LECTURE 17

Algorithms for Polynomial Interpolation

We have thus far three algorithms for determining the polynomial interpolation of a given set of data.

1. Brute Force Method. Set
$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
and solve the following set of $n+1$ equations
$$\begin{aligned} a_n (x_0)^n + a_{n-1} (x_0)^{n-1} + \cdots + a_1 x_0 + a_0 &= y_0 \\ a_n (x_1)^n + a_{n-1} (x_1)^{n-1} + \cdots + a_1 x_1 + a_0 &= y_1 \\ &\vdots \\ a_n (x_n)^n + a_{n-1} (x_n)^{n-1} + \cdots + a_1 x_n + a_0 &= y_n \end{aligned}$$
for the $n+1$ coefficients.
2. Newton Form Method. Set
$$P(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n (x - x_0)(x - x_1) \cdots (x - x_{n-1})$$
and use the following recursive formulae to determine the coefficients c_k:
$$\begin{aligned} c_k &= \frac{y_k - P_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})} \\ P_k(x) &= P_{k-1}(x) + c_k (x - x_0)(x - x_1) \cdots (x - x_{k-1}) \end{aligned}$$
3. Lagrange Form Method. For $k = 0, 1, \ldots, n$ compute the cardinal functions
$$\ell_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{(x - x_j)}{(x_k - x_j)} = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}$$
and then set
$$P(x) = \sum_{k=0}^{n} y_k \ell_k(x)$$

There is one more method that is, computationally, much more efficient than any of the algorithms above. This is the so-called method of **divided differences**.

NOTATION 17.1. Let $\{(x_i, y_i) \mid i = 0, \ldots, n\}$ be an ordered set of $n+1$ data points, let $x_k, x_{k+1}, \ldots, x_{k+j}$ be any set of j consecutive nodes and let

$$\mathcal{P}_k(x) = c_{k,0} + c_{k,1}(x - x_k) + c_{k,2}(x - x_k)(x - x_{k+1}) + \cdots + c_{k,j}(x - x_k)(x - x_{k+1}) \cdots (x - x_{k+j-2})(x - x_{k+j-1})$$

be the Newton form of the interpolation polynomial for $\{(x_i, y_j) \mid i = k, \ldots, k+j\}$. The **divided difference**
$$f[x_k, x_{k+1}, \ldots, x_{k+j}]$$
is the highest order coefficient $c_{k,j}$ of $\mathcal{P}_k(x)$.

REMARK 17.2. Note that if we set $k = 0$ then the divided differences $f[x_0, x_1, x_j]$ are just the coefficients c_j appearing in the Newton of the interpolation polynomial for $\{(x_i, y_i) \mid i = 0, \ldots, n\}$. For in this case, the polynomials

$$P_j(x) = P_{j-1}(x) + c_j (x - x_0)(x - x_1) \cdots (x - x_{j-1})$$

and

$$\mathcal{P}_0(x) = c_{k,0} + c_{k,1}(x - x_0) + c_{k,2}(x - x_0)(x - x_1)$$

$$\begin{aligned}
\mathcal{P}_0(x) &= c_{k,0} + c_{k,1}(x - x_0) + c_{k,2}(x - x_0)(x - x_1) + \cdots \\
&\quad \cdots + c_{0,j}(x - x_0) \cdots (x - x_{j-1}) + \cdots + c_{0,n}(x - x_0) \cdots (x - x_{n-1}) \\
&= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\
&\quad \cdots + f[x_0, x_1, \ldots, x_j](x - x_0) \cdots (x - x_{j-1}) + \cdots \\
&\quad \cdots + f[x_0, x_1, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}) \\
&= c_0 + c_1(x - x_0) + c_{2k,2}(x - x_0)(x - x_1) + \cdots \\
&\quad \cdots + c_j (x - x_0) \cdots (x - x_{j-1}) + \ldots \\
&\quad \cdots + c_n (x - x_0) \cdots (x - x_{n-1})
\end{aligned}$$

THEOREM 17.3. *The divided differences satisfy the following recursion relations*

$$f[x_i, x_{i+1}, \ldots, x_{i+j}] = \frac{f[x_{i+1}, x_{i+1}, \ldots, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i}$$

Proof. It suffices to prove this for $i = 0$ and $j = n$: because for any other choice of i or j, we can always construct a new set of data $\{(\tilde{x}_k, \tilde{y}_k) = (x_{i+k}, y_{i+k}) \mid k = 0, 1, \ldots, \tilde{n} = j\}$ for which the relation

$$f[\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_{\tilde{n}}] = \frac{f[\tilde{x}_1, \ldots, \tilde{x}_n] - f[\tilde{x}_1, \ldots, \tilde{x}_{\tilde{n}-1}]}{\tilde{x}_{\tilde{n}} - \tilde{x}_0}$$

is equivalent to the relation in the problem statement. Let $P_{n-1}(x)$ be the polynomial of degree $\leq n - 1$ that interpolates the data at the first n data points $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ and let $Q_{n-1}(x)$ be the polynomial that interpolates the data at the last n data points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Then if we set

$$Q(x) = Q_{n-1}(x) + \frac{x - x_n}{x_n - x_0}(Q_{n-1}(x) - P_{n-1}(x))$$

then

$$Q(x_i) = \begin{cases} Q_{n-1}(x_0) + \frac{x_0 - x_n}{x_n - x_0}(Q_{n-1}(x) - P_{n-1}(x)) = P_{n-1}(x_0) = y_0 \quad , \quad \text{if } i = 0 \\ Q_{n-1}(x_i) + \frac{x_i - x_n}{x_n - x_0}(Q_{n-1}(x_i) - P_{n-1}(x_i)) = y_i - \frac{x_i - x_n}{x_n - x_0}(0) = y_i \quad , \quad \text{if } 0 < i < n \\ Q_{n-1}(x_n) + \frac{x_n - x_n}{x_n - x_0}(Q_{n-1}(x_n) - P_{n-1}(x_n)) = y_n \quad , \quad \text{if } i = n \end{cases}$$

Therefore, $Q(x)$ is the interpolation polynomial $P(x)$ for the data $\{(x_i, y_i) \mid i = 0, \ldots, n\}$. Now the coefficient of the highest power of x for $P(x)$ is

$$P(x) \approx f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_n) \approx f[x_0, \ldots, x_n] x^n + \mathcal{O}(x^{n-1})$$

while the coefficient of the highest power of x for $Q(x)$ will be

$$\begin{aligned}
Q(x) &\approx \frac{x - x_n}{x_n - x_0}((f[x_1, \ldots, x_n](x - x_1) \cdots (x - x_n) - f[x_0, \ldots, x_{n-1}](x - x_0) \cdots (x - x_{n-1})) + O(x^{n-1}) \\
&\approx \left(\frac{f[x_1, \ldots, x_n]}{x_n - x_0} - \frac{f[x_0, \ldots, x_{n-1}]}{x_n - x_0} \right) x^n - \mathcal{O}(x^{n-1})
\end{aligned}$$

Since the highest order coefficients must agree we have

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

Now consider how we might construct the interpolation polynomial using divided differences. Suppose $P(x)$ is the interpolation polynomial for a problem with four data points x_0, x_1, x_2, x_3. We then have

$$\begin{aligned} P(x) &= c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + c_3(x-x_0)(x-x_1)(x-x_2) \\ &= f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) + f[x_0,x_1,x_2,x_3](x-x_0)(x-x_1)(x-x_2) \end{aligned}$$

Now according to the preceding theorem

$$\begin{aligned} f[x_0,x_1,x_2,x_3] &= \frac{f[x_1,x_2,x_3] - f[x_0,x_1,x_2]}{x_3-x_0} \\ &= \frac{1}{x_3-x_0}\left(\frac{f[x_2,x_3]-f[x_1,x_2]]}{x_3-x_1} - \frac{f[x_1,x_2]-f[x_0,x_1]}{x_1-x_0}\right) \\ &= \frac{1}{(x_3-x_0)(x_3-x_1)}\left(\frac{f[x_3]-f[x_2]}{x_3-x_2} - \frac{f[x_2]-f[x_1]}{x_2-x_1}\right) \\ &\quad -\frac{1}{(x_3-x_0)(x_1-x_0)}\left(\frac{f[x_2]-f[x_1]}{x_2-x_1} - \frac{f[x_1]-f[x_0]}{x_1-x_0}\right) \\ &= \frac{1}{(x_3-x_0)(x_3-x_1)}\left(\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}\right) \\ &\quad -\frac{1}{(x_3-x_0)(x_1-x_0)}\left(\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}\right) \end{aligned}$$

However, this is not how we'll want to compute divided differences (by breaking down the complicated $f[x_0,\dots,x_i]$ to the original data). Rather, we'll employ a bottoms up approach. For notational and calculational convenience, we'll set

$$\begin{aligned} F[[i,0] &\equiv f[x_i] = y_i \\ F[i,j] &\equiv f[x_i, x_{i+1}, \dots, x_{i+j}] \end{aligned}$$

so that our recursion relations

$$f[x_i, x_{i+1}, \dots, x_{i+j}] = \frac{f[x_{i+1},\dots,x_{i+j}] - f[x_i,\dots,x_{i+j=1}]}{x_{i+j}-x_i}$$

can be expressed a bit more succinctly as

$$F[i,j] = \frac{F[i+1,j-1] - F[i,j-1]}{x_{i+j}-x_i}$$

- Step 1. Set
$$F[i,0] = y_i \quad , \quad i = 0,\dots,n$$
- Step 2. Set
$$F[i,1] = \frac{F[i+1,0]-F[i,0]}{x_{i+1}-x_i} \quad , \quad i = 0,\dots,n-1$$
- Step 3. Set
$$F[i,2] := \frac{f[i+1,1]-f[i,1]}{x_{i+2}-x_i} \quad , \quad i = 0,\dots,n-2$$
- $\vdots$
- Step $n-1$. Set
$$f[i,\dots,n-1] := \frac{f[i+1,n-2]-f[i,n-2]}{x_n-x_0} \quad , \quad i = 0,1$$
- Step n. Set
$$F[0,n] := \frac{F[1,n-1]-F[0,n-1]}{x_n-x_0} \quad ,$$

- For each i from 0 to n, set

$$c_i = F[0,i] \equiv f[x_0, x_1, \ldots, x_i]$$

- The Newton form of the interpolation polynomial will then be

$$P(x) = \sum_{i=0}^{n} c_i \left(\prod_{j=0}^{i-1} (x - x_j) \right)$$

The following Maple code implements this algorithm.

```
#initialize array F[i,0]
for i from 0 to n do
   F[i,0] := Y[i];
od:

#apply recursion relations
for j from 1 to n do
   for i from 0 to n-j do
      F[i,j] := (F[i+1,j-1] - F[i,j-1])/(X[i+j] - X[i]);
   od:
od:

#construct Newton form of interpolation polynomial
p := F[0,0];  #  = Y[0];
g := 1
for i from 1 to n do
     g := (x-X[i-1])*g
  p := p + F[0,i]*g;
od:

#identify the total coefficient of each power of x

for i from 0 to n do
   c := coeff(p,x,i):
   c := evalf(c,3):
   lprint(`coefficient of x to the`,i,`is`,c):
od:
```

Suppose we implement this code on a test function like

$$T(x) = 2x^8 - 10x^5 - 20x - 50 \quad ,$$

setting up our data points in the following simple-minded way

$$\begin{aligned} x_i &= a + \frac{b-a}{n} i \\ y_i &= T(x_i) \end{aligned}$$

for various choices of n, and intervals $[a,b]$. Surprisingly, we don't get consistent results. For example, for $n = 10$, $[a,b] = [-10, 10]$, we obtain

$$\begin{aligned} P(x) &= (0.0)x^10 + (0.0)x^9 + (2.0)x^8 + (0.0)x^7 + (0.0)x^6 - (10.0)x^5 \\ &\quad + (0.0)x^4 + (0.0)x^3 + (0.0)x^2 + (6.0 \times 10^5)x + (1.0 \times 10^1) \end{aligned}$$

On the other hand, taking $n = 10$, $[a,b] = [-0.005, 0.005]$, we obtain

$$\begin{aligned} P(x) &= -(2.8 \times 10^5)x^{10} + (-1.4 \times 10^5)x^9 + (827)x^8 + (41.3)x^7 - (7.52)x^6 - (10.0)x^5 \\ &\quad +(2.3 \times 10^{-4})x^4 + (1.1 \times 10^{-5})x^3 + (1.6 \times 10^{-8})x^2 - (20.0)x - (50.0) \end{aligned}$$

Of course, such discrepancies can be traced to the floating point errors in the numerical algorithm. The way to get around such inconsistencies is to take a three step approach.

- Carry out an interpolation in a region $[a,b]$ where the data is changing very rapidly to get a handle on the higher degree terms of $P(x)$.
- Carry out an interpolation in a region $[a,b]$ where the rate at which data is changing rather moderately to get a handle on the lower degree terms of $P(x)$.
- Look for a region where the high degree terms and the low degree terms have about the same influence to get a final interpolation for $P(x)$.

Note, however, that the test regions described above need not be near the origin, nor can there sizes be predicted *a priori*. Consider, for example, what might be appropriate regions for interpolating the following polynomial

$$T(x) = 10^{-6}\left[(x-500)^5 - x + 500\right]$$