1 Introduction

In numerical methods, approximating continuous functions is a fundamental problem. Polynomial interpolation offers a powerful approach to achieve this, where we construct a polynomial that passes through a given set of points sampled from the function. However, the selection of these interpolation points significantly impacts the accuracy of the polynomial approximation. This document explores the critical role of interpolation point choice, examining phenomena like Runge's phenomenon and introducing Chebyshev interpolation as a strategy to enhance approximation accuracy. We will also discuss the Lebesgue constant as a tool to analyze the stability and convergence of interpolation schemes, independent of the specific function being approximated.

2 Polynomial Interpolation Error

Given a continuous function $f(x)$ defined on an interval $[a,b]$, and a set of interpolation points $x_0, x_1, \ldots, x_n$ within this interval, we can construct a polynomial interpolant $p_n(x)$ of degree n that passes through the points $(x_i, f(x_i))$. The error between the function $f(x)$ and its polynomial interpolant $p_n(x)$ at a point x is given by the formula:

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\theta)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

where $f^{(n+1)}(\theta)$ denotes the $(n+1)$-th derivative of f evaluated at some point θ in the interval (a,b), and $\prod_{i=0}^{n}(x - x_i)$ is the product of terms $(x - x_i)$ for each interpolation point x_i.

Here,

- $f(x) - p_n(x)$: represents the interpolation error at a point x.
- $f^{(n+1)}(\theta)$: is the $(n+1)$-th derivative of the function f at some unknown point θ within the interval (a,b). This term reflects how smooth the function f is.
- $(n+1)!$: is the factorial of $(n+1)$, which scales down the error as the degree of the polynomial n increases.
- $\prod_{i=0}^{n}(x - x_i)$: is the product term that depends on the interpolation points x_i. This term vanishes at each interpolation point $x = x_i$, ensuring that $p_n(x_i) = f(x_i)$.

2.1 Proof for $n = 1$

Let's prove the interpolation error formula for the case $n = 1$. Consider a function f that is twice differentiable on an interval $[a,b]$. Let $p_1(x)$ be the linear interpolant of $f(x)$ based on interpolation points x_0 and x_1. We define a function $q(x)$ as:

$$q(x) = p_1(x) + \lambda(x - x_0)(x - x_1)$$

where λ is a constant to be determined. Note that the quadratic term $(x - x_0)(x - x_1)$ vanishes at x_0 and x_1, so $q(x_0) = p_1(x_0) = f(x_0)$ and $q(x_1) = p_1(x_1) = f(x_1)$.

Now, fix an arbitrary point $x_{\text{hat}} \in (x_0, x_1)$. We choose λ such that $q(x_{\text{hat}}) = f(x_{\text{hat}})$. This gives us:

$$f(x_{\text{hat}}) = p_1(x_{\text{hat}}) + \lambda(x_{\text{hat}} - x_0)(x_{\text{hat}} - x_1)$$

Solving for λ:

$$\lambda = \frac{f(x_{\text{hat}}) - p_1(x_{\text{hat}})}{(x_{\text{hat}} - x_0)(x_{\text{hat}} - x_1)}$$

Define the error function $e(x) = f(x) - q(x)$. By construction, $e(x_0) = f(x_0) - q(x_0) = 0$, $e(x_1) = f(x_1) - q(x_1) = 0$, and $e(x_{\text{hat}}) = f(x_{\text{hat}}) - q(x_{\text{hat}}) = 0$. Thus, $e(x)$ has at least three roots in the interval $[x_0, x_1]$ at x_0, x_1, and x_{hat}.

By Rolle's Theorem, since $e(x)$ has three roots, its first derivative $e'(x)$ must have at least two roots in (x_0, x_1), and its second derivative $e''(x)$ must have at least one root in (x_0, x_1). Let $\theta \in (x_0, x_1)$ be a root of $e''(x)$, so $e''(\theta) = 0$.

Now we compute the second derivative of $e(x)$:

$$e''(x) = \frac{d^2}{dx^2}[f(x) - q(x)] = f''(x) - q''(x)$$

Since $q(x) = p_1(x) + \lambda(x - x_0)(x - x_1)$ and $p_1(x)$ is linear, $p_1''(x) = 0$. Also, $\frac{d^2}{dx^2}[(x - x_0)(x - x_1)] = \frac{d^2}{dx^2}[x^2 - (x_0 + x_1)x + x_0x_1] = 2$. Thus, $q''(x) = 2\lambda$.

Therefore, $e''(x) = f''(x) - 2\lambda$. Since $e''(\theta) = 0$, we have $f''(\theta) - 2\lambda = 0$, which implies $\lambda = \frac{f''(\theta)}{2}$.

Substituting this value of λ back into the expression for λ:

$$\frac{f''(\theta)}{2} = \frac{f(x_{\text{hat}}) - p_1(x_{\text{hat}})}{(x_{\text{hat}} - x_0)(x_{\text{hat}} - x_1)}$$

Rearranging this, we get the error formula for $n = 1$ at x_{hat}:

$$f(x_{\text{hat}}) - p_1(x_{\text{hat}}) = \frac{f''(\theta)}{2!}(x_{\text{hat}} - x_0)(x_{\text{hat}} - x_1)$$

Since x_{hat} was an arbitrary point in (x_0, x_1), this formula holds for any $x \in (x_0, x_1)$ with some $\theta \in (x_0, x_1)$. This proves the error formula for $n = 1$.

3 Runge's Phenomenon

Definition 3.1 (Runge's Phenomenon)**.** Runge's Phenomenon refers to the problem encountered when interpolating certain functions, particularly high-degree polynomials with equally spaced interpolation points. In such cases, increasing the degree of the polynomial does not necessarily lead to better approximation; instead, oscillations can develop near the edges of the interval, and the interpolant may diverge from the function.

A famous example demonstrating Runge's phenomenon is given by the Runge function:

$$f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1]$$

When we interpolate this function using equally spaced points, we observe that as the number of interpolation points increases, the polynomial interpolant oscillates wildly near the boundaries of the interval $[-1, 1]$.

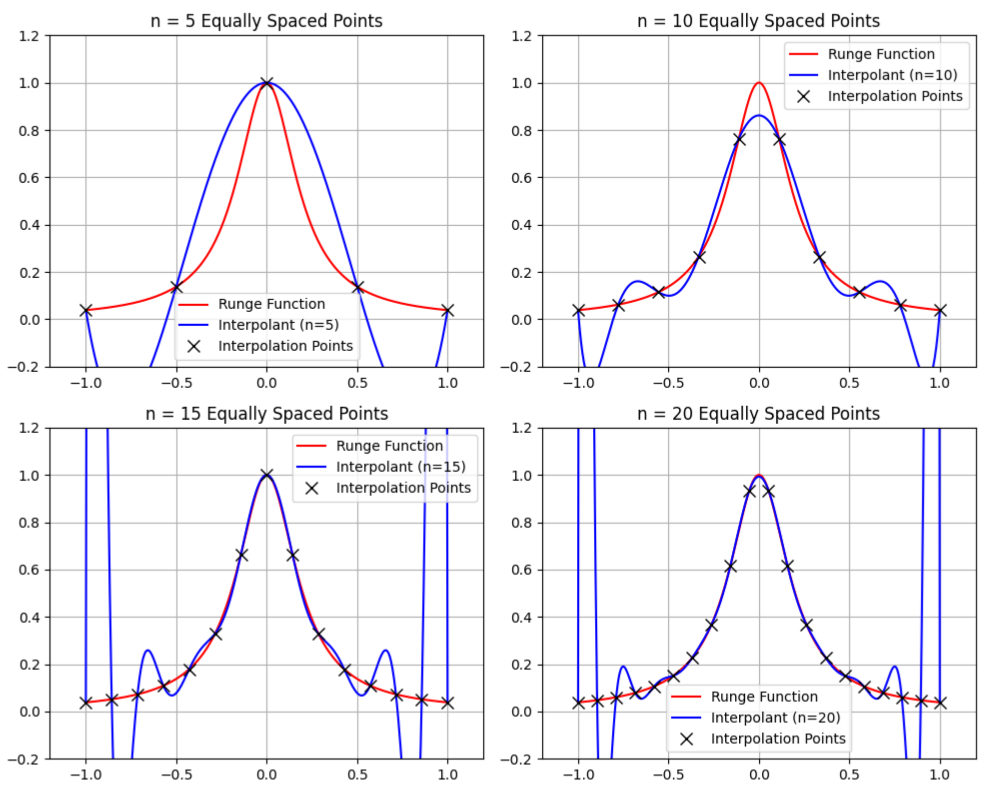

Figure 1: Runge's Phenomenon: Polynomial interpolation of the Runge function using equally spaced points for increasing polynomial degrees. Oscillations at the edges become more pronounced with higher degrees.

As illustrated in Figure 1, for low degrees, the interpolant reasonably approximates the Runge function. However, as the degree increases (n=10, 15, 20), oscillations near $x = \pm 1$ become increasingly severe. Instead of converging to the Runge function, the polynomial interpolant diverges at the edges of the interval.

To quantify the approximation error over the entire interval, we use the infinity norm (or maximum norm):

Definition 3.2 (Infinity Norm)**.** The infinity norm of the error between a function $f(x)$ and its polynomial interpolant $p_n(x)$ on the interval $[-1, 1]$ is defined as:

$$||f - p_n||_\infty = \max_{x \in [-1,1]} |f(x) - p_n(x)|$$

For Runge's function with equally spaced points, the infinity norm of the error actually increases exponentially as the degree of the polynomial increases. This indicates that polynomial interpolation with equally spaced points is not a reliable method for approximating continuous functions in general, especially for higher degrees.

4 Chebyshev Interpolation

To mitigate Runge's phenomenon, we can strategically choose interpolation points to minimize the error term. Recall the error formula:

$$|f(x) - p_n(x)| = \left| \frac{f^{(n+1)}(\theta)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \right|$$

To minimize the error, especially when we have limited control over the derivatives of f, we can focus on minimizing the polynomial term $\prod_{i=0}^{n}(x - x_i)$ in the interval $[-1, 1]$.

Theorem 4.1 (Approximation Theory Result)**.** For $x \in [-1, 1]$, the minimum value of the infinity norm of polynomials of the form $\prod_{i=0}^{n}(x - x_i)$ is $1/2^n$. This minimum value is achieved when the roots $\{x_i\}_{i=0}^{n}$ are the roots of the Chebyshev polynomial of the first kind of order $n + 1$, $T_{n+1}(x)$, scaled by $1/2^n$. The monic polynomial achieving this minimum is $T_{n+1}(x)/2^n$.

Definition 4.2 (Chebyshev Polynomials of the First Kind)**.** Chebyshev polynomials of the first kind, $T_n(x)$, can be defined in two equivalent ways:

1. **Cosine Definition:** $T_n(x) = \cos(n \arccos(x))$ for $x \in [-1, 1]$.
2. **Recurrence Relation:**

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x), \quad n \geq 1 \end{aligned}$$

The Chebyshev polynomial $T_{n+1}(x)$ of order $n+1$ has $n+1$ roots. We use these roots as our interpolation points. The roots of $T_n(x)$ are given by:

$$x_j = \cos\left(\frac{2j-1}{2n}\pi\right), \quad j = 1, 2, \ldots, n$$

These points are called **Chebyshev points**. They are not equally spaced; instead, they are clustered more densely near the endpoints ± 1 of the interval $[-1, 1]$.

Using Chebyshev points for polynomial interpolation, especially for smooth functions, leads to significantly faster convergence and avoids the severe oscillations seen with equally spaced points.

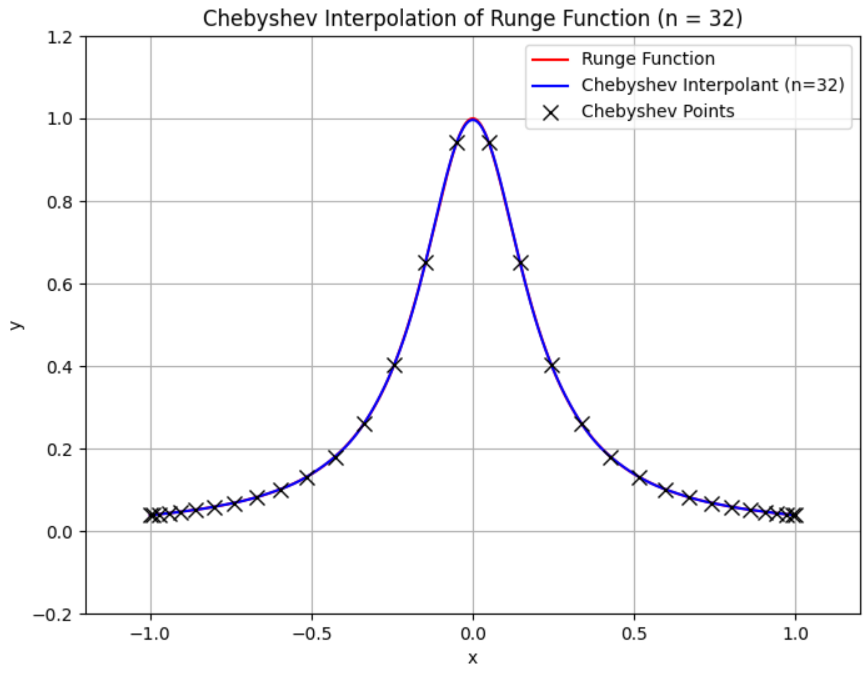

Figure 2: Chebyshev Interpolation of the Runge function with n=32 points. The interpolant closely approximates the Runge function, with Chebyshev points clustered near the interval ends.

Figure 2 shows the polynomial interpolant of the Runge function using Chebyshev points with $n = 32$. The approximation is significantly improved compared to using equally spaced points. The oscillations are effectively controlled, and the interpolant is almost indistinguishable from the original Runge function. Notice the Chebyshev points are clustered towards the ends of the interval $[-1, 1]$. This clustering is key to controlling the polynomial term in the error formula and reducing oscillations.

5 Code Example and Convergence

The provided Python code `ch_inter.py` (Listing 1) demonstrates and compares polynomial interpolation using both linearly spaced points and Chebyshev points. It interpolates a chosen function (Runge function or absolute value function) and plots the results.

Listing 1: Listing 1: `ch_inter.py`

```
#!/usr/bin/python

from math import *
import matplotlib.pyplot as plt
import numpy as np

# Function to interpolate
def f(x):
    return 1/(1+25*x**2)
    #return abs(x)

# Function to evaluate the Lagrange Interpolation
def lagr(xp,yp,x):
    lm=0.
    for k in range(xp.size):
        xc=xp[k]
        L=1.
        for l in range(xp.size):
            if l != k:
                L=L*(x-xp[l])/(xp[k]-xp[l])
        lm=lm+yp[k]*L
    return lm

# Control points
n=16
xp=np.linspace(-1.,1.,n) # (Linearly spaced)
#xp=np.array([cos((2*j-1)*pi/(2*n)) for j in range(n)]) # (Chebyshev points)
yp=np.array([f(q) for q in xp])

# Sample points
xx=np.linspace(-1.,1.,500)
yy=np.array([lagr(xp,yp,q) for q in xx])
yyf=np.array([f(q) for q in xx])

# Plot figure using Matplotlib
plt.figure(1)
plt.plot(xx,yy,'-',label='Interpolant')
plt.plot(xx,yyf,'-',label='Function')
plt.plot(xp,yp,'*')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

The code allows switching between linearly spaced points and Chebyshev points by commenting/uncommenting lines. It also allows switching between the Runge function and the absolute value function. Running this code with the Runge function and Chebyshev points illustrates the effective interpolation achieved.

To compare the convergence rates for different functions, particularly for the Runge function (smooth) and $f(x) = |x|$ (not smooth), we can examine the infinity norm of the error as the number of interpolation points increases. The code `ch_inter.py` can be adapted to calculate and plot these convergence rates, although it's not explicitly shown in the provided code snippet. The lecture mentions that for the Runge function, convergence is very rapid with Chebyshev points, while for $|x|$, convergence is slower due to the lack of smoothness (non-differentiability at $x = 0$).

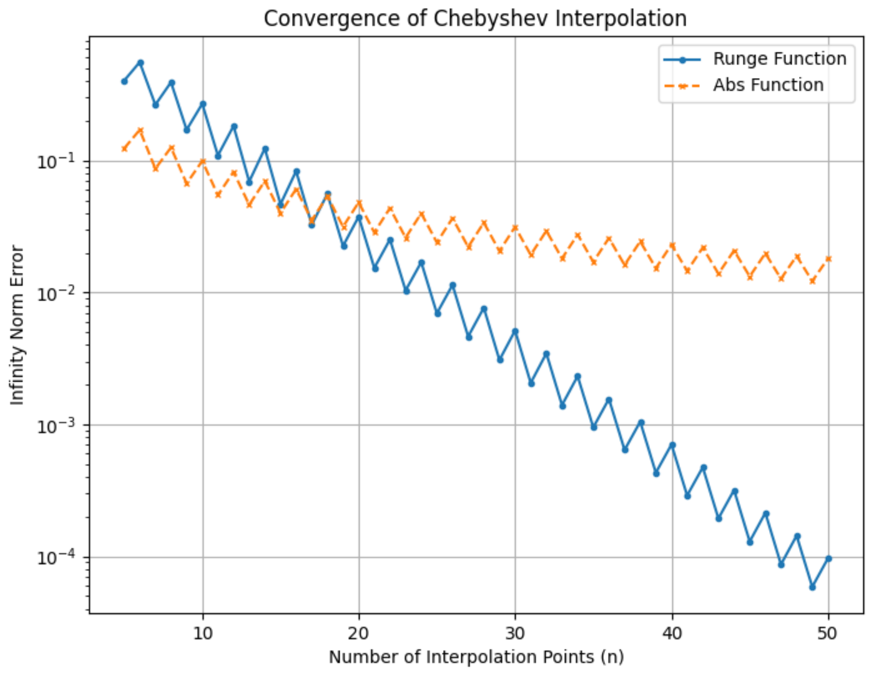

Figure 3: Convergence of Chebyshev Interpolation for Runge Function (smooth) and Absolute Value Function (not smooth). The error is plotted against the number of interpolation points on a semi-logarithmic scale.

Figure 3 demonstrates the convergence rates. The Runge function exhibits much faster convergence (exponential convergence) compared to the absolute value function. This highlights that the smoothness of the function significantly impacts the convergence rate of polynomial interpolation, even when using Chebyshev points.

6 The Lebesgue Constant

The error bound derived from the derivative of f depends on the function itself, as $f^{(n+1)}(\theta)$ varies with f. To obtain a measure of the quality of interpolation points that is independent of the function f, we introduce the Lebesgue constant.

Definition 6.1 (Lebesgue Constant)**.** For a set of interpolation points $X = \{x_0, x_1, \ldots, x_n\}$ on an interval $[a, b]$, the Lebesgue constant $\Lambda_n(X)$ is defined as:

$$\Lambda_n(X) = \max_{x\in[a,b]} \sum_{k=0}^{n} |L_k(x)|$$

where $L_k(x)$ are the Lagrange basis polynomials associated with the interpolation points X.

The Lebesgue constant is always greater than or equal to 1. To see this, consider any interpolation point $x_j \in X$. Then,

$$\sum_{k=0}^{n} |L_k(x_j)| = |L_j(x_j)| + \sum_{k\neq j} |L_k(x_j)| = |1| + \sum_{k\neq j} |0| = 1$$

Since $\Lambda_n(X)$ is the maximum of this sum over all $x \in [a, b]$, it must be at least 1.

Polynomial interpolation can be viewed as a linear operator I_n that maps a continuous function f to its polynomial interpolant $p_n = I_n(f)$. We can express $I_n(f)$ in terms of Lagrange basis polynomials:

$$I_n(f)(x) = p_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x)$$

The Lebesgue constant provides a bound on the operator norm of I_n. Let's consider the infinity norm of $I_n(f)$:

$$||I_n(f)||_\infty = \max_{x\in[a,b]} |I_n(f)(x)| = \max_{x\in[a,b]} \left|\sum_{k=0}^{n} f(x_k)L_k(x)\right|$$

Using the triangle inequality, we have:

$$||I_n(f)||_\infty \le \max_{x\in[a,b]} \sum_{k=0}^{n} |f(x_k)||L_k(x)|$$

Since $|f(x_k)| \le \max_{x\in[a,b]} |f(x)| = ||f||_\infty$, we can write:

$$||I_n(f)||_\infty \le ||f||_\infty \max_{x\in[a,b]} \sum_{k=0}^{n} |L_k(x)| = \Lambda_n(X)||f||_\infty$$

Thus, the Lebesgue constant $\Lambda_n(X)$ bounds the operator norm of I_n.

The Lebesgue constant is crucial because it relates the interpolation error to the best possible polynomial approximation error. Let $p_n^*(x)$ be the best n-th degree polynomial approximation to $f(x)$ in the infinity norm. Then, the error of interpolation $f - I_n(f)$ can be bounded by:

$$||f - I_n(f)||_\infty \le ||f - p_n^*||_\infty + ||p_n^* - I_n(f)||_\infty$$

Since $I_n(p_n^*) = p_n^*$ (interpolating a polynomial of degree n gives itself), we have $p_n^* - I_n(f) = I_n(p_n^*) - I_n(f) = I_n(p_n^* - f)$. Therefore,

$$||p_n^* - I_n(f)||_\infty = ||I_n(p_n^* - f)||_\infty \le \Lambda_n(X)||p_n^* - f||_\infty = \Lambda_n(X)||f - p_n^*||_\infty$$

Substituting this back into the error bound:

$$||f - I_n(f)||_\infty \le ||f - p_n^*||_\infty + \Lambda_n(X)||f - p_n^*||_\infty = (1 + \Lambda_n(X))||f - p_n^*||_\infty$$

This inequality shows that the interpolation error is at most $(1+\Lambda_n(X))$ times the error of the best polynomial approximation. If $\Lambda_n(X)$ is small, then the interpolation error is not much larger than the best possible error.

6.1 Wirestrass Approximation Theorem and Bernstein Polynomials

The Wirestrass Approximation Theorem states that any continuous function on a closed interval can be uniformly approximated by polynomials to arbitrary accuracy. Formally, for any continuous function f on $[a, b]$ and any $\epsilon > 0$, there exists a polynomial p_n^* of some degree n such that $||f - p_n^*||_\infty < \epsilon$. This theorem guarantees that the best approximation error $||f - p_n^*||_\infty$ can be made arbitrarily small as $n \to \infty$.

Bernstein polynomials provide a constructive proof of the Wirestrass Approximation Theorem. For a function f defined on $[0, 1]$, the Bernstein polynomial of degree n is given by:

$$B_n(f, x) = \sum_{m=0}^{n} f\left(\frac{m}{n}\right)\binom{n}{m}x^m(1-x)^{n-m}$$

As $n \to \infty$, $B_n(f, x)$ converges uniformly to $f(x)$ for any continuous function f on $[0, 1]$. While Bernstein polynomials guarantee convergence, they typically converge very slowly and are not practically used for interpolation. However, they are theoretically important for proving the Wirestrass Approximation Theorem.

6.2 Calculating Lebesgue Constant and `lsum.py`

The Python code `lsum.py` (Listing 2) calculates the sum of the absolute values of Lagrange basis polynomials, $\sum_{k=0}^{n} |L_k(x)|$, which is used to evaluate the Lebesgue constant. By finding the maximum value of this sum over the interval $[1, 1]$, we can approximate the Lebesgue constant for a given set of interpolation points.

Listing 2: Listing 2: `lsum.py`

```
#!/usr/bin/python

from math import *
import matplotlib.pyplot as plt
import numpy as np

# Function to calculate the sum of absolute values of Lagrange polynomials
def lsum(xp,x):
    ls=0.
    for k in range(xp.size):
        xc=xp[k]
        L=1.
        for l in range(xp.size):
            if l != k:
                L=L*(x-xp[l])/(xp[k]-xp[l])
        ls=ls+abs(L)
    return ls


# Control points (either linearly spaced, or Chebyshev)
n=16
xp=np.linspace(-1.,1.,n) # (Linearly spaced)
#xp=np.array([cos((2*j-1)*pi/(2*n)) for j in range(n)]) # (Chebyshev points)
yp=np.array([0. for q in xp])

# Sample points
xx=np.linspace(-1.,1.,500)
yy=np.array([lsum(xp,q) for q in xx])


# Plot figure using Matplotlib
plt.figure(1)
plt.plot(xx,yy,'-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

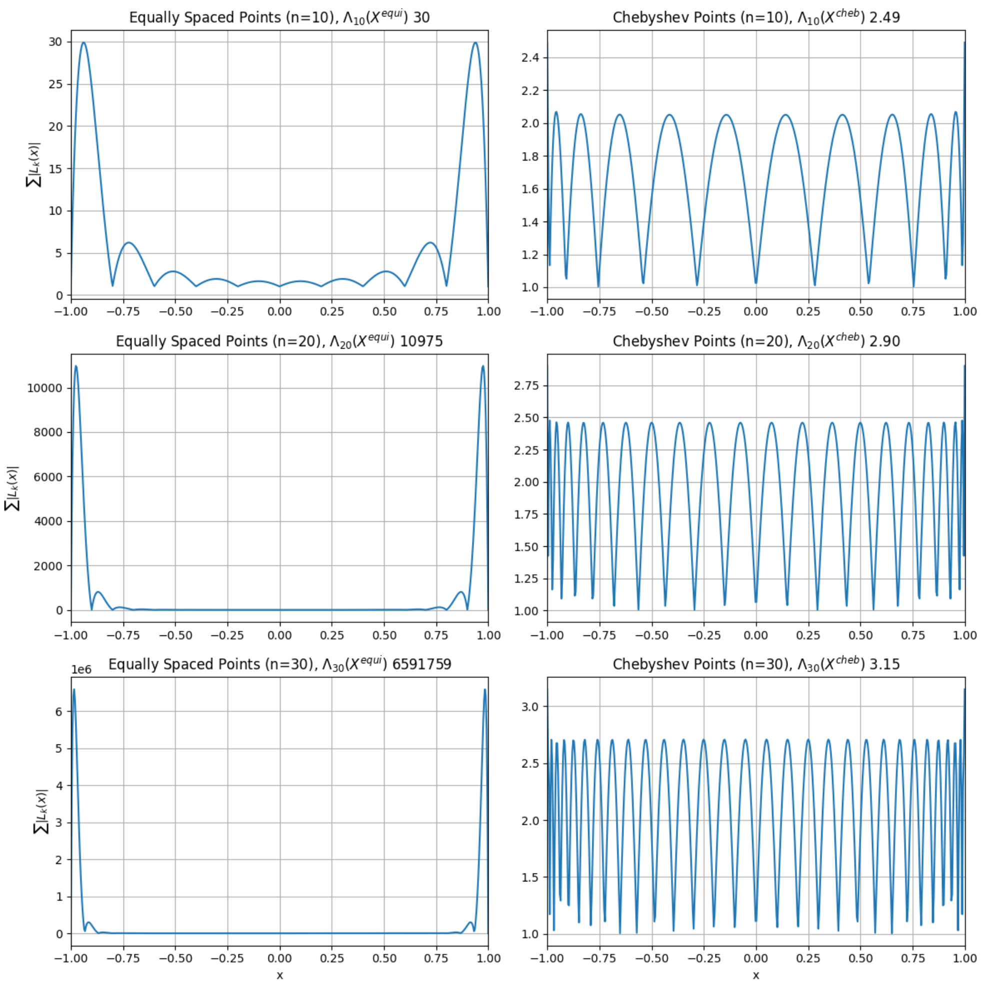

Figure 4: Comparison of $\sum_{k=0}^{n} |L_k(x)|$ for equally spaced points and Chebyshev points for n=10, 20, and 30 (number of interpolation points is n+1). The Lebesgue constant $\Lambda_n(X)$ is approximately the maximum value of these plots.

Figure 4 compares the plots of $\sum_{k=0}^{n} |L_k(x)|$ for equally spaced and Chebyshev points for $n = 10, 20$, and 30. For equally spaced points, the sum of Lagrange basis polynomials grows dramatically at the endpoints as n increases, leading to large Lebesgue constants. For Chebyshev points, the sum remains much smaller and more uniformly bounded, resulting in significantly smaller Lebesgue constants.

The approximate values of the Lebesgue constants from Figure 4 are:

- For equally spaced points (X^{equi}):
  - $\Lambda_{10}(X^{equi}) \approx 29.9$
  - $\Lambda_{20}(X^{equi}) \approx 10,987$
  - $\Lambda_{30}(X^{equi}) \approx 6,600,000$
- For Chebyshev points (X^{cheb}):
  - $\Lambda_{10}(X^{cheb}) \approx 2.49$
  - $\Lambda_{20}(X^{cheb}) \approx 2.9$
  - $\Lambda_{30}(X^{cheb}) \approx 3.15$

The Lebesgue constant for equally spaced points grows explosively with n, indicating the instability and divergence associated with Runge's phenomenon. In contrast, the Lebesgue constant for Chebyshev points grows much more slowly.

7 Scaling of Lebesgue Constant

The asymptotic scaling of the Lebesgue constant as $n \to \infty$ differs significantly for equally spaced and Chebyshev points:

- **Equally Spaced Points:** $\Lambda_n(X^{equi}) \sim O\left(\frac{2^n}{en \log n}\right)$. This exponential growth confirms the rapid deterioration of interpolation quality with equally spaced points as n increases.
- **Chebyshev Points:** $\Lambda_n(X^{cheb}) \sim O(\log n)$. The logarithmic growth is much more favorable, ensuring that the Lebesgue constant remains reasonably small even for large n.

While Chebyshev points provide a favorable scaling of the Lebesgue constant and greatly improve interpolation compared to equally spaced points, they are not theoretically optimal in minimizing the Lebesgue constant. However, they represent a very effective and practical choice for polynomial interpolation, offering a good balance between computational simplicity and approximation accuracy.

8 Summary

In summary, we have considered two key aspects of polynomial interpolation:

1. **Polynomial Interpolation for Fitting Discrete Data:**
   - We can achieve "zero error" at the interpolation points regardless of their choice. We are guaranteed to fit the discrete data exactly.
   - Lagrange polynomial basis is a well-conditioned and effective method for this task.
2. **Polynomial Interpolation for Approximating Continuous Functions:**
   - We use the same methodology as for discrete data to construct the interpolant.
   - The choice of interpolation points becomes crucial in determining the magnitude of the approximation error $||f - I_n(f)||_\infty$.
   - Chebyshev points are a superior choice compared to equally spaced points for function approximation, as they control oscillations and provide better convergence properties, reflected in the slower growth of their Lebesgue constant.

Choosing appropriate interpolation points, such as Chebyshev points, is essential for achieving accurate and stable polynomial approximations of continuous functions. The Lebesgue constant provides a valuable tool for analyzing and comparing the stability and potential error amplification of different interpolation schemes, independent of the function being approximated.