13 Polynomial interpolation

13.1 Lagrange interpolation polynomial

Consider the following problem: Let $\boldsymbol{x} = (x_0, \ldots, x_n)$ and $\boldsymbol{y} = (y_0, \ldots, y_n)$ be two data sets, which represent points on the plane $(x_0, y_0), \ldots, (x_n, y_n)$ (hence $n+1$ points in total); is it possible to find some polynomial p, which satisfies

$$p(x_i) = y_i, \quad i = 0, \ldots, n?$$

(I.e., the polynomial passes exactly through these points). Why would I want doing something like this? Naturally, these two data sets can be interpreted as observations from some experiment of an unknown function f. If I am able to construct such polynomial then I hope I can *interpolate* the values of my function at other points $x \neq x_i$ by setting

$$f(x) \approx p(x),$$

(of course I need to be careful with how close I am to the actual function f).

If I am given only one point (x_0, y_0) then it is natural to consider the constant polynomial $p_0(x) = y_0$ (here and below the index in my polynomial indicates the degree) as my best interpolation; if I have two points (x_0, y_0), (x_1, x_2) it is natural to consider the straight line connecting these two points (linear interpolation):

$$p_1(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0),$$

etc. Therefore it seems plausible to assume that $n+1$ points one should be able to construct p_n, i.e., polynomial of degree n, that passes exactly through these points. And indeed this expectation is correct.

Theorem 13.1. *Let $\boldsymbol{x}, \boldsymbol{y}$ be as above with $x_i \neq x_j$ for $i \neq j$. Then there exists a unique polynomial p_n satisfying $p_n(x_i) = y_i, i = 0, \ldots, n$.*

Proof. The proof of this theorem is constructive: I will present a formula that actually builds the required polynomial.

Consider

$$L_k(x) = \frac{(x - x_0)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_0)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)} = \prod_{i=0,\, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}, \quad k = 0, \ldots, n.$$

By constructions L_k are polynomial of degree n and they satisfies

$$L_k(x_i) = \begin{cases} 0, & i \neq k, \\ 1, & i = k. \end{cases}$$

Now consider

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k. \tag{13.1}$$

Math 488/688: Numerical Analysis
e-mail: artem.novozhilov@ndsu.edu. Fall 2021

Clearly, p_n is a polynomial of degree at most n (as the sum of polynomials of degree n), and, moreover,

$$p_n(x_k) = y_k, \quad k = 0, \ldots, n,$$

as required.

What is left to show is that this polynomial is unique. Assume that there is another polynomial q_n such that $q_n(x_i) = y_i$ for all i. Consider their difference

$$p_n(x) - q_n(x) = g_n(x).$$

I have that $g_n(x_i) = y_i - y_i = 0$, that is g_n has $n+1$ roots. By the fundamental theorem of algebra, a nonconstant polynomial of degree n can have at most n distinct rules, therefore g_n is identically zero, and hence $p_n = q_n$, i.e., the required polynomial is unique. ■

Definition 13.2. *The polynomial*

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k = \sum_{k=0}^{n} \left(\prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \right) y_k$$

is called the Lagrange interpolation polynomial.

Example 13.3. Let $f(x) = e^x$ and consider $\boldsymbol{x} = (-1, 0, 1)$. My $\boldsymbol{y}$ data are then $\boldsymbol{y} = (e^{-1}, 1, e)$.

According to the formula above, I have

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{1}{2} x(x-1), \quad L_1(x) = 1 - x^2, \quad L_2(x) = \frac{1}{2} x(x+1).$$

Therefore my Lagrange interpolation polynomial is given by

$$p_2(x) = \frac{1}{2} x(x-1)e^{-1} + (1 - x^2) + \frac{1}{2} x(x+1)e = 1 + (\sinh 1)x + (\cosh 1 - 1)x^2,$$

see the figure.

In the same figure it can be seen that the difference between f and p_2 is quite noticeable, but the expectation is that if I increase the number of points, the error I am making in replacing f with its interpolation polynomial will decrease. For this particular example this is true, see the figures.

And indeed, for instance, while computing $p_4(0.5)$ instead of $f(0.5)$ I make a relative error in only 0.3%, which is quite acceptable for many purposes. Unfortunately, this kind of nice behavior for the interpolation polynomials is not a norm as the following example shows.

Example 13.4. For this example I will take the classical

$$f(x) = \frac{1}{1 + x^2}$$

on $[-5, 5]$. I will be using equally spaced points on $[-5, 5]$, i.e., if $n+1$ is the number of points then

$$x_i = a + i\frac{b-a}{n}, \quad i = 0, \ldots, n, \quad y_i = f(x_i).$$

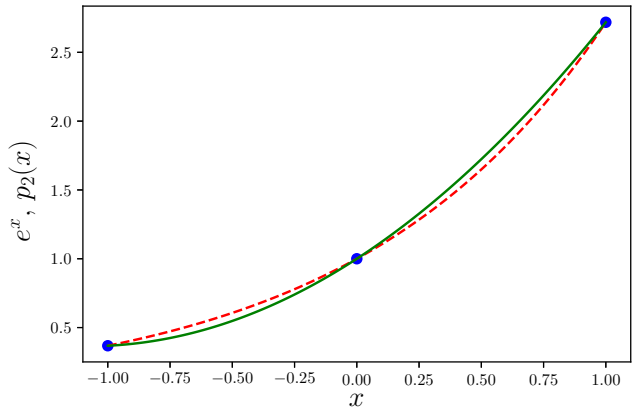

Figure 1: Comparison of the function $f(x) = e^x$ (red) and its Lagrange interpolation polynomial $p_2(x)$ (green) built by 3 points (blue circles) on $[-1, 1]$.

Here are the illustrations for $n = 4, 6, 8, 10, 12, 14$.

A perfunctory inspection indicates that despite the number of interpolation points increase the error I am making, namely,

$$\max_{x\in[-5,5]} \left| \frac{1}{1+x^2} - p_n(x) \right|,$$

also increases (opposite to our naive expectation).

It can be shown that the following inequality holds:

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!} |\pi_{n+1}(x)|, \quad x \in [a, b], \tag{13.2}$$

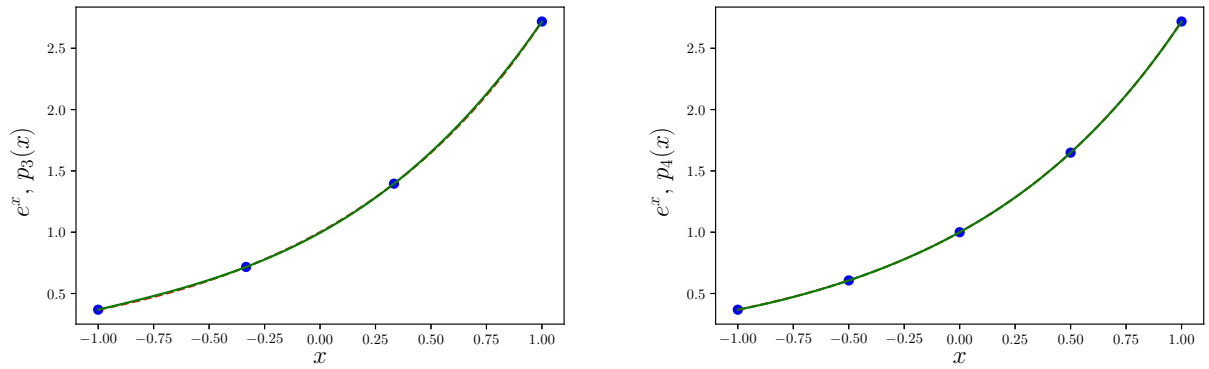

Figure 2: Comparison of the function $f(x) = e^x$ (red) and its Lagrange interpolation polynomials p_3 (left) and p_4 (right) on $[-1, 1]$.

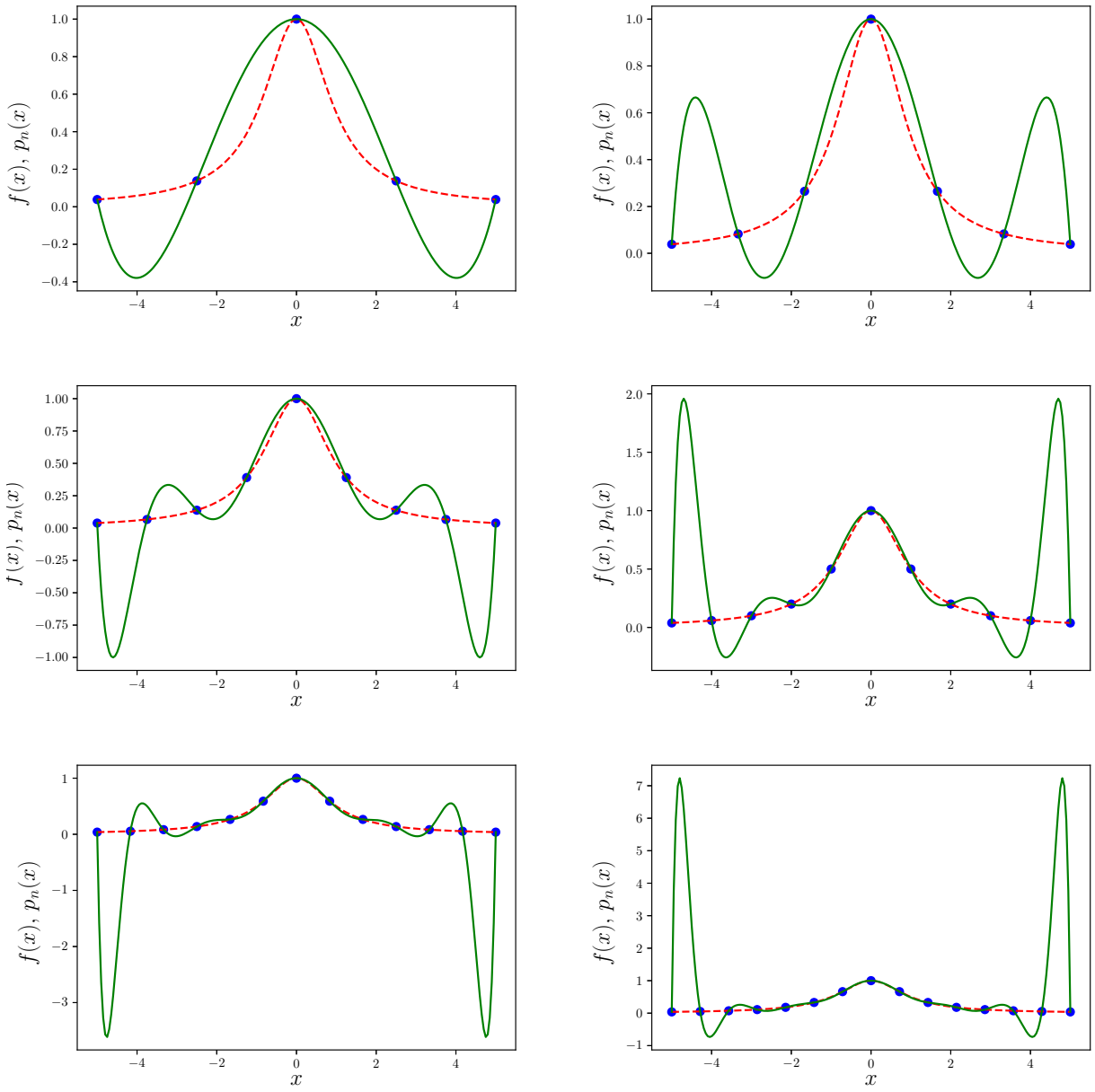

Figure 3: Comparison of the function $f(x) = \frac{1}{1+x^2}$ (red) and its Lagrange interpolation polynomials $p_4, p_6, p_8, p_{10}, p_{12}, p_{14}$ on $[-5, 5]$.

where $\pi_{n+1}(x) = (x - x_0)(x - x_1)\dots(x - x_n)$ is a polynomial of degree $n + 1$ and

$$M_{n+1} = \max_{x\in[a,b]} \left| f^{(n+1)}(x) \right|$$

is the maximum of the $(n+1)$-st derivative of f on the given interval $[a, b]$. For the function $f(x) = \frac{1}{1+x^2}$ it is clear that the sequence $M_{n+1} \max_x |\pi_{n+1}(x)|$ goes to infinity faster then the sequence $1/(n + 1)!$

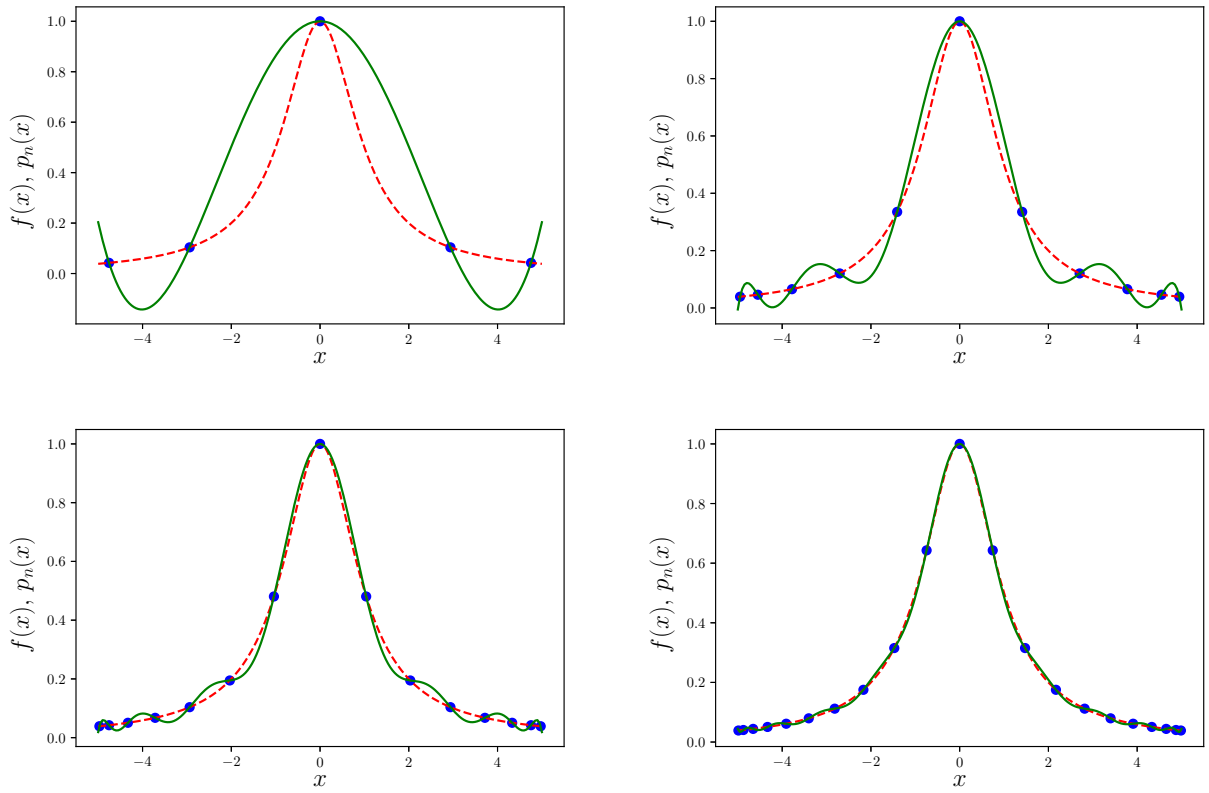

Figure 4: Comparison of the function $f(x) = \frac{1}{1+x^2}$ (red) and its Lagrange interpolation polynomials p_n for $n = 4, 10, 14, 20$ on $[-5, 5]$. The interpolation points are chosen as in (13.3).

approaches zero, and hence increasing the order of the interpolation polynomial implies bigger and bigger error.

Can I somehow mitigate this phenomenon? The answer is surprising "yes," and I will show only the final result, since the actual mathematical explanation how it all "works," is beyond these lecture notes.

Let me take different interpolation points instead of equally spaced $x_0, \ldots, x_n$, which I used before. Namely, I choose

$$x_j = \frac{b-a}{2} \cos \frac{(j+0.5)\pi}{n+1} + \frac{b+a}{2}, \quad j = 0, \ldots, n. \tag{13.3}$$

For this choice my interpolation polynomial behaves nicer, see the figure.

13.2 Python code

In the examples above I used the direct implementation of (13.1). I am planning to add a more sophisticated method later. Here I would like to note that when dealing with Lagrange interpolation polynomial it is better to evaluate $p_n(x)$ at a given point x using (13.1) instead of first computing the coefficients of your polynomial and then using these coefficients to calculate $p_n(x)$.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def lagrange_interpolation_polynomial(x: float, x_data: list, y_data: list):

    n = len(x_data)

    tmp = 0
    for i in range(n):

        prod = 1
        for j in range(n):
            if i != j:
                prod *= (x - x_data[j])/(x_data[i] - x_data[j])

        tmp += prod * y_data[i]

    return tmp

def f(x):
    return np.exp(x)

a = -1
b = 1
number_of_points = 4

x_data = np.linspace(a, b, number_of_points)
y_data = [f(i) for i in x_data]
x = np.linspace(-1, 1, 100)
y = [f(i) for i in x]
yy = [lagrange_interpolation_polynomial(i, x_data, y_data) for i in x]

plt.rc('text', usetex=True)
plt.plot(x_data,y_data,'bo')
plt.plot(x,y,'r--') #add my line to the previous figure
plt.plot(x,yy,'g-') #add my line to the previous figure
plt.xlabel(r'\LARGE $x$')
plt.ylabel(r'\LARGE $e^x,\,p_3(x)$')
plt.savefig('fig13_2.eps', format='eps')
```

13.3 Computing the derivatives

The Lagrange interpolation polynomial can be used to compute the derivative of given function f that is given in a table form. From (13.1) one can immediately see that

$$p_n'(x) = \sum_{k=0}^{n} L_k'(x) y_k,$$

hence all I need is to calculate

$$L_k'(x) = \sum_{i=0, i\neq k}^{n} \frac{1}{x_k - x_i} \prod_{m=0, m\neq i, m\neq k}^{n} \frac{x - x_m}{x_k - x_m}\,.$$

It is possible to show that under quite general conditions for sufficiently large n $p_n'(x)$ is getting close to $f'(x)$, which gives some hope that one can always estimate the derivative. This is not true in general. Without going into any technical details, I would like to mention only that numerical differentiation is a numerically unstable procedure, and should always be performed with caution, since it is usually true that any measurements errors as well as inevitable roundoff errors due to floating point arithmetics will be magnified during numerical differentiation.

13.4 Coefficients of interpolation polynomial

I mentioned that it is rarely one needs the explicit expressions for the coefficients of the interpolation polynomial. The value $p_n(x)$ is usually computed directly, without the knowledge of the explicit form of p_n. Assume for a second that for some unknown reason need these coefficients. How to find them? Probably the most natural way is to start with the polynomial

$$p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots a_1 x + a_0$$

and realize that the data we have allows to write our problem as a system of $n+1$ linear equations with $n+1$ unknowns $(a_0, \ldots, a_n)$, in the matrix form

$$\begin{bmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ \vdots & & \ddots & & \\ x_{n-1}^n & x_{n-1}^{n-1} & \ldots & x_{n-1} & 1 \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{bmatrix} \cdot \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

The matrix of this system has a special form and is called *Vandermonde matrix*. When solving this system one has to remember that Vandermonde matrix can be ill-conditioned even for moderate values of n, so care should be exercised.