Lecture 19 and 20

Friday, February 21, 2020

We know that there is only one polynomial of degree $\le n-1$ that fits n points $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$. Consequently, the labeling of points is not important: if we label these n points differently, we still get the same polynomial. This polynomial is called the interpolation polynomial.

There are several ways. We have learned two of them: the direct method (solving a system of linear equations) and Lagrange's formula. Lagrange's approach doesn't require solving any system of equations. Rather, one first establishes a list of n basis polynomials $L_1, L_2, \dots, L_n$. Each is of degree $n-1$ and satisfies

$$L_i(x) = \begin{cases} 1 & \text{if } x = x_i \\ 0 & \text{if } x = x_j, \ j \ne i. \end{cases}$$

These polynomials turn out to have very simple form. If another data point, say (x_{n+1}, y_{n+1}), is added to the data set then all the Lagrange basis functions have to be recomputed: $\tilde{L}_1, \tilde{L}_2, \dots, \tilde{L}_n, \tilde{L}_{n+1}$.

Another method to find the interpolation polynomial is Newton's formula. This method is based on the idea: suppose $P(x)$ is the interpolation polynomial of n points $(x_1,y_1),\dots,(x_n,y_n)$. Let (x_{n+1},y_{n+1}) be another data point. How should we adjust $P(x)$ to get a new interpolation polynomial (of $n+1$ points)?

Let $Q(x)$ be the new interpolation polynomial. Write

$$Q(x) = P(x) + R(x).$$

We want to find $R(x)$. Because both P and Q fit the points (x_1,y_1), $(x_2,y_2),\dots,(x_n,y_n)$, we have

$$Q(x_i) = P(x_i) = y_i \qquad \forall\, i = 1,2,\dots,n.$$

Thus, $R(x_1) = R(x_2) = \dots = R(x_n) = 0$. We can choose

$$R(x) = c(x-x_1)(x-x_2)\dots(x-x_n).$$

Now c has to be chosen suitably such that $Q(x_{n+1}) = y_{n+1}$. This equation can be written as

$$\underline{P}(x_{n+1}) + c(x_{n+1}-x_1)\cdots(x_{n+1}-x_n) = y_{n+1}.$$

From here, we can solve for c. Once we have c, we have $R(x)$. Once we have $R(x)$, we have $Q(x)$.

$$c = \frac{y_{n+1} - \underline{P}(x_{n+1})}{(x_{n+1}-x_1)\cdots(x_{n+1}-x_n)}$$

There is an algorithmic way to find c. We write

$$c = \underbrace{\underline{P}[x_1, x_2, \ldots, x_{n+1}]}_{\text{Called divided difference.}}$$

The divided differences are computed recursively as follows: for any function f whose values are known at $x_1, x_2, \ldots, x_n$, we define

$$f[a] = f(a),$$

$$f[x_1, \ldots, x_{k+1}] = \frac{f[x_2, \ldots, x_{k+1}] - f[x_1, \ldots, x_k]}{x_{k+1} - x_1}$$

Ex: Let f be a function such that $f(1) = 1$, $f(2) = 0$, $f(3) = 2$, $f(4) = -1$.

Find $f[3,2,4]$.

We have

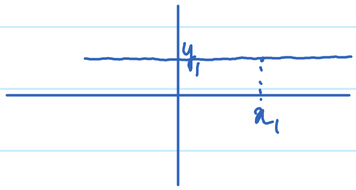
$$f[3,2,4] = \frac{f[2,4] - f[3,2]}{4-3}$$

$$= \frac{\frac{f[4]-f[2]}{4-2} - \frac{f[2]-f[3]}{2-3}}{4-3} = \frac{\frac{f(4)-f(2)}{4-2} - \frac{f(2)-f(3)}{2-3}}{4-3} = -\frac{5}{2}$$

Let $(x_1, y_1), \dots, (x_n, y_n)$ be n data points. Newton's approach goes as follows:

- Find a polynomial that fits only one point (x_1, y_1).

  This is the constant polynomial $P_1(x) = y_1$.



- How to adjust $P_1(x)$ to make it fit also the second point (x_2, y_2)?

$$P_2(x) = P_1(x) + c_1 (x - x_1)$$

where $c_1 = P[x_1, x_2]$.

- How to adjust $P_2(x)$ to make it fit also the third point (x_3, y_3)?

$$P_3(x) = P_2(x) + c_2 (x - x_1)(x - x_2)$$

where $c_2 = P[x_1, x_2, x_3]$.

At the end, we get $P(x) = P_n(x) = c_0 + c_1(x - x_1) + c_2(x - x_1)(x - x_2) + \dots + c_{n-1}(x - x_1)\cdots(x - x_{n-1})$

where $c_k = P[x_1, x_2, \dots, x_{k+1}]$.

There is a trick to find $c_0, c_1, \dots, c_{n-1}$ without using the recursive formula of the divided differences [The recursive formula is suitable for coding, but not very easy to do by hand.]

Ex: Find a polynomial that fits the points $(1,1), (2,1), (3,2), (0,-1)$.

In this problem, $x_1 = 1$, $y_1 = 1$,

$x_2 = 2$, $y_2 = 1$,

$x_3 = 3$, $y_3 = 2$,

$x_4 = 0$, $y_4 = -1$.

We write $x_1, \dots, x_4$ and $y_1, \dots, y_4$ in two columns:

$$\begin{array}{ccccc}
x_1 & \overset{c_0}{y_1} & & & \\
 & & \overset{c_1}{\dfrac{y_1-y_2}{x_1-x_2}} = \alpha & & \\
x_2 & y_2 & & \overset{c_2}{\dfrac{\alpha-\beta}{x_1-x_3}} = \delta & \\
 & & \dfrac{y_2-y_3}{x_2-x_3} = \beta & & \overset{c_3}{\dfrac{\delta-\lambda}{x_1-x_4}} \\
x_3 & y_3 & & \dfrac{\beta-\gamma}{x_2-x_4} = \lambda & \\
 & & \dfrac{y_3-y_4}{x_3-x_4} = \gamma & & \\
x_4 & y_4 & & &
\end{array}$$

In our problem,

$$\begin{array}{ccccc}
1 & \overset{c_0}{1} & & & \\
 & & \dfrac{1-1}{1-2} = \overset{c_1}{0} & & \\
2 & 1 & & \dfrac{1-0}{3-1} = \overset{c_2}{\dfrac{1}{2}} & \\
 & & \dfrac{2-1}{3-2} = 1 & & \dfrac{1/2-0}{1-0} = \overset{c_3}{\dfrac{1}{2}} \\
3 & 2 & & \dfrac{1-0}{2-0} = 0 & \\
 & & \dfrac{2-(-1)}{3-0} = 1 & & \\
0 & -1 & & &
\end{array}$$

Thus, the interpolation polynomial is

$$\begin{aligned} P(x) &= 1 + 0(x-1) + \tfrac{1}{2}(x-1)(x-2) + \tfrac{1}{2}(x-1)(x-2)(x-3) \\ &= \frac{1}{2}x^3 - \frac{5}{2}x^2 + 4x - 1. \end{aligned}$$

* Let us compare the Lagrange's formula with Newton's formula.

In both formulas, the interpolation polynomial is a linear combinations of certain "basis polynomials":

Lagrange basis polynomials: $L_1, L_2, \ldots, L_n$

Newton basis polynomials: $N_0, N_1, \ldots, N_{n-1}$, where

$N_0(x) = 1$ (constant function),

$N_1(x) = x - x_1$,

$N_2(x) = (x-x_1)(x-x_2)$,

$\ldots$

$N_{n-1}(x) = (x-x_1)\ldots(x-x_n)$.

$$P(x) = y_1 L_1(x) + \cdots + y_n L_n(x),$$

$$P(x) = c_0 N_1(x) + c_1 N_2(x) + \cdots + c_{n-1} N_{n-1}(x).$$

In Lagrange's formula, one has to recompute all the basis polynomials when another data point is added to the list. In Newton's formula, we only add one new basis polynomial and compute the new coefficient c_n.

Newton's formula is helpful when we derive an error estimate for the polynomial interpolation. Lagrange formula provides a somewhat more explicit formula for the polynomial $P(x)$. The derivation of Lagrange formula is also simpler to understand.