

The diagonal (consisting of the last entry in each row) corresponds to the divided differences $f[2], f[1,2], f[0,1,2]$, which are the divided differences we would calculate when building the solution in the order $\vec{X} = (2, 1, 0)$.

$$p_2(t) \quad = 0 \cdot 1 + 2 \cdot (t - 2) + 3 \cdot (t - 2)(t - 1)$$

It is easy to verify that these polynomials are equivalent.

2 Access and Manipulation of Polynomials in Newton Form

We would like to show that Newton polynomials have properties desirable for numerical analysis, particularly:

1. Newton polynomials are easy to evaluate. That is, the computational cost of evaluating the polynomial at a point does not grow too quickly with respect to the degree.
2. Newton Polynomials are easy to differentiate. That is, the computational cost of evaluating the derivative at a point does not grow too large with respect to the degree of the polynomial.

2.1 Evaluating Newton Polynomials

Consider the following example.

EXAMPLE 2.1. What is the value of the polynomial

$$P(t) = 2 + 3(t - 4) - 5(t - 4)(t + 7)$$

at the point $t = 11$?

The parameters of this problem are:

$$\begin{array}{rr} 2 \textit{ centers}: & 4, -7 \\ \text{and } 3 \textit{ coefficients}: & \overline{2, 3, -5} \end{array}$$

Note that the number of coefficients is always one more than the number of centers. The order in which centers and coefficients appear is important.

We are going to show that it is inefficient to simply plug in the numbers and compute the operations. In particular we are going to see how the number of operations grows as size of the polynomial increases.

The following table shows the number of multiplications required for each term.

polynomial:	$P(t) = 2$	$+ 3(t-4)$	$- 5(t-4)(t+7)$	
number of multiplications:	0	1	2	

In general, when the degree of the polynomial is n we have

$$0 + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

multiplications. Note that we are counting only the number of multiplication because multiplications are far more computationally expensive than additions, so this gives a good estimate of the complexity of the evaluation.

This "brute force" method is highly inefficient. In fact suppose that we add a new term

$$P(t) = 2 + 3(t-4) - 5(t-4)(t+7) + 6(t-4)(t+7)(t-8)$$

then we would need three more multiplications. However, we can observe that $(t-4)$ appears as a factor in each term and $(t+7)$ is a factor in most of the terms. We can make use of this fact to rewrite the polynomial as

$$P(t) = 2 + (t-4)[3 - 5(t+7)]$$

Notice that $P_1(t) = 3 - 5(t+7)$ is a Newton polynomial itself. The parameters of $P_1(t)$ are the same as before, but crossing out the leftmost ones

$$\begin{array}{r} \cancel{4},\ -7 \\ \hline \cancel{2},\ 3,\ -5 \end{array}$$

We can rewrite the original polynomial as $P_2(t) = 2 + (t-4)P_1(t)$. This did not happen by accident. In fact suppose we have one more term

$$\begin{array}{r} 4,\ -7,\ 8 \\ \hline 2,\ 3,\ -5,\ 6 \end{array}$$

Now the whole polynomial would be $P_3(t) = 2 + (t-4)P_2(t)$ (i.e. only the index of the polynomial is shifted). This indicates that each additional term requires just one more multiplication. So the number of multiplications is linear in the number of terms (and, at worst, linear in the degree of the polynomial). □

Now we are ready to consider the general case. The Newton polynomial of degree n is

$$P_n(t) = c_0 + c_1(t - x_0) + \ldots + c_n(t - x_0)\cdots(t - x_n)$$

and its parameters are

$$\begin{array}{ll} n \textit{ centers}: & x_0,\ x_1,\ \ldots,\ x_{n-1} \\ \text{and } n+1 \textit{ coefficients}: & \hline c_0,\ c_1,\ c_2,\ \ldots,\ c_n \end{array}$$

If we delete the first two parameters we have that the parameters of

$$P_{n-1}(t) = c_1 + c_2(t - x_1) + \ldots + c_n(t - x_1)\cdots(t - x_{n-1})$$

are

$$\begin{array}{r} x_1,\ \ldots,\ x_{n-1} \\ \hline c_1,\ c_2,\ \ldots,\ c_n \end{array}$$

In general

$$P_{n-j}(t) = c_j + c_{j+1}(t - x_j) + \ldots + c_n(t - x_j)\cdots(t - x_{n-1})$$

has parameters

$$\frac{x_j, \quad \ldots, x_{n-1}}{c_j, c_{j+1}, \ldots, c_n}$$

The algorithm to evaluate the polynomial is

1. $P_0(t) = c_n$
2. `for` $j = n - 1, \ldots, 0$
3. $P_{n-j+1}(t) = c_j + (t - x_j)P_{n-j-1}(t)$

We call this technique "*nested multiplication*".

EXAMPLE 2.2. *Consider the Newton polynomial* $P(t) = 3 - 2(t - 5) + 7(t - 5)(t + 11)$ *whose parameters are*

$$\frac{x_0 = 5, \quad x_1 = -11}{c_0 = 3, c_1 = -2, c_2 = 7}$$

We want to evaluate $P(t)$ *at* $t = 4$*, which means we want to compute* $P_2(4)$*. The steps are*

1. $P_0(4) = 7$
2. $P_1(4) = -2 + (4 + 11)7 = 103$
3. $P_2(4) = 3 + (4 - 5)103 = -100$

2.2 Differentiating Newton Polynomials

We are going to show how to evaluate the derivative of a Newton polynomial efficiently. Consider the following example.

EXAMPLE 2.3. *Find the derivative of the following cubic polynomial*

$$P(t) = 3 - 2(t - 5) + 7(t - 5)(t + 11) + 9(t - 5)(t + 11)(t - 1)$$

and evaluate it at a certain point.

Just finding the derivative symbolically could be difficult if we use the product rule. In fact, for the last term we would have to take the derivative of $(t - 5)$ and multiply by the other 2 terms, then add the derivative of $(t + 11)$ times the other 2 terms, and so on... You can imagine that as the degree of the polynomial grows, this task quickly becomes quite difficult (or at least time-consuming).

Fortunately, there is a method that makes this task easier. The idea is to look at the recurrence relation defining the Newton representation and look at what happens when we differentiate it. Recall that

$$P_n(t) = c_0 + (t - x_0)P_{n-1}(t)$$

When c_0 is the first coefficient, and x_0 the first center. Differentiating on both sides we get

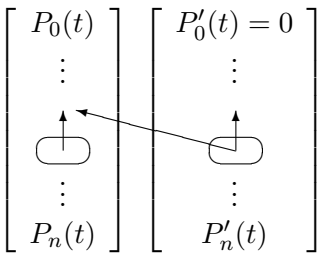
$$P'_n(t) = P_{n-1}(t) + (t - x_0)P'_{n-1}(t)$$

We can easily compute $P'_n(t)$ if we already know $P_{n-1}(t)$ and $P'_{n-1}(t)$.

The algorithm is

1. $P_0(t) = c_n$
2. $P_0'(t) = 0$
3. `for` $j = n-1, \ldots, 0$
4. $\quad P_{n-j+1}(t) = c_j + (t - x_j)P_{n-j-1}(t)$
5. $\quad P'_{n-j}(t) = P_{n-j-1}(t) + (t - x_j)P'_{n-j-1}(t)$

In other words, we are building the following two vectors starting from the top. The arrows indicate the dependences.



Note that the coefficients do not appear explicitly in the formula of the derivatives (vector on the right). However, the coefficients are used in the computation of the polynomials (vector on the left), and the polynomials are used in the computation of the derivatives. So the derivatives still depend on the coefficients.

Note that this method is very efficient. Once we have calculated the values of $P_i(t)$ (at a cost of n multiplications), we need only one multiplication per term to evaluate the derivative, for a total of $2n$ multiplications.

On the other hand, it is not easy to integrate Newton polynomials, but later in the course we will see some efficient methods for integrating functions and these methods will work for Newton polynomials.

We end with another example.

EXAMPLE 2.4. *Evaluate p(t) = 3 + 2(t-1) + 4(t-1)(t-7) and its derivative.*

Nodes:		*1*	*7*
Coefficients:	*3*	*2*	*4*

Thus:

$p_0(t) = 4$	$p_0'(t) = 0$
$p_1(t) = 2 + (t-7)p_0(t)$	$p_1'(t) = p_0(t) + (t-7)p_0'(t)$
$p_2(t) = 3 + (t-1)p_1(t)$	$p_2'(t) = p_1(t) + (t-1)p_1'(t)$

So if, say, $t = 1$, we just plug 1 into the expressions above.

3 Error Analysis of Polynomial Interpolation

It is very important to understand the error in polynomial interpolation. Given a function $f : [a, b] \to \mathbb{R}$ and a set of points $X := \{x_0, \ldots, x_n\} \subset [a, b]$, a polynomial interpolant has the property that the value of the polynomial at the points X is the same as the function. We denote this as

$$P|_X = f|_X$$

Note that the error of our polynomial P with respect to the true function f is defined as $e(t) := |f(t) - P(t)|$ for all $t \in [a, b]$. Ultimately, we are going to give an upper bound for the error.

Of course, it is easy to bound the error very loosely, for example it is easy to see that $e(t) \leq \max_{x\in[a,b]} |f(x)| + \max_{x\in[a,b]} |P(x)|$ holds for any $t \in [a, b]$. However, this bound may be much larger than the true error - we want a tighter estimate of the error, and one that gives us more insight into the sources of error.

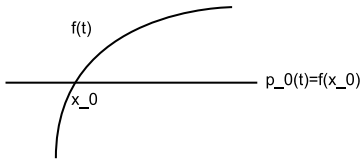


Figure 1: Interpolating one point

What do we expect, therefore, of a "good" error formula?

1. First, we would expect that $e(x_i) = 0$ for all our interpolation points $x_i \in X$.
2. Second, consider interpolating a function f based on one point $(x_0, f(x_0))$, as in Figure 1. Of course, there will always be some error in this case, unless f is itself constant. If the function f moves very fast, then the error will be big. If the function f moves slowly, then there will be less error. The derivative is what controls this. So we require that $|f'|$ not be too large.

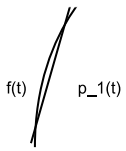


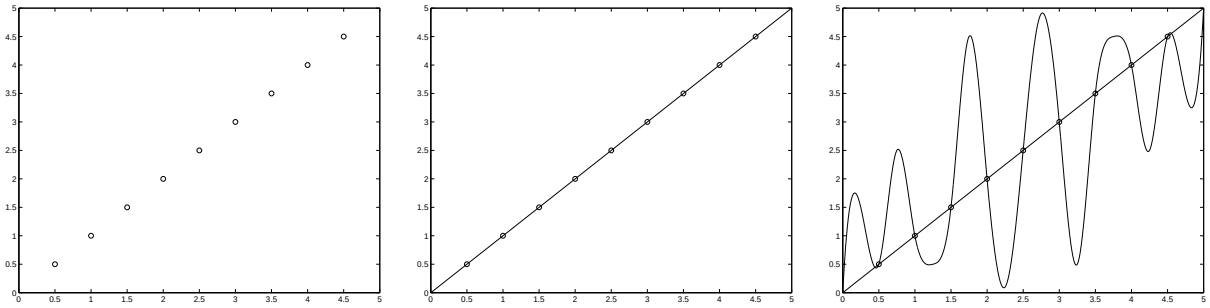
Figure 2: Interpolating two points

Consider next interpolating a function f based on two points, as in Figure 2. Here the error doesn't look too bad, even though $|f'|$ is rather large, because the slope of our interpolating polynomial p_1 is also large. So here we need to look at the second derivative, f''.

So we can guess that a good error formula if we interpolate $n+1$ points will somehow involve (1) $(t - x_i)$ for all points of interpolation x_i, and (2) the $(n+1)$st derivative.

We can see this in another way from the examples below.

EXAMPLE 3.1. *The main problem is that we can always "cheat". Suppose that we are given the points of Figure 3a. The points are aligned and the solution is the linear polynomial of Figure 3b (note that for aligned points the polynomial is linear regardless of the number of points). However, knowing the points, we could decide to choose a very bad function for which polynomial interpolation behaves extremely poorly, as in Figure 3c.*



(a) (b) (c)

Figure 3: Example of bad polynomial interpolation

The way to recognize this kind of function is to look at the derivative. The function of the previous example has a derivative that changes frequently. Now, consider another example

EXAMPLE 3.2. *The first derivative of the function in Figure 4 changes frequently, but the interpolation works pretty well in this case.*

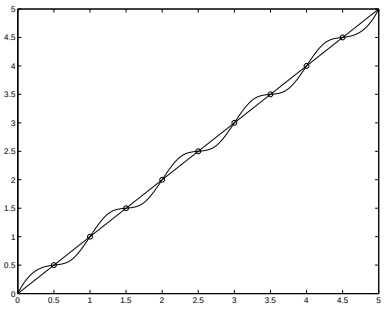


Figure 4: Example of good polynomial interpolation

From example 3.2 we can see that looking only at the first derivative we might not get enough information. In fact, the general formula for calculating the error of a polynomial interpolant P is given by

$$f(t) - P(t) = \frac{f^{(n-1)}(c)}{(n+1)!}(t - x_0)(t - x_1)\cdots(t - x_n)$$

where c lies somewhere in the smallest interval containing $X \cup \{t\}$.

Note that to do the error analysis we do not need to know the interpolating polynomial. We just need to know the derivative of the function and the position of the points. Also note that we are not going to compute the exact value of the error, but we are looking for a good bound.

EXAMPLE 3.3. *Suppose that we are given the function* $f(t) = \sin t$ *and the points* $X = (.5, 1.5, 2.5)$. *We want to estimate* $f(1) - p(1)$, *i.e., we want to estimate the error at the point* $t = 1$.

Using the error formula, we calculate that

$$f(1) - P(1) = \frac{f^{(3)}(c_t)}{3!}(t - .5)(t - 1.5)(t - 2.5)$$

where the point c_t *depends on the value of* t. *In other words we don't know exactly where* c_t *is, but we know that it lies in the interval containing* X. *In this case we consider* $c_t \in [.5, 2.5]$.

The 3*rd derivative of* $\sin t$ *is* $-\cos t$, *which is bounded in absolute value by 1, so*

$$|f^{(3)}(c_t)| \leq 1$$

So we can estimate that

$$|f(1) - P(1)| \leq \left|\frac{1}{3!}(1 - .5)(1 - 1.5)(1 - 2.5)\right| = \frac{1}{16}$$

Note that, since we do not know the precise value of c_t, this is almost certainly an overestimate of the exact error. However, without more knowledge, this is the best we can do.