Math104A Homework2

November 3, 2024

Brandon Su

1. (a) The general Lagrangian form of the interpolation is

$$P_n = \sum_{j=0}^{n} f_j \frac{\prod_{k\neq j}(x-x_k)}{\prod_{k\neq j}(x_j-x_k)}.$$

Hence,

$$\begin{aligned}
P_2(x) &= \sum_{j=0}^{2} f_j \frac{\prod_{k\neq j}(x-x_k)}{\prod_{k\neq j}(x_j-x_k)} \\
&= 1\times\frac{(x-1)(x-3)}{(0-1)(0-3)} + 1\times\frac{(x-0)(x-3)}{(1-0)(1-3)} - 5\times\frac{(x-0)(x-1)}{(3-0)(3-1)} \\
&= \frac{x^2-4x+3}{3} + \frac{x^2-3x}{-2} - 5\frac{x^2-x}{6} \\
&= -x^2+x+1
\end{aligned}$$

(b) $f(2) \approx P_2(2) = -2^2+2+1 = -1$

2. (a) Barycentric Formula

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[6]:
```
def Barycentric_Weights(xs):
    '''
    Input: xs: A list of interpolation nodes (x_j)
    Ouput: The barycentric weights for each node: lambda_j
    '''
    n = len(xs)
    lambda_j = np.ones(n)
    for j in range(n):
        for k in range(n):
            if k != j:
                lambda_j[j] = lambda_j[j] / (xs[j] - xs[k])
    return lambda_j
```

[19]:
```python
def Barycentric_Formula(xs, fs, lambda_j, x):
    '''
    Inputs:
    xs: A list of interpolation nodes (x_j)
    y_values: f(x_j)
    lambda_weights: the weights calculated by the function Barycentric_Weights,
 ↪lambda_j
    x: the point to evaluate the interpolation P_n(x)
    Ouput: value of the approximation at the point x
    '''
    num = 0
    dem = 0
    for j in range(len(xs)):
        if x == xs[j]:
            return fs[j]
        l_jx = lambda_j[j] / (x - xs[j])
        num +=  fs[j] *l_jx
        dem += l_jx
    return num / dem
```

Test the implementation: Let $xs = [0, 1, 3]$ and use Barycentric_Weights function to calculate the weights. Then let $fs = [1, 1, -5]$, $x = 2$ and plug them all in the Barycentric_Formula function.

[105]:
```python
w = Barycentric_Weights([0, 1, 3])
Barycentric_Formula([0,1,3], [1, 1, -5], w, 2)
```

[105]: -0.9999999999999998

As we can see, the test result is really close to the one we calculated by hand.

(b) Use the code in (a) to find $P_5(2)$ as an approximation of $f(2)$.

[8]:
```python
x_j = [0.00, 0.25, 0.50, 0.75, 1.25, 1.50]
f_xj = [0.0000, 0.7071, 1.0000, 0.7071, -0.7071, -1.0000]
weights = Barycentric_Weights(x_j)
f2 = Barycentric_Formula(x_j, f_xj, weights, 2)
f2
```

[8]: 0.8519999999999989

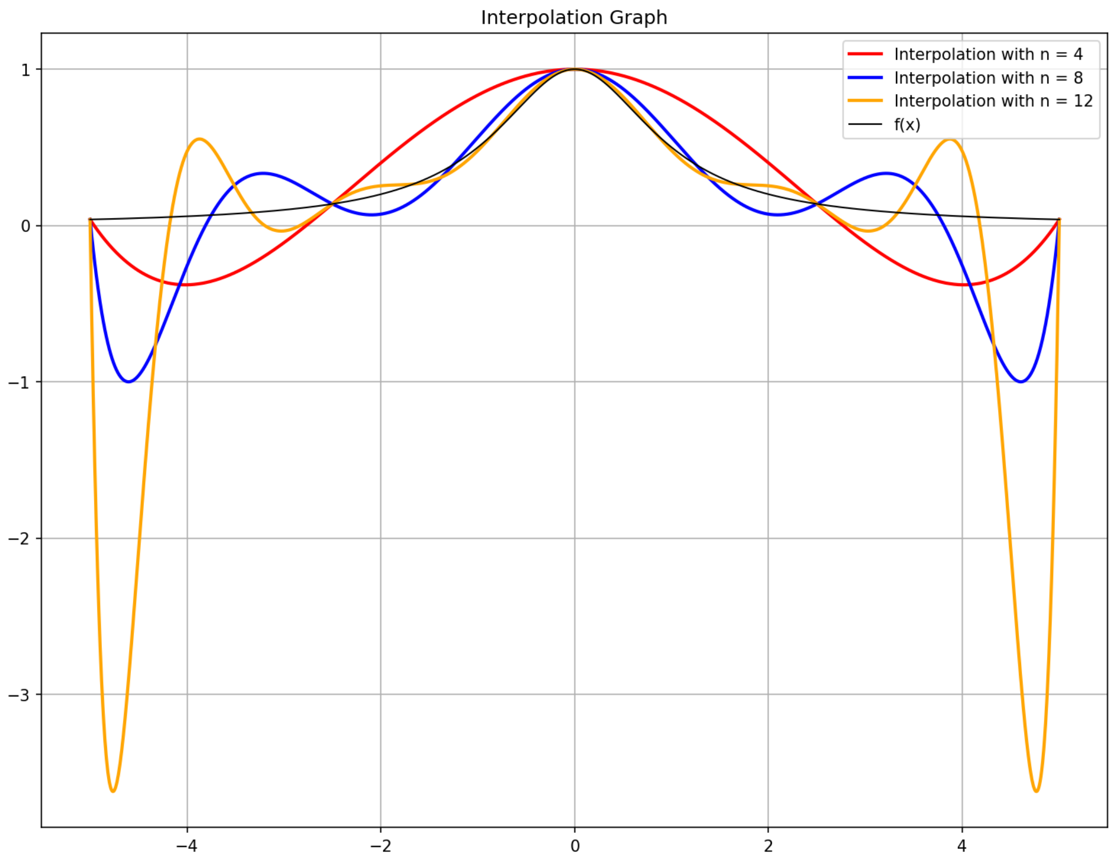
The approximation of $f(2)$ by $P_5(2)$ is 0.8519999999999989.

3. The Runge Example.

[104]:
```python
from scipy.special import binom

def plotting(ns, barycentric_function, nodes_function, function):
    xs = np.linspace(-5, 5, 5000)
    fxs = function(xs)
    plt.figure(figsize=(12, 10), dpi=150)
    color = {4:"red", 8:"blue", 12:"orange", 100:"green"}
    for n in ns:
        xj = nodes_function(n)
        lambdaj = barycentric_function(n)
        fj = function(np.array(xj))
        Pjs = []

        for i in xs:
            Pj = Barycentric_Formula(xj, fj, lambdaj, i)
            Pjs.append(Pj)
        plt.plot(xs, Pjs, color=color[n], linewidth=2, label=f"Interpolation␣
↪with n = {n}")

    plt.plot(xs, fxs, color="black", linewidth=1, label="f(x)")
    plt.title("Interpolation Graph")
    plt.legend()
    plt.grid(True)
    plt.show()
```

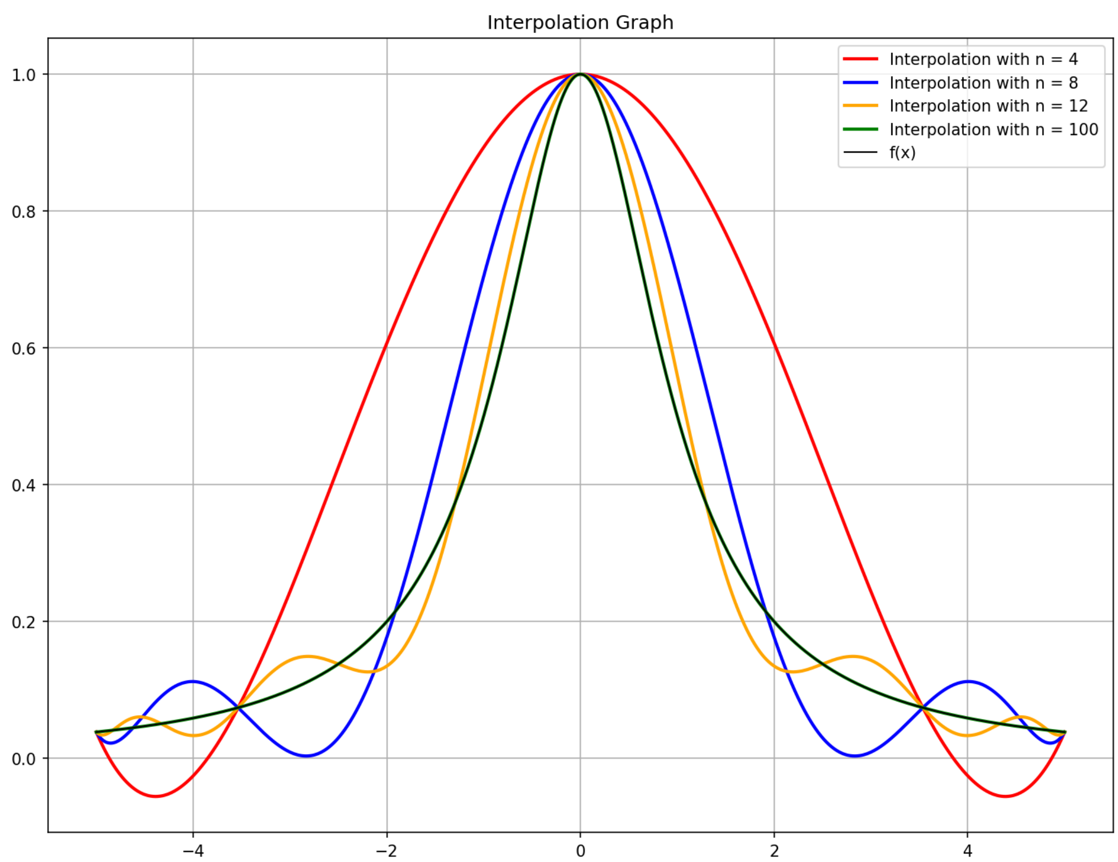
(a) the equidistributed nodes $x_j = -5 + j(10/n)$, $j = 0, \dots, n$ for $n = 4, 8$, and 12.

[102]:
```
def equidistributed_barycentric(n):
    weights = np.ones(n + 1)
    for j in range(n + 1):
        weights[j] = (-1) ** j * binom(n, j)
    return weights

def nodes_a(n):
    nodes = []
    for j in range(n + 1):
        nodes.append(-5 + j * (10 / n))
    return nodes

def f_1(x):
    return 1 / (1 + x**2)

plotting([4, 8, 12], equidistributed_barycentric, nodes_a, f_1)
```



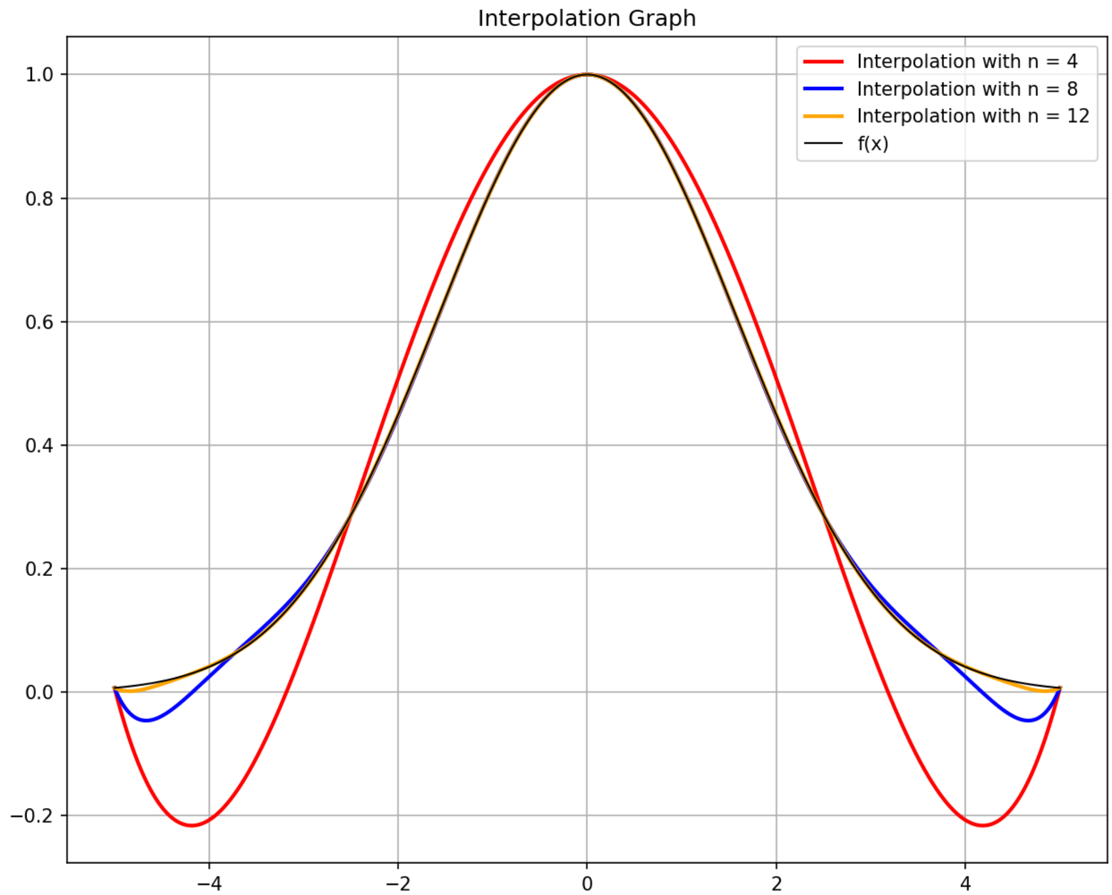
(b) the nodes $x_j = 5\cos(\frac{j\pi}{n})$, $j = 0, \ldots, n$ for $n = 4, 8, 12$, and 100.

[103]:
```
def b_barycentric(n):
    weights = np.ones(n + 1)
    for j in range(1, n):
        weights[j] = (-1) ** j
    weights[0] *= 1 / 2
    weights[-1] *= 1 / 2
    return weights

def nodes_b(n):
    nodes = []
    for j in range(n + 1):
        nodes.append(5 * np.cos((j * np.pi) / n))
    return nodes

plotting([4, 8, 12, 100], b_barycentric, nodes_b, f_1)
```



(c) Repeat (a) for $f(x) = e^{-x^2/5}$ for $x \in [-5, 5]$ and comment on the result.

[88]:
```
def f_2(x):
    return np.e**(-x**2 / 5)

plotting([4, 8, 12], equidistributed_barycentric, nodes_a, f_2)
```



We can see that when using the same equidistributed nodes for $f(x) = e^{-x^2/5}$ over the same interval $[-5, 5]$, the interpolation does a better job. At $n = 12$, the interpolation graph closely aligns with the actual function $f(x)$, suggests a good fit for the function. Unlike the Runge example $f(x) = \frac{1}{1+x^2}$, which exhibits sharper and sudden changes near the edges, the interpolation for this function displays a smooth curve.

Compare (a) and (b): With equidistributed nodes, the interpolation polynomial exhibits oscillations near the boundary points, which becomes more violent as n increases. By using nodes in (b), the interpolation is much smoother and exhibits fewer oscillations near the edges, which improves the accuracy a lot, indicating a better approximation. And we can see when $n = 100$, the interpolation graph is almost identical to the actual function. Also, we can observe that for both nodes, the interpolation accuracy remains high in the central region of the interval.