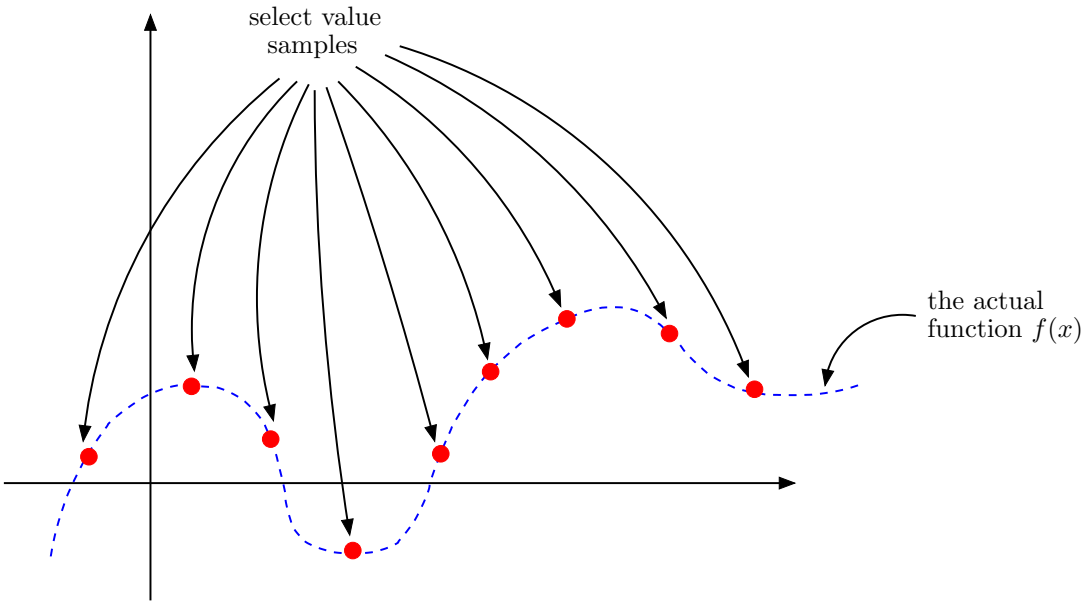
Polynomial, Lagrange, and Newton Interpolation

Mridul Aanjaneya

November 14, 2017

Interpolation

We are often interested in a certain function $f(x)$, but despite the fact that f may be defined over an entire interval of values $[a, b]$ (which may be the entire real line) we only know its precise value at select point $x_1, x_2, \ldots, x_N$.



There may be several good reasons why we could only have a limited number of values for $f(x)$, instead of its entire graph:

- Perhaps we do not have an analytic formula for $f(x)$ because it is the result of a complex process that is only observed experimentally. For example, $f(x)$ could correspond to a physical quantity (temperature, density, concentration, velocity, etc.) which varies over time in a laboratory experiment. Instead of an explicit formula, we use a measurement device to capture sample values of f at predetermined points in time.

- Or, perhaps we do have a formula for $f(x)$, but this formula is not trivially easy to evaluate. Consider for example:

$$f(x) = \sin(x) \quad \text{or} \quad f(x) = \ln(x) \quad \text{or} \quad f(x) = \int_0^x e^{-t^2} dt$$

  Perhaps evaluating $f(x)$ with such a formula is a very expensive operation and we want to consider a less expensive way to obtain a "crude approximation". In fact, in years when computers were not as ubiquitous as today, trigonometric tables were very popular. For example:

Angle	$\sin(\theta)$	$\cos(\theta)$
...	...	...
44°	0.695	0.719
45°	0.707	0.707
46°	0.719	0.695
47°	0.731	0.682
...	...	...

  If we were asked to approximate the value of $\sin(44.6°)$, it would be natural to consider deriving an estimate from these tabulated values, rather than attempting to write an analytic expression for this quantity.

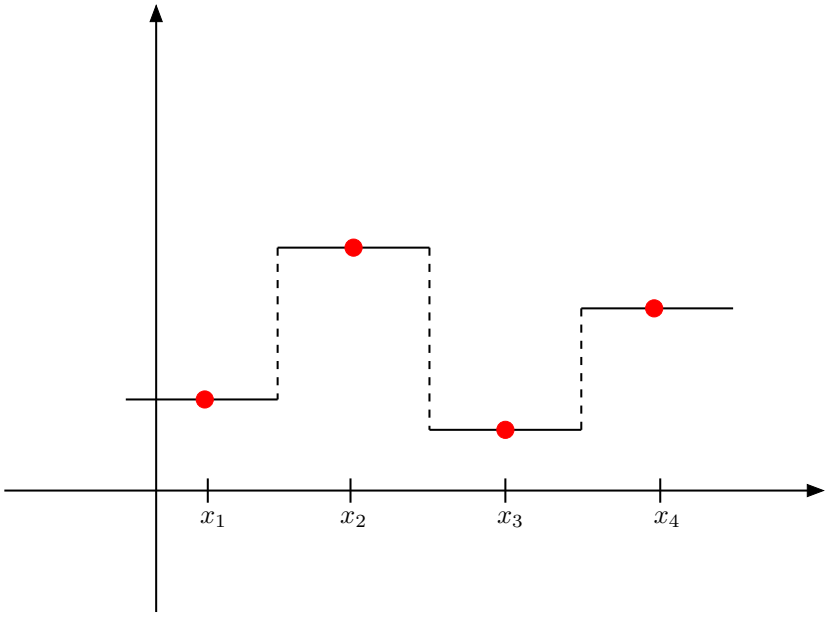
Interpolation methods attempt to answer questions about the value of $f(x)$ at points *other* than the ones it was sampled at. An obvious question would be to ask what is an estimate for $f(x^*)$ for a value x^* different than any sample we have collected; similar questions can be asked about the derivatives $f'(x^*), f''(x^*), \ldots$ at such locations.

The question of how to reconstruct a smooth function $f(x)$ that agrees with a number of collected sample values is not a straightforward one, especially since there is more than one way to accomplish this task. First, let us introduce some notation: let us write $x_1, x_2, \ldots, x_N$ for the x-locations where f is being sampled and denote the known value of $f(x)$ at $x = x_1$ as $y_1 = f(x_1)$, at $x = x_2$ as $y_2 = f(x_2)$, etc. Graphically, we seek to reconstruct a function $f(x), x \in [a, b]$ such that the plot of f passes through the following points:

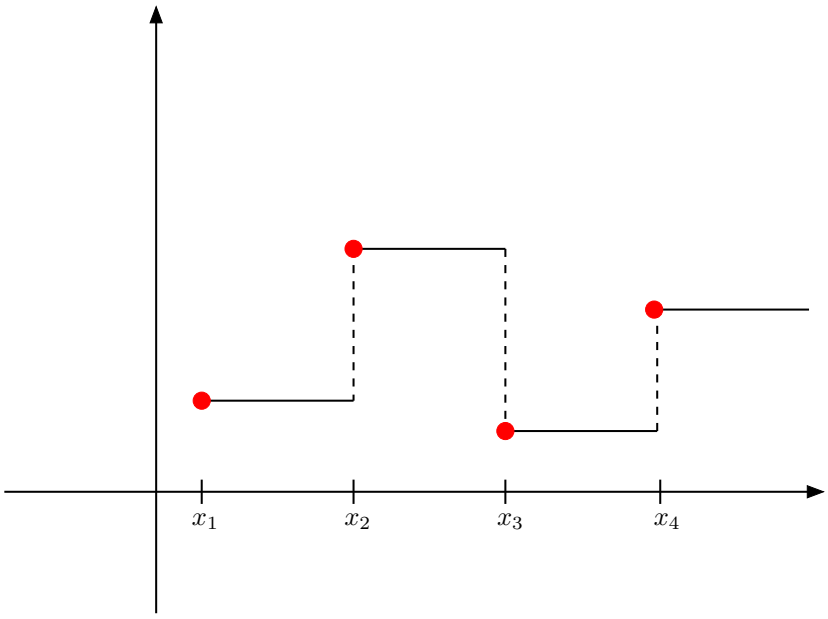
$$(x_1, y_1) \ , \ (x_2, y_2) \ , \ \ldots \ , \ (x_N, y_N)$$

Here are some possible ways to do that:

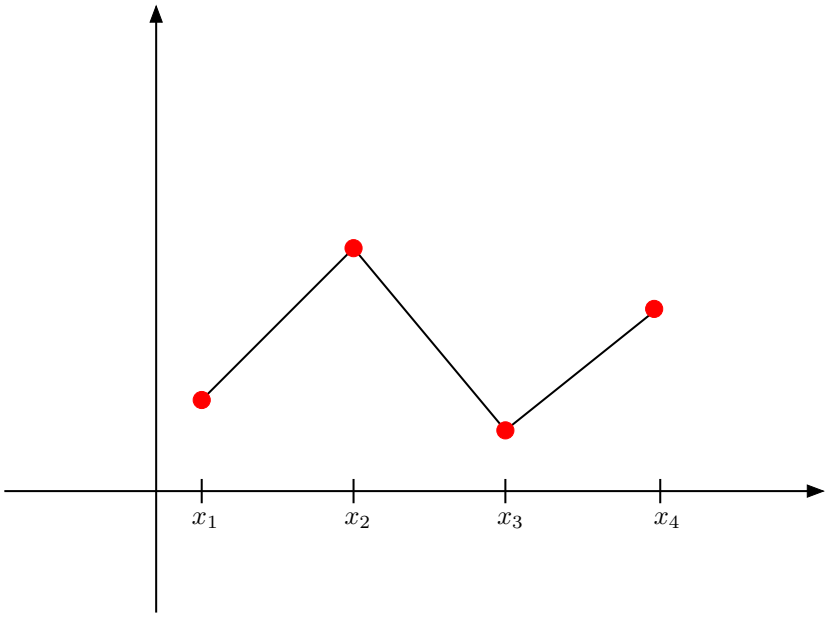
- For every x, pick the x_i closest to it, and set $f(x) = y_i$



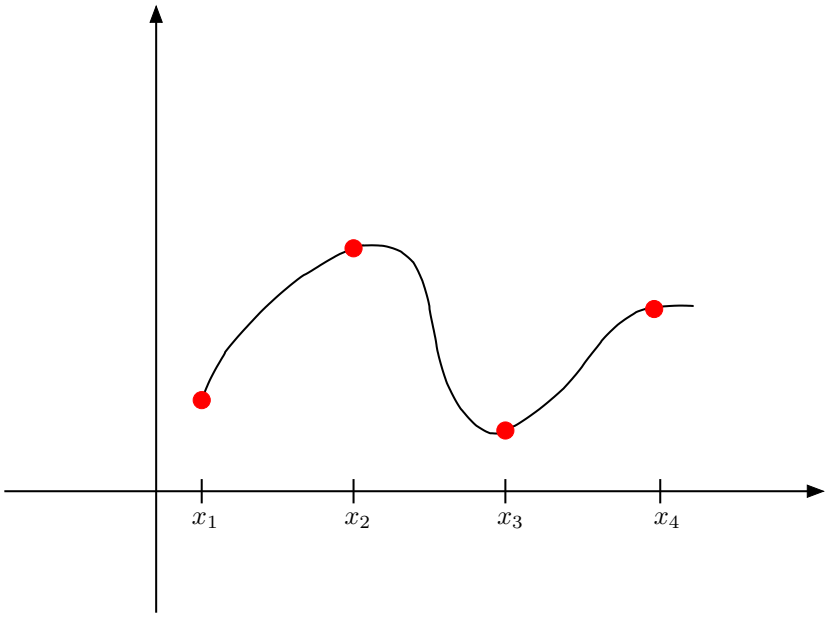
- Or, simply pick the value to the "left":



- Try connecting every two horizontally neighboring points with a straight line



- Or, try to find a smoother curve that connects them all ...



It is not trivial to argue that any particular one of these alternatives is "better", without having some knowledge of the nature of $f(x)$, or the purpose this reconstructed function will be used for. For example:

- It may appear that the discontinuous approximation generated by the "pick the closest sample" method is awkward and not as well-behaved. However, the real function $f(x)$ being sampled could have been just as discontinuous to begin with, for example, if $f(t)$ denoted the transaction amount for the customer of a bank being served at time $= t$.
- Sometimes, we may know that the real $f(x)$ is supposed to have some degree of smoothness. For example, if $f(t)$ is the position of a moving vehicle in a highway, we would expect both $f(t)$ and $f'(t)$ (velocity), possibly even $f''(t)$ (acceleration) to be continuous functions of time. In this case, if we seek to estimate $f'(t)$ at a given time, we may prefer the piecewise-linear reconstruction. If $f''(t)$ is needed, the even smoother method might be preferable.

Polynomial Interpolation

A commonly used approach is to use a properly crafted polynomial function

$$f(x) = \mathcal{P}_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

to interpolate the points $(x_0, y_0), \ldots, (x_k, y_k)$. Some benefits:

- Polynomials are relatively simple to evaluate. They can be evaluated very efficiently using *Horner's method*, also known as *nested evaluation* or *synthetic division*:

  $$\mathcal{P}_n(x) = a_0 + x(a_1 + x(a_2 + x(\ldots(a_{n-1} + x a_n)\ldots)))$$

  which requires only n additions and n multiplications. For example,

  $$1 - 4x + 5x^2 - 2x^3 + 3x^4 = 1 + x(-4 + x(5 + x(-2 + 3x)))$$

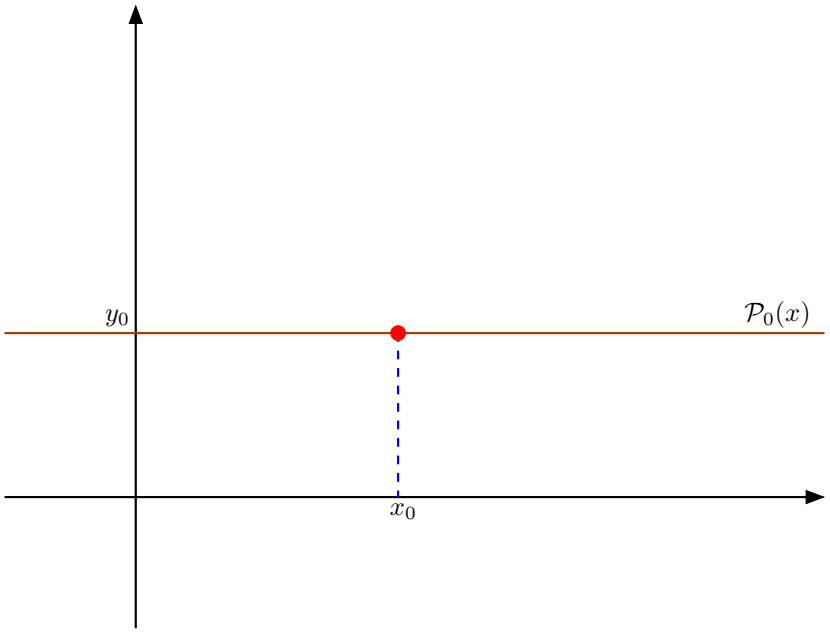
- We can easily compute derivatives $\mathcal{P}'_n, \mathcal{P}''_n$ if desired.
- Reasonably established procedure to determine the coefficients a_i.
- Polynomial approximations are *familiar* from, e.g., Taylor series.

And some disadvantages:

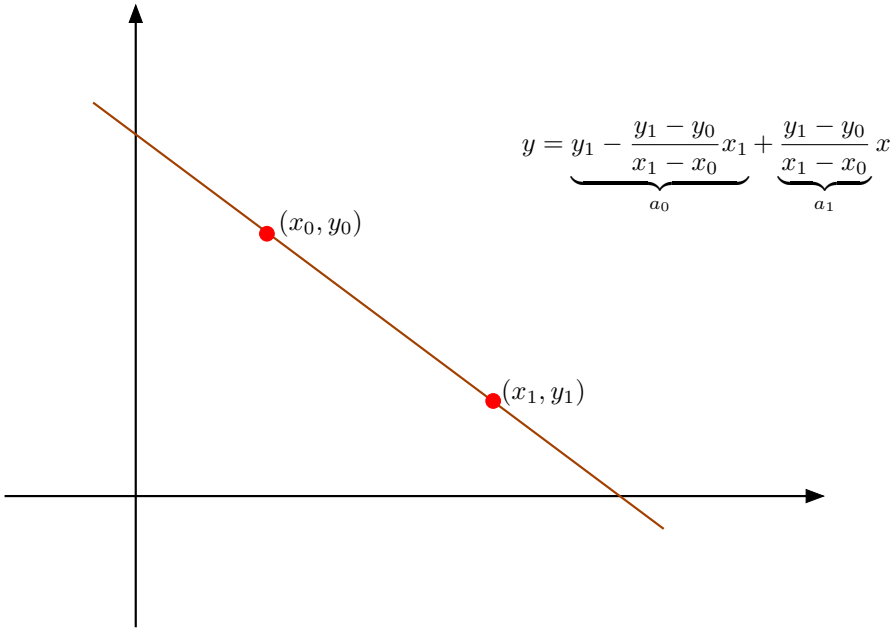
- Fitting polynomials can be problematic, when
  1. We have *many* data points (i.e., k is large), or
  2. Some of the samples are too close together (i.e., $|x_i - x_j|$ is small).

In the interest of simplicity (and for some other reasons), we try to find the most basic, yet adequate, $\mathcal{P}_n(x)$ that interpolates $(x_0, y_0), \ldots, (x_k, y_k)$. For example,

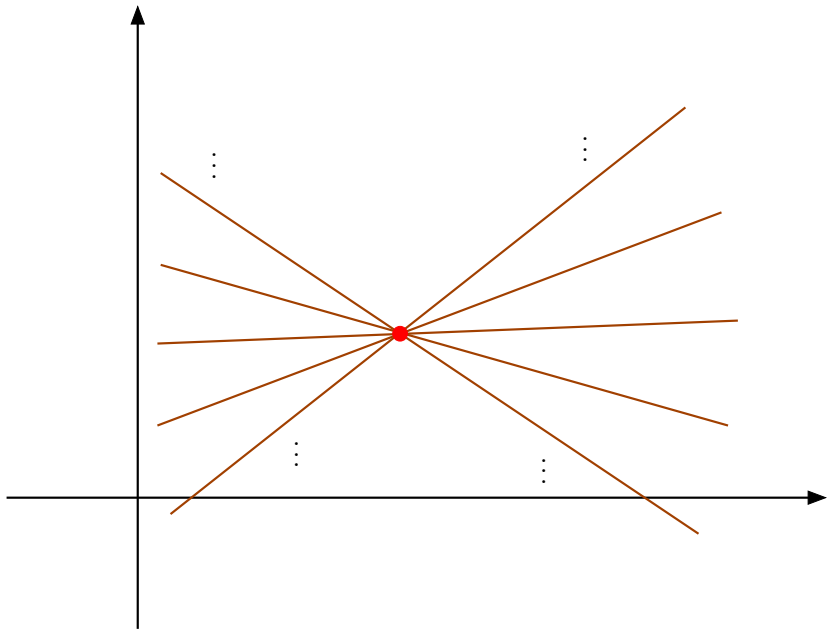
- If $k = 0$ (only one data sample), a 0-degree polynomial (i.e., a constant function) will be adequate.



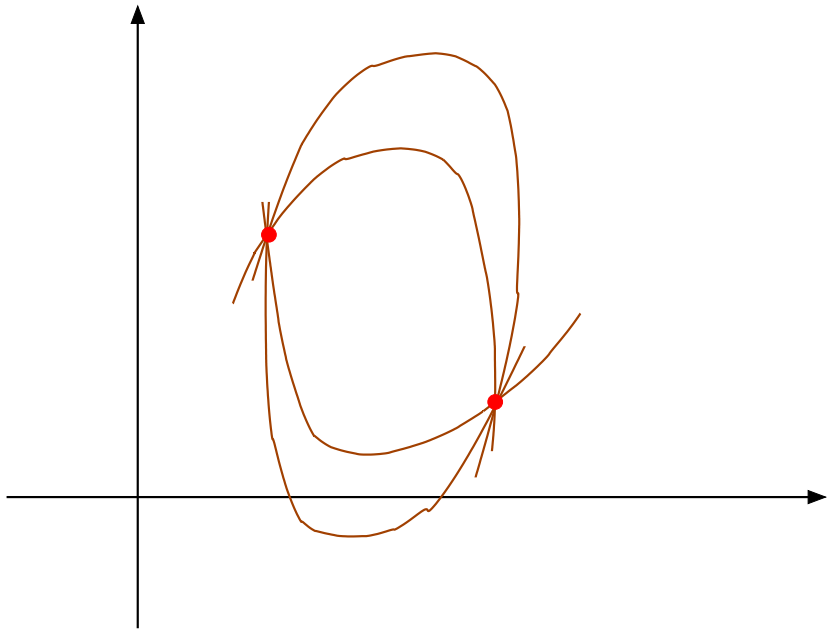
- If $k = 1$, we have two points (x_0, y_0) and (x_1, y_1). A 0-degree polynomial $\mathcal{P}_0(x) = a_0$ will not always be able to pass through both points (unless $y_0 = y_1$), but a 1-degree polynomial $\mathcal{P}_1(x) = a_0 + a_1 x$ always can.



These are not the only polynomials that accomplish the task, e.g., when $k = 0$,



or



The problem with using a degree higher than the minimum necessary is that:

- More than 1 solution becomes available, with the "right" one being unclear.
- Wildly varying curves become permissible, producing questionable approximations.

In fact, we can show that using a polynomial $\mathcal{P}_n(x)$ of degree n is the *best* choice when interpolating $n+1$ points. In this case, the following properties are assured:

- **Existence:** Such a polynomial *always* exists (assuming that all the x_i's are different! It would be impossible for a function to pass through 2 points on the same vertical line). We will show this later, by explicitly constructing such a function. For now, we can at least show that such a task would have been impossible (in general) if we were only allowed to use degree-$(n-1)$ polynomials. In fact, consider the points

  $$(x_0, y_0 = 0), (x_1, y_1 = 0), \ldots, (x_{n-1}, y_{n-1} = 0), (x_n, y_n = 1)$$

  Thus, if a degree-$(n-1)$ polynomial was able to interpolate these points, we would have:

  $$\mathcal{P}_{n-1}(x_0) = \mathcal{P}_{n-1}(x_1) = \ldots = \mathcal{P}_{n-1}(x_{n-1}) = 0$$

  $\mathcal{P}_{n-1}(x)$ can only equal zero at *exactly* $n-1$ locations *unless* it is the zero polynomial. Since $\mathcal{P}_{n-1}(x)$ is zero at n locations, we conclude that $\mathcal{P}_{n-1}(x) \equiv 0$. This is a contradiction as $\mathcal{P}_{n-1}(x_n) \neq 0$!

- **Uniqueness:** We can sketch a proof by contradiction. Assume that

  $$\mathcal{P}_n(x) = p_0 + p_1 x + \ldots + p_n x^n$$
  $$\mathcal{Q}_n(x) = q_0 + q_1 x + \ldots + q_n x^n$$

  both interpolate every (x_i, y_i), i.e., $\mathcal{P}_n(x_i) = \mathcal{Q}_n(x_i) = y_i$, for all $0 \leq i \leq n$. Define another n-degree polynomial

  $$\mathcal{R}_n(x) = \mathcal{P}_n(x) - \mathcal{Q}_n(x) = r_0 + r_1 x + \ldots + r_n x^n$$

  Apparently, $\mathcal{R}_n(x_i) = 0$ for all $0 \leq i \leq n$. From algebra, we know that *every* polynomial of degree n has at most n real roots, *unless* it is the zero polynomial, i.e., $r_0 = r_1 = \ldots = r_n = 0$. Since we have $\mathcal{R}_n(x) = 0$ for $n+1$ distinct values, we must have $\mathcal{R}_n(x) = 0 \Rightarrow \mathcal{P}_n(x) = \mathcal{Q}_n(x)$!

The most basic procedure to determine the coefficients $a_0, a_1, \ldots, a_n$ of the interpolating polynomial $\mathcal{P}_n(x)$ is to write a linear system of equations as follows:

$$\begin{aligned}
a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_{n-1} x_1^{n-1} + a_n x_1^n &= \mathcal{P}_n(x_1) = y_1 \\
a_0 + a_1 x_2 + a_2 x_2^2 + \ldots + a_{n-1} x_2^{n-1} + a_n x_2^n &= \mathcal{P}_n(x_2) = y_2 \\
&\vdots \\
a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_{n-1} x_n^{n-1} + a_n x_n^n &= \mathcal{P}_n(x_n) = y_n
\end{aligned}$$

or, in matrix form:

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} & x_n^n \end{bmatrix}}_{V_{(n+1)\times(n+1)}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}}_{a_{(n+1)}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}}_{y_{(n+1)}}$$

The matrix V is called a *Vandermonde matrix*. The set of functions $\{1, x, x^2, \dots, x^n\}$ represent the *monomial basis*. We will see that V is non-singular, thus, we can solve the system $V\tilde{a} = \tilde{y}$ to obtain the coefficients $\tilde{a} = (a_0, a_1, \dots, a_n)$. Let's evaluate the merit and drawbacks of this approach:

- Cost to determine the polynomial $\mathcal{P}_n(x)$: *very costly.*

  Since a dense $(n+1) \times (n+1)$ linear system has to be solved. This will generally require time proportional to n^3, making large interpolation problems intractable. In addition, the Vandermonde matrix is notorious for being challenging to solve (especially with Gaussian elimination) and prone to large errors in the computed coefficients $\{a_i\}$, when n is large and/or $x_i \approx x_j$.

- Cost to evaluate $f(x)$ (x=arbitrary) if coefficients are known: *very cheap.* Using Horner's method:

  $$a_0 + a_1 x + a_2 x^2 + \dots a_n x^n = a_0 + x(a_1 + x(a_2 + x(\dots(a_{n-1} + x a_n)\dots)))$$

- Availability of derivatives: *very easy.* For example,

  $$\mathcal{P}_n'(x) = a_1 + 2a_2 x + 3a_3 x^2 + \dots + (n-1)a_{n-1}x^{n-2} + n a_n x^{n-1}$$

- Allows incremental interpolation: *no!*

  This property examines if interpolating through $(x_0, y_0), \dots, (x_n, y_n)$ is *easier* if we already know a polynomial (of degree $n-1$) that interpolates through $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$. In our case, the system $V\tilde{a} = \tilde{y}$ would have to be solved from scratch for the $n+1$ data points.

To illustrate polynomial interpolation using the monomial basis, we will determine the polynomial of degree 2 interpolating the three data points $(-2, -27)$, $(0, -1)$, $(1, 0)$. In general, there is a unique polynomial

$$\mathcal{P}_2(x) = a_0 + a_1 x + a_2 x^2$$

Writing down the Vandermonde system for this data gives

$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

Solving this system by Gaussian elimination yields the solution $\tilde{a} = (-1, 5, -4)$ so that the interpolating polynomial is

$$\mathcal{P}_2(x) = -1 + 5x - 4x^2$$

Lagrange Interpolation

Lagrange interpolation is an alternative way to define $\mathcal{P}_n(x)$, without having to solve expensive systems of equations. For a given set of $n+1$ points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, define the Lagrange polynomials of degree-n $l_0(x)$, $l_1(x), \ldots, l_n(x)$ as:

$$l_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Then, the interpolating polynomial is simply

$$\mathcal{P}_n(x) = y_0 l_0(x) + y_1 l_1(x) + \ldots + y_n l_n(x) = \sum_{i=0}^{n} y_i l_i(x)$$

Note that no solution of a linear system is required here. We just have to explain what every $l_i(x)$ looks like. Since $l_i(x)$ is a degree-n polynomial, with the n-roots $x_0, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, x_{i+2}, \ldots, x_n$, it must have the form

$$\begin{aligned} l_i(x) &= C_i(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n) \\ &= C_i \prod_{j \neq i}(x - x_j) \end{aligned}$$

Now, we require that $l_i(x_i) = 1$, thus

$$1 = C_i \cdot \prod_{j \neq i}(x_i - x_j) \Rightarrow C_i = \frac{1}{\prod_{j \neq i}(x_i - x_j)}$$

Thus, for every i, we have

$$\begin{aligned} l_i(x) &= \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} \\ &= \prod_{j \neq i} \frac{(x - x_j)}{(x_i - x_j)} \end{aligned}$$

Note: This result essentially proves *existence* of a polynomial interpolant of degree n that passes through $n+1$ data points. We can also use it to prove that the Vandermonde matrix V is non-singular. If it *were* singular, a right hand side $\tilde{y} = (y_0, \ldots, y_n)$ would have existed such that $V\tilde{a} = \tilde{y}$ would have no solution, which is a contradiction!

Let's use Lagrange interpolation to compute an interpolating polynomial to the three data points $(-2, -27), (0, -1), (1, 0)$.

$$\begin{aligned} \mathcal{P}_2(x) &= -27\frac{(x-0)(x-1)}{(-2-0)(-2-1)} + (-1)\frac{(x-(-2))(x-1)}{(0-(-2))(0-1)} + 0\frac{(x-(-2))(x-0)}{(1-(-2))(1-0)} \\ &= -27\frac{x(x-1)}{6} + \frac{(x+2)(x-1)}{2} = -1 + 5x - 4x^2 \end{aligned}$$

Recall form Lecture 8 that this is the same polynomial we computed using the monomial basis!

Let us evaluate the same four quality metrics we saw before for the Vandermonde matrix approach.

- Cost of determining $\mathcal{P}_n(x)$: *very easy*.

  We are essentially able to write a formula for $\mathcal{P}_n(x)$ without solving any systems. However, if we want to write $\mathcal{P}_n(x) = a_0 + a_1 x + \ldots + a_n x^n$, the cost of evaluating the a_i's would be very high! Each $l_i(x)$ would need to be expanded, leading to $O(n^2)$ operations for each $l_i(x)$ implying $O(n^3)$ operations for $\mathcal{P}_n(x)$.

- Cost of evaluating $\mathcal{P}_n(x)$ for an arbitrary x: *significant*.

  We do not really need to compute the a_i's beforehand, if we only need to evaluate $\mathcal{P}_n(x)$ at a select few locations. For each $l_i(x)$ the evaluation requires n subtractions and n multiplications, implying a total of $O(n^2)$ operations (better than $O(n^3)$ for computing the a_i's).

- Availability of derivatives: *not readily available*.

  Differentiating each $l_i(x)$ (since $\mathcal{P}_n'(x) = \sum y_i l_i'(x)$) is not trivial; the above expression has n terms each with $n-1$ products per term.

- Incremental interpolation: *not accomodated*.

Still, Lagrange interpolation is a good quality method if we can accept its limitations.

Newton Interpolation

Newton interpolation is yet another alternative, which enables *both* efficient evaluation *and* allows for incremental construction. Additionally, it allows both the coefficients $\{a_i\}$ as well as the derivative $\mathcal{P}_n'(x)$ to be evaluated efficiently. For a given set of data points $(x_0, y_0), \ldots, (x_n, y_n)$, the Newton basis functions are given by

$$\pi_j(x) = (x - x_0)(x - x_1)\ldots(x - x_{j-1}) = \prod_{k=1}^{j-1}(x - x_k), \quad j = 0, \ldots, n$$

where we take the value of the product to be 1 when the limits make it vacuous. In the Newton basis, a given polynomial has the form

$$\mathcal{P}_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_{n-1}(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

From the definition, we see that $\pi_j(x_i) = 0$ for $i < j$, so that the basis matrix A with $a_{ij} = \pi_j(x_i)$ is *lower triangular*. To illustrate Newton interpolation, we use it to determine the interpolating polynomial for the three data points

$(-2,-27),(0,-1),(1,0)$. With the Newton basis, we have the lower triangular linear system

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & x_1 - x_0 & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

For the given data, this system becomes

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

whose solution is $\tilde{a} = (-27, 13, -4)$. Thus, the interpolating polynomial is

$$\mathcal{P}_2(x) = -27 + 13(x+2) - 4(x+2)x = -1 + 5x - 4x^2$$

which is the same polynomial we computed earlier!

Incremental Construction

The Newton basis functions can be derived by considering the problem of building a polynomial interpolant *incrementally* as successive new data points are added. Here is the basic idea:

- **Step 0:** Define a degree-0 polynomial $\mathcal{P}_0(x)$ that just interpolates (x_0, y_0). Obviously, we can achieve that by simply selecting

  $$\mathcal{P}_0(x) = y_0$$

- **Step 1:** Define a degree-1 polynomial $\mathcal{P}_1(x)$ that now interpolates both (x_0, y_0) and (x_1, y_1). We also want to take advantage of the previously defined $\mathcal{P}_0(x)$ by constructing $\mathcal{P}_1$ as

  $$\mathcal{P}_1(x) = \mathcal{P}_0(x) + \mathcal{M}_1(x)$$

  where $\mathcal{M}_1(x)$ is a degree-1 polynomial and it needs to satisfy

  $$\underbrace{\mathcal{P}_1(x_0)}_{=y_0} = \underbrace{\mathcal{P}_0(x_0)}_{=y_0} + \mathcal{M}_1(x_0) \Rightarrow \mathcal{M}_1(x_0) = 0$$

  Thus, $\mathcal{M}_1(x) = c_1(x - x_0)$. We can determine c_1 using:

  $$\mathcal{P}_1(x_1) = \mathcal{P}_0(x_1) + c_1(x_1 - x_0) \Rightarrow c_1 = \frac{\mathcal{P}_1(x_1) - \mathcal{P}_0(x_1)}{x_1 - x_0} = \frac{y_1 - \mathcal{P}_0(x_1)}{x_1 - x_0}$$

- **Step 2:** Now construct $\mathcal{P}_2(x)$ which interpolates the three points (x_0, y_0), $(x_1, y_1), (x_2, y_2)$. Define it as:

  $$\mathcal{P}_2(x) = \mathcal{P}_1(x) + \mathcal{M}_2(x)$$

where $\mathcal{M}_2(x)$ is a degree-2 polynomial. Once again we observe that

$$\left.\begin{array}{rcl} \underbrace{\mathcal{P}_2(x_0)}_{=y_0} & = & \underbrace{\mathcal{P}_1(x_0)}_{=y_0} + \mathcal{M}_2(x_0) \\ \underbrace{\mathcal{P}_2(x_1)}_{=y_1} & = & \underbrace{\mathcal{P}_1(x_1)}_{=y_1} + \mathcal{M}_2(x_1) \end{array}\right\} \Rightarrow \mathcal{M}_2(x_0) = \mathcal{M}_2(x_1) = 0$$

Thus, $\mathcal{M}_2(x)$ must have the form:

$$\mathcal{M}_2(x) = c_2(x - x_0)(x - x_1)$$

Substituting $x \leftarrow x_2$, we get an expression for c_2

$$\begin{array}{rcl} y_2 = \mathcal{P}_2(x_2) & = & \mathcal{P}_1(x_2) + c_2(x_2 - x_0)(x_2 - x_1) \\ & \Rightarrow & c_2 = \dfrac{y_2 - \mathcal{P}_1(x_2)}{(x_2 - x_0)(x_2 - x_1)} \end{array}$$

- **Step k:** In the previous step, we constructed a degree-$(k-1)$ polynomial that interpolates $(x_0, y_0), \ldots, (x_{k-1}, y_{k-1})$. We will use this $\mathcal{P}_{k-1}(x)$ and now define a degree-k polynomial $\mathcal{P}_k(x)$ such that all of $(x_0, y_0), \ldots, (x_k, y_k)$ are interpolated. Again,

  $$\mathcal{P}_k(x) = \mathcal{P}_{k-1}(x) + \mathcal{M}_k(x)$$

  where $\mathcal{M}_k(x)$ is a degree-k polynomial.

  Now we have for any $i \in \{0, 1, \ldots, k-1\}$

  $$\underbrace{\mathcal{P}_k(x_i)}_{=y_i} = \underbrace{\mathcal{P}_{k-1}(x_i)}_{=y_i} + \mathcal{M}_k(x_i) \Rightarrow \mathcal{M}_k(x_i) = 0$$

  Thus, the degree-k polynomial $\mathcal{M}_k$ must have the form

  $$\mathcal{M}_k(x) = c_k(x - x_0) \ldots (x - x_{k-1})$$

  Substituting $x \leftarrow x_k$ gives

  $$\begin{array}{rcl} y_k & = & \mathcal{P}_k(x_k) = \mathcal{P}_{k-1}(x_k) + c_k(x_k - x_0) \ldots (x_k - x_{k-1}) \\ \Rightarrow c_k & = & \dfrac{y_k - \mathcal{P}_{k-1}(x_k)}{\prod_{j=0}^{k-1}(x_k - x_j)} \end{array}$$

Every polynomial $\mathcal{M}_i(x)$ in this process is written as

$$\mathcal{M}_i(x) = c_i \mathcal{N}_i(x) \quad \text{where} \quad \mathcal{N}_i(x) = \prod_{j=0}^{i-1}(x - x_j)$$

After n steps, the interpolating polynomial $\mathcal{P}_n(x)$ is then written as:

$$\mathcal{P}_n(x) = c_0\mathcal{N}_0(x) + c_1\mathcal{N}_1(x) + \ldots + c_n\mathcal{N}_n(x)$$

where

$$\begin{aligned}
\mathcal{N}_0(x) &= 1 \\
\mathcal{N}_1(x) &= x - x_0 \\
\mathcal{N}_2(x) &= (x - x_0)(x - x_1) \\
&\vdots \\
\mathcal{N}_k(x) &= (x - x_0)(x - x_1)\dots(x - x_{k-1})
\end{aligned}$$

These are the *Newton polynomials* (compare with the Lagrange polynomials $l_i(x)$). Note that the x_i's are called the *centers*.

We illustrate the incremental Newton interpolation by building the Newton interpolant incrementally as the new data points are added. We begin with the first data point $(x_0, y_0) = (-2, -27)$, which is interpolated by the constant polynomial

$$\mathcal{P}_0(x) = y_0 = -27$$

Incorporating the second data point $(x_1, y_1) = (0, -1)$, we modify the previous polynomial so that it interpolates the new data point as well:

$$\begin{aligned}
\mathcal{P}_1(x) &= \mathcal{P}_0(x) + \mathcal{M}_1(x) = \mathcal{P}_0(x) + c_1(x - x_0) \\
&= \mathcal{P}_0(x) + \frac{y_1 - \mathcal{P}_0(x)}{x_1 - x_0}(x - x_0) \\
&= -27 + \frac{-1 - (-27)}{0 - (-2)}(x - (-2)) \\
&= -27 + 13(x + 2)
\end{aligned}$$

Finally, we incorporate the third data point $(x_2, y_2) = (1, 0)$, modifying the previous polynomial so that it interpolates the new data point as well:

$$\begin{aligned}
\mathcal{P}_2(x) &= \mathcal{P}_1(x) + \mathcal{M}_2(x) = \mathcal{P}_1(x) + c_2(x - x_0)(x - x_1) \\
&= \mathcal{P}_1(x) + \frac{y_2 - \mathcal{P}_1(x_2)}{(x_2 - x_0)(x_2 - x_1)}(x - x_0)(x - x_1) \\
&= -27 + 13(x + 2) + \frac{0 - 12}{(1 - (-2))(1 - 0)}(x - (-2))(x - 0) \\
&= -27 + 13(x + 2) - 4(x + 2)x
\end{aligned}$$

Divided Differences

So far, we saw two ways of computing the Newton interpolant, triangular matrix and incremental interpolation. There is, however, another efficient and systematic way to compute them, called *divided differences*. A divided difference is a function defined over a set of sequentially indexed centers, e.g.,

$$x_i, x_{i+1}, \dots, x_{i+j-1}, x_{i+j}$$

The divided difference of these values is denoted by:

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

The value of this symbol is defined recursively as follows. For divided differences with one argument,

$$f[x_i] \equiv f(x_i) = y_i$$

With two arguments:

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

With three:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

With $j + 1$ arguments:

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

The fact that makes divided differences so useful is that $f[x_i, \ldots, x_{i+j}]$ can be shown to be the coefficient of the *highest power of* x in a polynomial that interpolates through

$$(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_{i+j-1}, y_{i+j-1}), (x_{i+j}, y_{i+j})$$

Why is this so useful?

Remember, the polynomial that interpolates

$$(x_0, y_0), \ldots, (x_k, y_k)$$

is

$$\mathcal{P}_k(x) = \underbrace{\mathcal{P}_{k-1}(x)}_{\text{highest power}=x^{k-1}} + \underbrace{c_k(x - x_0)\ldots(x - x_{k-1})}_{=c_k x^k + \text{lower powers}}$$

Thus, $c_k = f[x_0, x_1, x_2, \ldots, x_k]$! Or, in other words,

$$\begin{aligned} \mathcal{P}_n(x) &= f[x_0] \\ &+ f[x_0, x_1](x - x_0) \\ &+ f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\vdots \\ &+ f[x_0, x_1, \ldots, x_n](x - x_0)\ldots(x - x_{n-1}) \end{aligned}$$

So, if we can quickly evaluate the divided differences, we have determined $\mathcal{P}_n(x)$! Let us see a specific example:

$$\begin{aligned}
(x_0, y_0) &= (-2, -27) \\
(x_1, y_1) &= (0, -1) \\
(x_2, y_2) &= (1, 0) \\
f[x_0] &= y_0 = -27 \\
f[x_1] &= y_1 = -1 \\
f[x_2] &= y_2 = -0 \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{-1 - (-27)}{0 - (-2)} = 13 \\
f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{0 - (-1)}{1 - 0} = 1 \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{1 - 13}{1 - (-2)} = -4
\end{aligned}$$

Thus,

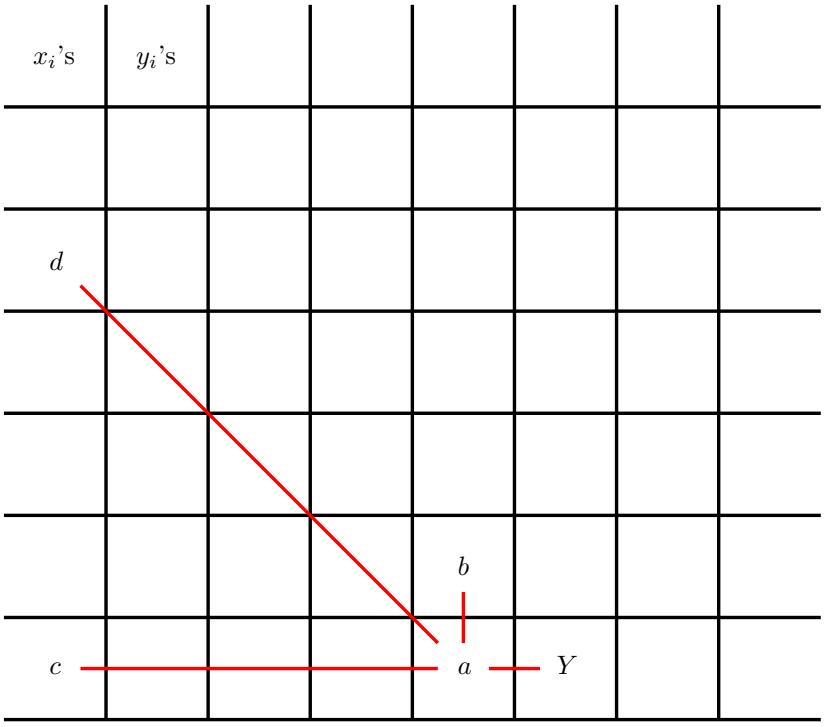
$$\begin{aligned}
\mathcal{P}_2(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&= -27 + 13(x + 2) - 4(x + 2)x
\end{aligned}$$

Divided differences are usually tabulated as follows:

	$f[\cdot]$	$f[\cdot, \cdot]$	$f[\cdot, \cdot, \cdot]$	$\ldots$
x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0, x_1]$		
x_2	$f[x_2]$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$	
x_3	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$\ldots$
x_4	$f[x_4]$	$f[x_3, x_4]$	$f[x_2, x_3, x_4]$	$\ldots$

The recursive definition can be implemented directly on the table as follows:

$$Y = \frac{a - b}{c - d}$$



For example, for the sample set $(x_0, y_0) = (-2, -27)$, $(x_1, y_1) = (0, -1)$, $(x_2, y_2) = (1, 0)$,

x_i's	y_i's			
-2	-27			
0	-1	13		
1	0	1	-4	

Easy evaluation

$$\begin{aligned}
\mathcal{P}_4(x) &= c_0 \\
&+ c_1(x - x_0) \\
&+ c_2(x - x_0)(x - x_1) \\
&+ c_3(x - x_0)(x - x_1)(x - x_2) \\
&+ c_4(x - x_0)(x - x_1)(x - x_2)(x - x_3)
\end{aligned}$$

$$= c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + (x - x_2)[c_3 + (x - x_3)\underbrace{c_4}_{\mathcal{Q}_4(x)}]]]$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathcal{Q}_0(x)}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathcal{Q}_1(x)}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathcal{Q}_2(x)}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathcal{Q}_3(x)}$$

$$\mathcal{P}_4(x) = \mathcal{Q}_0(x)$$

Recursively: Define $\mathcal{Q}_n(x) = c_n$. Then

$$\mathcal{Q}_{n-1}(x) = c_{n-1} + (x - x_{n-1})\mathcal{Q}_n(x)$$

The value of $\mathcal{P}_n(x) = \mathcal{Q}_0(x)$ can be evaluated (in linear time) by iterating this recurrence n times. We also have

$$\begin{aligned} \mathcal{Q}_{n-1}(x) &= c_{n-1} + (x - x_{n-1})\mathcal{Q}_n(x) \\ \Rightarrow \mathcal{Q}'_{n-1}(x) &= \mathcal{Q}_n(x) + (x - x_{n-1})\mathcal{Q}'_n(x) \end{aligned}$$

Thus, once we have computed all the $\mathcal{Q}'_k$s, we can also compute all the derivatives too! Ultimately, $\mathcal{P}'_n(x) = \mathcal{Q}'_0(x)$.

Let us evaluate Newton's method, as we did with other methods:

- Cost of computing $\mathcal{P}_n(x)$: $O(n^2)$.
- Cost of evaluating $\mathcal{P}_n(x)$ for an arbitrary x: $O(n)$.

  This can be accelerated (similar to Horner's method) using the recursive scheme defined above.
- Availability of derivatives: *yes*, as discussed above.
- Allows for incremental interpolation: *yes*!