Lagrange Polynomials

Minh Van Nguyen
nguyenminh2@gmail.com

13 January 2010

1 Linear approximation

Given two points (x_0, y_0) and (x_1, y_1), we can find a polynomial of degree one (i.e. a linear polynomial) that passes through these points. If P is the polynomial that we seek, we want it to have the property that $P(x_0) = y_0$ and $P(x_1) = y_1$. To construct P, first define the functions

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \qquad \text{and} \qquad L_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

Then the linear polynomial in question is

$$\begin{aligned} P(x) &= L_0(x)y_0 + L_1(x)y_1 \\ &= \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1. \end{aligned} \tag{1}$$

To see that $P(x)$ passes through the points (x_0, y_0) and (x_1, y_1), substitute x_0 and x_1 into (1) to obtain

$$\begin{aligned} P(x_0) &= \frac{x_0 - x_1}{x_0 - x_1}y_0 + \frac{x_0 - x_0}{x_1 - x_0}y_1 \\ &= y_0 \\ P(x_1) &= \frac{x_1 - x_1}{x_0 - x_1}y_0 + \frac{x_1 - x_0}{x_1 - x_0}y_1 \\ &= y_1. \end{aligned}$$

Furthermore, we have

$$L_0(x_0) = \frac{x_0 - x_1}{x_0 - x_1} = 1 \qquad \text{and} \qquad L_0(x_1) = \frac{x_1 - x_1}{x_0 - x_1} = 0$$

and similarly

$$L_1(x_0) = \frac{x_0 - x_0}{x_1 - x_0} = 0 \qquad \text{and} \qquad L_1(x_1) = \frac{x_1 - x_0}{x_1 - x_0} = 1.$$

Copyright © 2010 Minh Van Nguyen. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation. See http://www.fsf.org/licensing/licenses/fdl.html for the full text of the license.

Example 1.1. *Consider the function* $f(x) = x^3$. *Find a linear polynomial through the points*

$$\left(-\frac{3}{2}, -\frac{27}{8}\right) \qquad \textit{and} \qquad \left(\frac{3}{2}, \frac{27}{8}\right)$$

that approximates $f(x)$.

Solution. We let

$$(x_0, y_0) = \left(-\frac{3}{2}, -\frac{27}{8}\right) \qquad \text{and} \qquad (x_1, y_1) = \left(\frac{3}{2}, \frac{27}{8}\right)$$

and substitute into (1) to get

$$\begin{aligned} P(x) &= \frac{x - 3/2}{-3/2 - 3/2}\left(-\frac{27}{8}\right) + \frac{x + 3/2}{3/2 + 3/2}\left(\frac{27}{8}\right) \\ &= \frac{9}{4}x. \end{aligned}$$

To see how good $P(x)$ is at approximating $f(x)$, consider plotting $P(x)$ and $f(x)$ on the same set of axes:

```
sage: f = x^3
sage: x0 = -1.5; y0 = f(x=x0)
sage: x1 = 1.5; y1 = f(x=x1)
sage: P1 = plot(f, (x,-2,2))
sage: P = ((x - x1) / (x0 - x1))*y0 + ((x - x0) / (x1 - x0))*y1
sage: P2 = plot(P, (x,-2,2), color="red")
sage: show(P1 + P2)
```

Figure 1 shows the plots of $f(x)$ (coloured blue) and $P(x)$ (coloured red). □

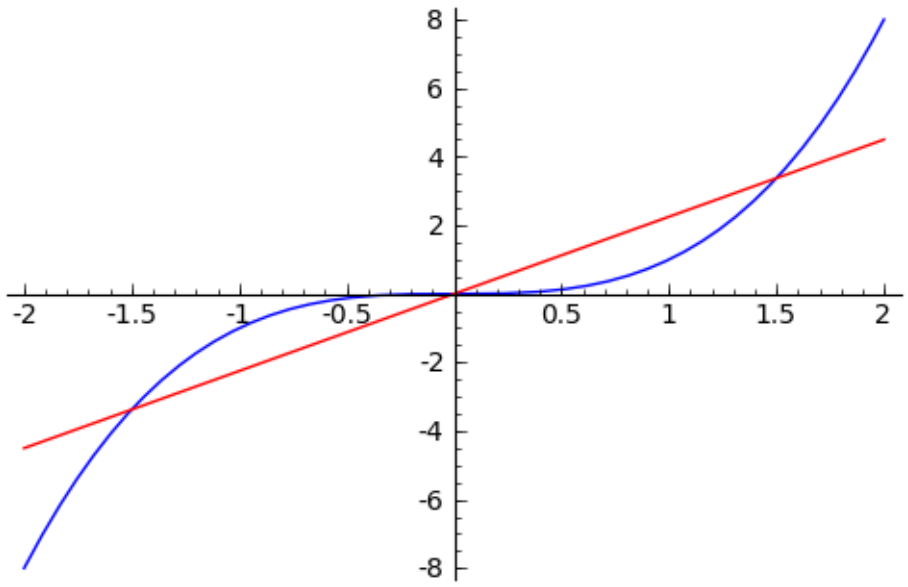

Figure 1: Approximating $f(x) = x^3$ by a linear polynomial through two points on $f(x)$.

2 Lagrange interpolating polynomial

If we are now given $n+1$ points

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n) \tag{2}$$

a natural question to ask is, "How can we adapt the above method for two points to the case of $n+1$ points?" For each $k = 0, 1, 2, \ldots, n$ we require a rational function $L_{n,k}(x)$ such that $L_{n,k}(x_i) = 0$ whenever $i \neq k$ and $L_{n,k}(x_k) = 1$. The condition $L_{n,k}(x_i) = 0$ for $i \neq k$ can be met if the numerator of $L_{n,k}(x)$ is

$$\prod_{i \neq k}^{n} (x - x_i) = (x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n).$$

To satisfy the condition $L_{n,k}(x_k) = 1$, the numerator and denominator of $L_{n,k}(x)$ must be equal to each other when evaluated at $x = x_k$. So the function $L_{n,k}(x)$ that we seek is

$$\begin{aligned} L_{n,k}(x) &= \frac{\prod_{i \neq k}^{n} (x - x_i)}{\prod_{i \neq k}^{n} (x_k - x_i)} \\ &= \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}. \end{aligned}$$

Then the general polynomial $P(x)$ that we require is defined in the same manner as equation (1). For the $n+1$ points in (2), the polynomial $P(x)$ that passes through each of those points is stated in Theorem 2.1 and is called the *n-th Lagrange interpolating polynomial*.

Theorem 2.1. ***Lagrange interpolating polynomial.*** *Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct numbers in the domain of a function f and suppose that f is defined at those $n+1$ values. Then there is a unique polynomial $P(x)$ of degree at most n such that $f(x_k) = P(x_k)$ for $k = 0, 1, 2, \ldots, n$. Furthermore, $P(x)$ is given by*

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} f(x_k) L_{n,k}(x) \\ &= f(x_0) L_{n,0}(x) + f(x_1) L_{n,1}(x) + \cdots + f(x_n) L_{n,n}(x) \end{aligned}$$

where

$$\begin{aligned} L_{n,k}(x) &= \frac{\prod_{i \neq k}^{n} (x - x_i)}{\prod_{i \neq k}^{n} (x_k - x_i)} \\ &= \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)} \end{aligned}$$

for $k = 0, 1, 2, \ldots, n$.

The next theorem gives a theoretical error bound when using the Lagrange interpolating polynomial to approximate a function.

Theorem 2.2. ***Error bound for Lagrange interpolating polynomial.*** *Suppose $x_0, x_1, \ldots, x_n$ are $n+1$ distinct numbers in the closed interval $[a, b]$ and let $f \in C^{n+1}[a, b]$. Then for each $x \in [a, b]$ there is a number $\xi(x) \in (a, b)$ such that*

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{k=0}^{n} (x - x_k)$$

where $P(x)$ is the Lagrange interpolating polynomial from Theorem 2.1.

Example 2.3. *Let $f(x) = \sin(x)$ and consider the following four points on the graph of $f(x)$:*

$$(0, 0), \qquad (2\pi/3,\ \sqrt{3}/2), \qquad (4\pi/3,\ -\sqrt{3}/2), \qquad (2\pi, 0).$$

Use Theorem 2.1 to find a polynomial that interpolates $f(x)$.

Solution. The particular $L_{3,k}(x)$ that we seek are

$$\begin{aligned}
L_{3,0} &= \frac{1}{16} \frac{(\pi - 3x)(2\pi - x)(4\pi - 3x)}{\pi^3} \\
L_{3,1} &= \frac{9}{16} \frac{(2\pi - x)(4\pi - 3x)x}{\pi^3} \\
L_{3,2} &= -\frac{9}{16} \frac{(2\pi - 3x)(2\pi - x)x}{\pi^3} \\
L_{3,3} &= \frac{1}{16} \frac{(2\pi - 3x)(4\pi - 3x)x}{\pi^3}.
\end{aligned}$$

The required interpolation polynomial is

$$P(x) = \frac{3^{\frac{7}{2}} x^3 - 3^{\frac{9}{2}} \pi x^2 + 2 \cdot 3^{\frac{7}{2}} \pi^2 x}{16\pi^3}.$$

To visualize how well $P(x)$ interpolates $f(x)$, use the following code to plot these two functions on the same set of axes:

```
sage: f = sin(x)
sage: a = 0; b = 2*pi; n = 3; d = (b - a) / n
sage: x0 = a; y0 = f(x=x0)
sage: x1 = x0 + d; y1 = f(x=x1)
sage: x2 = x1 + d; y2 = f(x=x2)
sage: x3 = x2 + d; y3 = f(x=x3)
sage: L0 = ((x-x1) * (x-x2) * (x-x3)) / ((x0-x1) * (x0-x2) * (x0-x3))
sage: L1 = ((x-x0) * (x-x2) * (x-x3)) / ((x1-x0) * (x1-x2) * (x1-x3))
sage: L2 = ((x-x0) * (x-x1) * (x-x3)) / ((x2-x0) * (x2-x1) * (x2-x3))
sage: L3 = ((x-x0) * (x-x1) * (x-x2)) / ((x3-x0) * (x3-x1) * (x3-x2))
sage: P = f(x=x0)*L0 + f(x=x1)*L1 + f(x=x2)*L2 + f(x=x3)*L3
sage: P1 = plot(f, (x,-1,7))
sage: P2 = plot(P, (x,-1,7), color="red")
sage: show(P1 + P2)
```

The resulting plot is shown in Figure 2. The graph of $f(x)$ is coloured blue, while that of $P(x)$ is coloured red. □

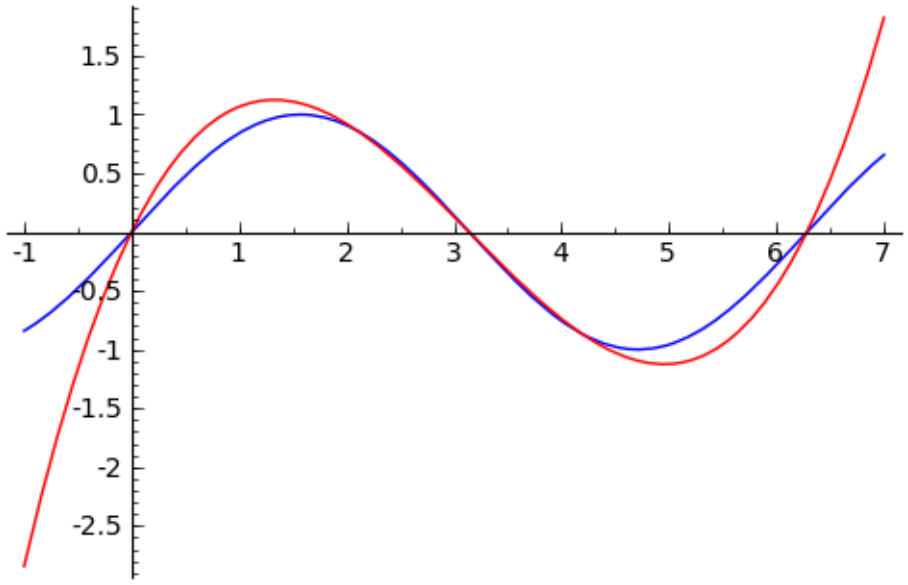

Figure 2: Interpolating $f(x) = \sin(x)$ using a cubic polynomial.

3 Lagrange polynomials via Neville's method

This section considers a method for recursively generating Lagrange interpolation polynomials. The technique is due to E.H. Neville who first published it in 1934.

Definition 3.1. *Let f be a function defined at the $n+1$ points $x_0, x_1, \ldots, x_n$ and suppose that $m_1, m_2, \ldots, m_k$ are k distinct integers such that $0 \leq m_i \leq n$ for each i. Then denote by $P_{m_1,m_2,\ldots,m_k}(x)$ the Lagrange interpolating polynomial that agrees with $f(x)$ at the k points $x_{m_1}, x_{m_2}, \ldots, x_{m_k}$.*

Theorem 3.2. ***Neville's method [1].*** *Let f be a function defined at all points in $X = \{x_0, x_1, \ldots, x_n\}$ and suppose x_i and x_j are distinct numbers in X. Then the k-th Lagrange interpolating polynomial $P(x)$ given by*

$$P(x) = \frac{(x - x_j)P_{0,1,\ldots,j-1,j+1,\ldots,k}(x) - (x - x_i)P_{0,1,\ldots,i-1,i+1,\ldots,k}(x)}{x_i - x_j}$$

interpolates the function f at the $k+1$ distinct numbers $x_0, x_1, \ldots, x_k$.

As a convenience, for $0 \leq j \leq i$ let $Q_{i,j}(x)$ denote the j-th Lagrange interpolating polynomial on the $j+1$ numbers $x_{i-j}, x_{i-j+1}, \ldots, x_{i-1}, x_i$ where

$$Q_{i,j} = P_{i-j,i-j+1,\ldots,i-1,i}.$$

Then Neville's method can be described in terms of $Q_{i,j}$ as shown in Algorithm 3.1. This algorithm generates the n-th Lagrange interpolating polynomial without using the rational function $L_{n,k}$ from Theorem 2.1. Upon termination, the algorithm returns a table Q, called the Neville table. Each entry $Q_{i,j}$ of the table is a polynomial of degree at most n and the entry $Q_{n,n}$ is the n-th Lagrange interpolating polynomial.

Example 3.3. *Suppose we are given the values in Table 1. Use Neville's method as stated in Algorithm 3.1 to find the 4-th Lagrange interpolating polynomial of these points.*

Input : The numbers $x_0, x_1, \ldots, x_n$. The $n+1$ values $f(x_0), f(x_1), \ldots, f(x_n)$ as the first column $Q_{0,0}, Q_{1,0}, \ldots, Q_{n,0}$ of Q.
Output: The Neville table Q where $P(x) = Q_{n,n}$.

Let Q be an $(n+1) \times (n+1)$ array
$Q_{i,0} \leftarrow f(x_i)$ for $i \leftarrow 0, 1, \ldots, n$
for $i \leftarrow 1, 2, \ldots, n$ **do**
    for $j \leftarrow 1, 2, \ldots, i$ **do**
        $$Q_{i,j} \leftarrow \frac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$$
    end
end
return Q

Algorithm 3.1: Neville's method for recursively generating the n-th Lagrange interpolating polynomial.

x	$f(x)$
0.7847	2.3610
1.0320	2.4706
2.3414	6.8767
3.6836	11.7008
4.9530	15.6005

Table 1: Some tabulated values of a function.

	x_i	0	1	2	3	4
0	0.7847	2.3610				
1	1.0320	2.4706	$Q_{1,1}$			
2	2.3414	6.8767	$Q_{2,1}$	$Q_{2,2}$		
3	3.6836	11.7008	$Q_{3,1}$	$Q_{3,2}$	$Q_{3,3}$	
4	4.9530	15.6005	$Q_{4,1}$	$Q_{4,2}$	$Q_{4,3}$	$Q_{4,4}$

Table 2: The Q table of a 4-th Lagrange interpolating polynomial.

Solution. For the points in Table 1, the Neville table Q described in Algorithm 3.1 is shown in Table 2. We already have the values $Q_{0,i}$ for $i = 0, \ldots, 4$. These are the $f(x_i)$ values

$$Q_{0,0} = 2.3610,\ Q_{0,1} = 2.4706,\ Q_{0,2} = 6.8767,\ Q_{0,3} = 11.7008,\ Q_{0,4} = 15.6005.$$

The remaining $Q_{i,j}$ values can be calculated recursively using Algorithm 3.1; these values are

$$\begin{aligned}
Q_{1,1} &= 0.443186413263243x + 2.01323162151233\\
Q_{2,1} &= 3.36497632503437x - 1.00205556743547\\
Q_{2,2} &= 1.87691264326532x^2 - 2.96660078575686x + 3.53317499992008\\
Q_{3,1} &= 3.59417374459842x - 1.53869840560274\\
Q_{3,2} &= 0.0864374036672401x^2 + 3.07338838750330x - 0.793194725306702\\
Q_{3,3} &= -0.617639532097720x^3 + 4.44511958168085x^2 - 6.09397642116479x\\
&\quad + 4.70427518391589\\
Q_{4,1} &= 3.07208129825114x + 0.384481329762090\\
Q_{4,2} &= -0.199912868106631x^2 + 4.79864877494087x - 3.26290312010083\\
Q_{4,3} &= -0.0730299086390897x^3 + 0.601809468933296x^2 + 1.98943605018381x\\
&\quad - 0.143172010839081\\
Q_{4,4} &= 0.130655092833680x^4 - 1.64219757357159x^3 + 7.10789767270635x^2\\
&\quad - 8.77864426201650x + 5.61682749870231
\end{aligned}$$

where the coefficients have been rounded off. We can also compute the required Lagrange interpolating polynomaial as follows:

```
sage: X = [0.7847, 1.0320, 2.3414, 3.6836, 4.9530]
sage: Y = [2.3610, 2.4706, 6.8767, 11.7008, 15.6005]
sage: PR = PolynomialRing(RR, "x")
sage: P = PR.lagrange_polynomial(zip(X,Y), algorithm="neville")[-1]
```

Thus the 4-th Lagrange interpolating polynomial that we seek is the polynomial $Q_{4,4}$ above. □

In practice, we usually do not require the full Neville table Q that results from using Algorithm 3.1. It is the entry $Q_{n,n}$ that we need, and in terms of computer implementation it is a waste of computer memory to generate the full table. Algorithm 3.2 presents a version of Neville's method that is more memory efficient than Algorithm 3.1. It does not generate the full Neville table, but only keeps track of both the current and previous rows of the table. After finish running, the algorithm returns the last row of the Neville table.

If the generated Lagrange interpolating polynomial $P(x)$ is not a reasonable approximation to $f(x)$, we can add k more interpolation points to get

$$(x_0, f(x_0)),\ (x_1, f(x_1)),\ \ldots,\ (x_n, f(x_n)),\ (x_{n+1}, f(x_{n+1})),\ \ldots,\ (x_{n+k}, f(x_{n+k})). \quad (3)$$

Input : The numbers $x_0, x_1, \ldots, x_n$. The $n+1$ values $f(x_0), f(x_1), \ldots, f(x_n)$.
Output: The n-th row of the Neville table Q where $P(x) = Q_{n,n}$.

Let Q be a vector of size $n+1$
$Q_0 \leftarrow f(x_0)$
$P \leftarrow Q$
for $i \leftarrow 1, 2, \ldots, n$ **do**
    $Q_0 \leftarrow f(x_i)$
    for $j \leftarrow 1, 2, \ldots, i$ **do**
        $Q_j \leftarrow \dfrac{(x - x_{i-j})Q_{j-1} - (x - x_i)P_{j-1}}{x_i - x_{i-j}}$
    end
    $P \leftarrow Q$
end
return Q

Algorithm 3.2: A memory efficient version of Neville's method.

Let Q be the Neville table of the n-th Lagrange interpolating polynomial $P(x)$ where $Q_{n,n} = P(x)$. Using the points in (3), we want to compute the $(n+k)$-th Lagrange interpolating polynomial $P'(x)$ without having to generate its Neville table. One solution is to use the last row of the Neville table for $P(x)$ and adapt Neville's method so that it starts its computation from the points

$$(x_{n+1}, f(x_{n+1})),\ (x_{n+2}, f(x_{n+2})),\ \ldots,\ (x_{n+k}, f(x_{n+k})).$$

Such a method is presented in Algorithm 3.3. This algorithm is essentially a modification of Algorithm 3.2 so that it can use the results of previous computation.

Example 3.4. *Let $f(x) = x^3\cos(x) - 2x$. Find a 2-nd Lagrange polynomial $P(x)$ that interpolates $f(x)$ at the points*

$$(x_0, y_0) = (0.5,\ f(x_0)), \quad (x_1, y_1) = (3.2,\ f(x_1)), \quad (x_2, y_2) = (1.7,\ f(x_2)).$$

Use $P(x)$ and the additional points

$$(x_3, y_3) = (5,\ f(x_3)), \quad (x_4, y_4) = (6,\ f(x_4))$$

to find a 4-th Lagrange interpolating polynomial $P'(x)$.

Solution. For the first three points, the 2-nd Lagrange interpolating polynomial that we seek is

$$P(x) = (-10.8765434414684x + 26.0869807210336)x - 11.2146566799134.$$

With the two additional points, the required 4-th Lagrange interpolating polynomial is

$$\begin{aligned} P'(x) = {} & -4.67639356528245x^4\cos(5) + 3.26185442464512x^4\cos(6) - 1.57644155638333x^4 \\ & + 53.3108866442200x^3\cos(5) - 33.9232860163093x^3\cos(6) + 20.7681601852390x^3 \\ & - 188.411896745230x^2\cos(5) + 113.806100875868x^2\cos(6) - 86.3086893150001x^2 \\ & + 234.100261878040x\cos(5) - 137.552401087285x\cos(6) + 115.285289941467x \\ & - 76.3187429854097\cos(5) + 44.3612201751737\cos(6) - 39.4532672476283. \end{aligned}$$

Input : The numbers $x_0, x_1, \ldots, x_n$. The $n+1$ values $f(x_0), f(x_1), \ldots, f(x_n)$. The last row R of the Neville table that results from computing the k-th Lagrange interpolating polynomial of the points $(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_k, f(x_k))$ where $0 < k < n$.

Output: The n-th row of the Neville table Q where $P(x) = Q_{n,n}$.

Let Q be a vector of size $n+1$ where $Q_i \leftarrow R_i$ for $i \leftarrow 0, 1, \ldots, k$
$P \leftarrow Q$
for $i \leftarrow k+1, k+2, \ldots, n$ **do**
    $Q_0 \leftarrow f(x_i)$
    for $j \leftarrow 1, 2, \ldots, i$ **do**
        $$Q_j \leftarrow \frac{(x - x_{i-j})Q_{j-1} - (x - x_i)P_{j-1}}{x_i - x_{i-j}}$$
    end
    $P \leftarrow Q$
end
return Q

Algorithm 3.3: A memory efficient version of Neville's method that uses the results of previous computation.

To visualize how well $P(x)$ and $P'(x)$ interpolate $f(x)$, we graph these three functions on one set of axes:

```
sage: f = x^3 * cos(x) - 2*x
sage: x0 = 0.5; y0 = f(x=x0)
sage: x1 = 3.2; y1 = f(x=x1)
sage: x2 = 1.7; y2 = f(x=x2)
sage: x3 = 5; y3 = f(x=x3)
sage: x4 = 6; y4 = f(x=x4)
sage: X = [x0, x1, x2]
sage: Y = [y0, y1, x2]
sage: PR = PolynomialRing(SR, "x")
sage: P(x) = PR.lagrange_polynomial(zip(X,Y), algorithm="neville")[-1]
sage: X = [x0, x1, x2, x3, x4]
sage: Y = [y0, y1, y2, y3, y4]
sage: P1(x) = PR.lagrange_polynomial(zip(X,Y), algorithm="neville")[-1]
sage: plot1 = plot(f, (x,0,6))
sage: plot2 = plot(P, (x,0,6), color="red")
sage: plot3 = plot(P1, (x,0,6), color="green")
sage: show(plot1 + plot2 + plot3)
```

The resulting plot is shown in Figure 3. □

References

[1] E. H. Neville. Iterative interpolation. *Journal of the Indian Mathematical Society*, 20:87–120, 1934.

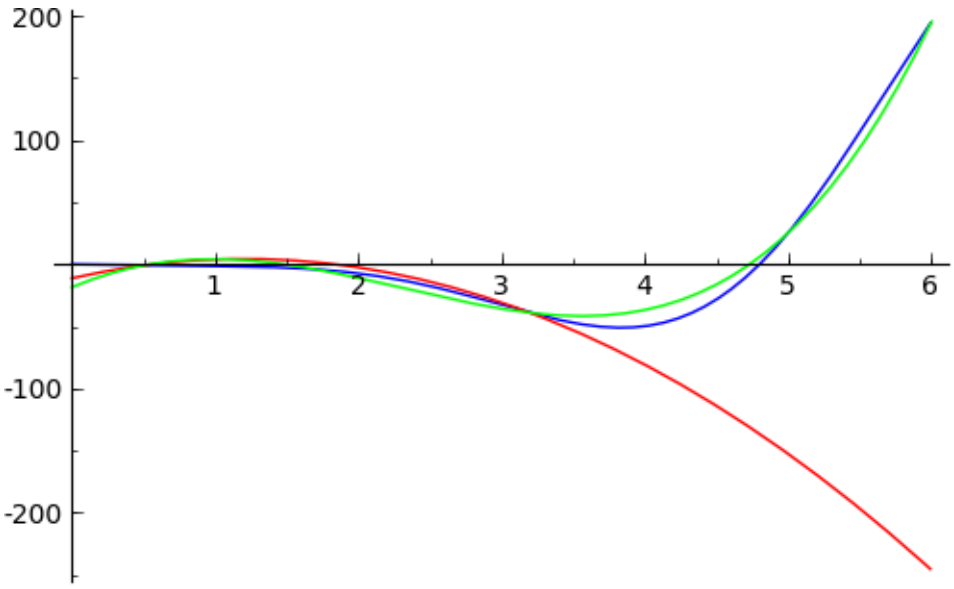

Figure 3: Interpolating $f(x) = x^3 \cos(x) - 2x$ using a quadratic and quartic polynomials.