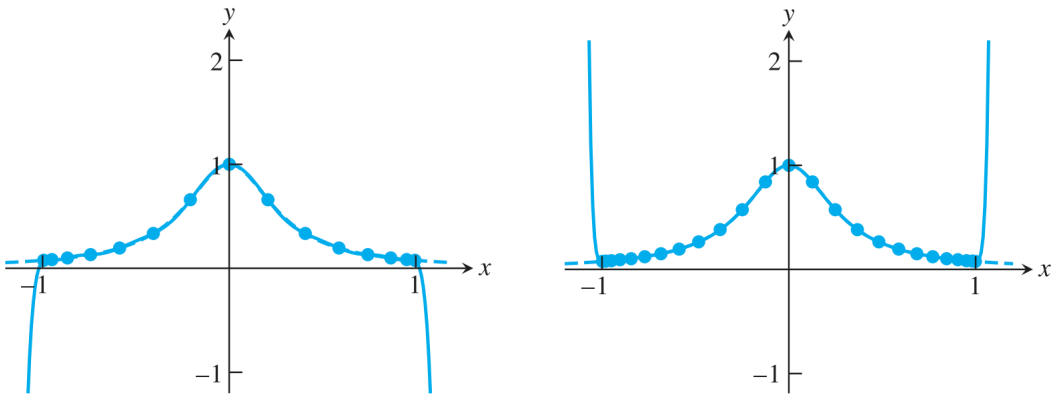
暨南大学本科实验报告专用纸

课程名称	数值计算实验			成绩评定	
实验项目名称	Computing Problems			指导老师	Liangda Fang
实验项目编号	02	实验项目类型	验证型	实验地点	
学生姓名		学号			
学院	国际学院	系		专业	计算机科学与技术
实验时间	2023 年 10 月 23 日上午 10:30 ~ 12:10				

I. Problem

Let $f(x) = e^{-2x}$ and the interval to be $[-1, 1]$.

1. Write a program generating the Newton's divided difference formula;
2. Use the program to generate a degree n polynomial with evenly spaced points and Chebyshev points for $n = 10$, 20 and 40;
3. Plot the polynomials for the above types (see Figure 3.8);



(a) (b)

Figure 3.8 Interpolation of Runge example with Chebyshev nodes. The Runge function $f(x)=1/(1+12x^2)$ is graphed along with its Chebyshev interpolation polynomial for (a) 15 points (b) 25 points. The error on [–1, 1] is negligible at this resolution. The polynomial wiggle of Figure 3.6 has vanished, at least between –1 and 1.

4. By sampling at a 0.05 step, create the empirical interpolation errors for each type, and plot a comparison (see Figure 3.11).

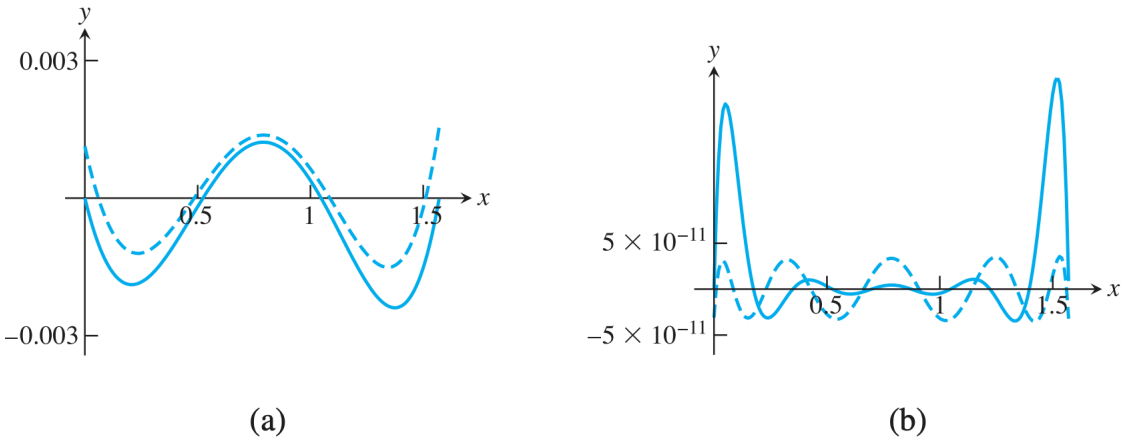


Figure 3.11 Interpolation error for approximating $f(x) = \sin x$. (a) Interpolation error for degree 3 interpolating polynomial with evenly spaced base points (solid curve) and Chebyshev base points (dashed curve). (b) Same as (a), but degree 9.

II. Algorithm Summary

Data compression through polynomial approximation is a useful technique. When dealing with a set of data points, accurately describing the underlying functions can be challenging. However, the value of the function y corresponding to a given x can be easily and nearly accurately calculated using polynomial interpolation. In the realm of polynomial interpolation, two common methods are the Newton Difference Formula and the use of Chebyshev nodes. Below, we will delve into the principles behind these two algorithms.

1. Main Theorem of Polynomial Interpolation

The fundamental theorem of polynomial interpolation is a cornerstone for this technique. It states that if you have n points (x_1, y_1), ..., (x_n, y_n) in a plane with distinct x_i, there exists one and only one polynomial P of degree $n-1$ or less, which satisfies $P(x_i) = y_i$ for $i = 1, ..., n$. Newton's divided differences offer a straightforward way to express the interpolating polynomial, ensuring that the resulting polynomial will be of a degree at most $n-1$. In essence, this means that the interpolation polynomials obtained through Newton's difference quotient formula are unique and align with those from other methods.

2. Newton's Divided Difference Formula

To provide a clearer understanding, we need to clarify some definitions. Let's assume the data points are derived from a function $f(x)$, and our goal is to create a polynomial that interpolates these points $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$. In this method, the coefficient of the x^{n-1} term in the unique polynomial is $f[x_1, \ldots, x_n]$.

With this definition in mind, the following formula for the interpolating polynomial holds, known as Newton' s divided difference formula:

$$P(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) + \ldots + f[x_1, \ldots, x_n](x - x_1) \ldots (x - x_{n-1})$$

Additionally, the coefficients $f[x_1, \ldots, x_k]$ from the above definition can be calculated recursively, as follows:

The computation of coefficients:

1. List the data points in a table:

$$\begin{array}{c|c} x_1 & f(x_1) \\ x_2 & f(x_2) \\ \vdots & \vdots \\ x_n & f(x_n) \end{array}$$

2. Define the divided differences, which are real numbers:

$$\begin{aligned} f[x_k] &= f(x_k) \\ f[x_k\ x_{k+1}] &= \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} \\ f[x_k\ x_{k+1}\ x_{k+2}] &= \frac{f[x_{k+1}\ x_{k+2}] - f[x_k\ x_{k+1}]}{x_{k+2} - x_k} \\ f[x_k\ x_{k+1}\ x_{k+2}\ x_{k+3}] &= \frac{f[x_{k+1}\ x_{k+2}\ x_{k+3}] - f[x_k\ x_{k+1}\ x_{k+2}]}{x_{k+3} - x_k} \end{aligned}$$

And so on. It's important to understand:

1. The polynomial obtained from the Newton difference formula interpolates the data points $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$.
2. The coefficients of the unique polynomial, denoted as $f[x_1, \ldots, x_n]$, can be calculated using analogous steps.

It's worth noting that the divided difference formula yields an interpolating polynomial structured as a nested polynomial. This format enables efficient evaluation.
After simplification, the interpolation polynomials calculated using Newton's difference formula are as follows:

$$P(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) + \ldots + f[x_1, \ldots, x_n](x - x_1) \ldots (x - x_{n-1})$$

The calculation process for Newton's difference formula is as follows:

Algorithm 1 Newton's Divided Difference Algorithm

procedure NEWTONDIVIDEDDIFFERENCE(x, y)
  ▷ Given arrays x and y containing data points
  for $j \leftarrow 1$ to n **do**
    $f[x_j] \leftarrow y_j$
  end for
  for $i \leftarrow 2$ to n **do**
    for $j \leftarrow 1$ to $n + 1 - i$ **do**
      $f[x_j, \ldots, x_{j+i-1}] \leftarrow \frac{f[x_{j+1} \ldots x_{j+i-1}] - f[x_j \ldots x_{j+i-2}]}{x_{j+i-1} - x_j}$
    end for
  end for
  return f
  ▷ Return the computed divided differences
end procedure

3. Interpolation Error

The interpolation error, denoted as $f(x) - P(x)$, represents the difference between the original function that provided the data points and the interpolating polynomial $P(x)$ when evaluated at x. The following theorem provides the interpolation error formula and establishes the error bounds for interpolation polynomials.

Theorem:

Assume that $P(x)$ is the interpolating polynomial (degree $n-1$ or less) that fits the n points $(x_1, y_1), \ldots, (x_n, y_n)$. The interpolation error is given by:

$$f(x) - P(x) = \frac{f^{(n)}(c)}{n!}(x - x_1)(x - x_2)\ldots(x - x_n)$$

where c lies between the smallest and largest of the numbers $x, x_1, \ldots, x_n$.

Runge Phenomenon:

The statement that increasing the number of sample points or using a higher-order interpolation polynomial necessarily results in smaller interpolation errors is not accurate. This is actually contradicted by the phenomenon known as the Runge phenomenon.

The Runge phenomenon occurs when interpolating a function with a high-degree polynomial using equidistant sample points. It leads to oscillations and large errors in the interpolation, especially towards the edges of the interval. To illustrate this, let's consider a specific function:

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

Now, if we use 5, 10, 15 and 20 equidistant sample points within this interval and apply Newton's divided difference method to construct interpolation polynomials, we can visualize the comparison between the interpolation polynomial curve and the original function curve.

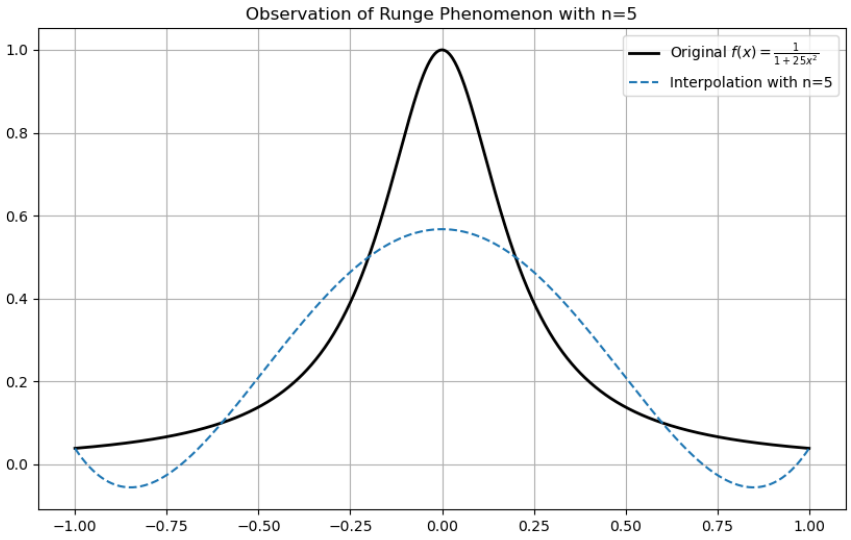


Figure 1: Runge Phenomenon with n=5

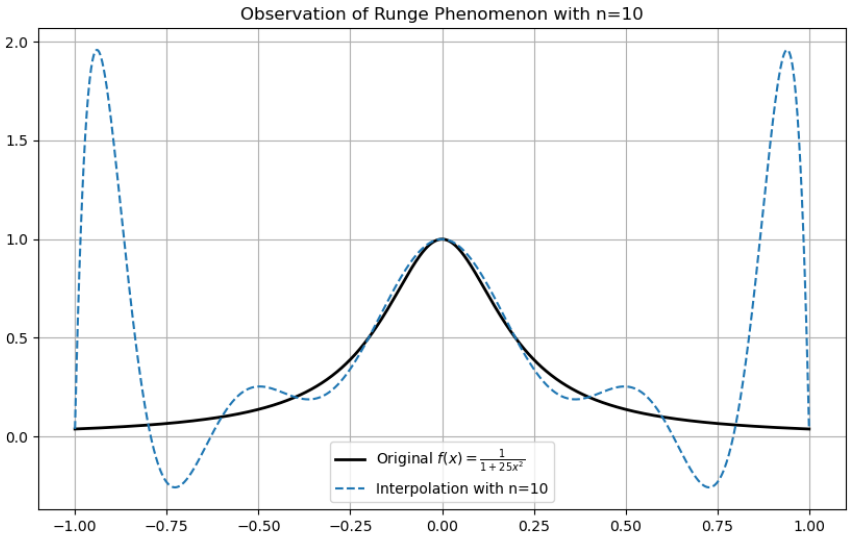


Figure 2: Runge Phenomenon with n=10

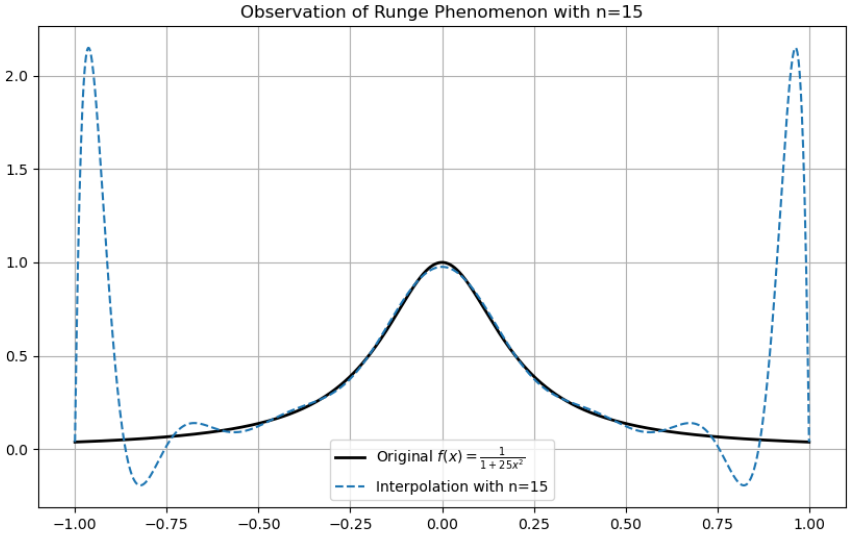


Figure 3: Runge Phenomenon with n=15

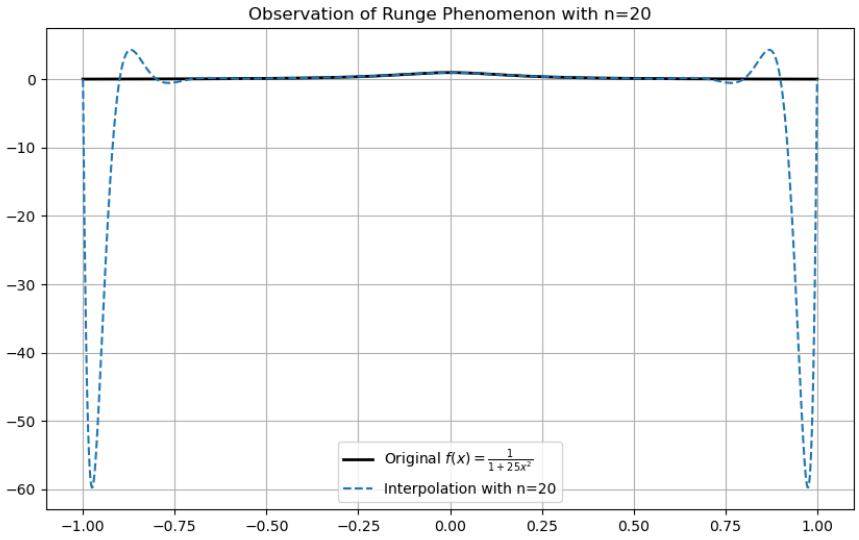


Figure 4: Runge Phenomenon with n=20

In the case of the Runge phenomenon, as we increase the number of sample points or the degree of the interpolation polynomial, we may actually observe larger oscillations and interpolation errors, especially at the edges of the interval. This happens due to the characteristics of equidistant sample points and the behavior of high-degree polynomials.

In summary, the Runge phenomenon highlights that simply adding more sample points or using higher-order interpolation polynomials doesn't guarantee smaller interpolation errors and can, in fact, lead to larger errors and oscillations in certain cases, especially when equidistant points are used. Careful consideration of interpolation methods and the choice of sampling points is essential to achieve accurate results.

4. Chebyshev Interpolation

While it's common to use evenly distributed points as the base points X for interpolation, it has been proven that the spacing of these base points can significantly impact the interpolation error. Chebyshev interpolation offers an optimal approach for selecting the spacing between points. The motive of Chebyshev interpolation is to improve control over the maximum value of the interpolation error within the interpolation interval.

If we fix the interval to be $[-1, 1]$, we aim to choose real numbers $-1 \leq x_1, \ldots, x_n \leq 1$ that minimize the value of E as much as possible. In fact:

$$E = \frac{1}{2^{n-1}}$$

This minimum is achieved by the following formula (where $T_n(x)$ denotes the Chebyshev polynomial):

$$x_k = \cos\left(\frac{(2k-1)\pi}{2n}\right)$$

In general, the minimum value of E occurs when the x_i values are selected as Chebyshev nodes.

III. Experimental Procedures

Step1: Define a variable n.
Step2: Define a function, `nest`, to evaluate a polynomial in its nested form using Horner's Method.
Step3: Define a function, `newton_divided_difference`, to compute the coefficients of an interpolating polynomial using Newton's Divided Difference Formula.
Step4: Utilize the `nest` and `newton_divided_difference` functions to generate polynomials of degree n using evenly spaced points and Chebyshev points for $n = 10, 20$, and 40.
Step5: Plot the generated polynomials for the specified types.
Step6: Sample at a 0.05 step to create empirical interpolation errors for each type and plot a comparison.

IV. Result Analysis

1. Interpolation Results and Plot

When n = 10, the result of step 5 is as follows:

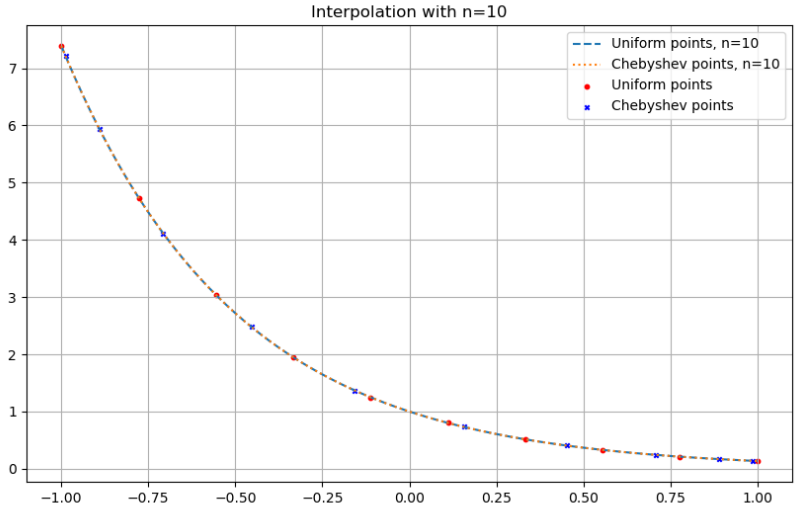


Figure 5: Interpolation with n=10

When n = 20, the result of step 5 is as follows:

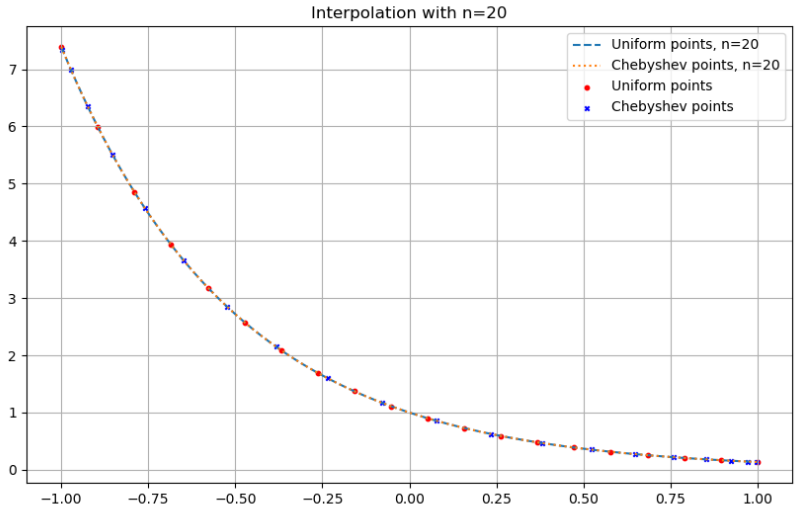


Figure 6: Interpolation with n=20

When n = 40, the result of step 5 is as follows:

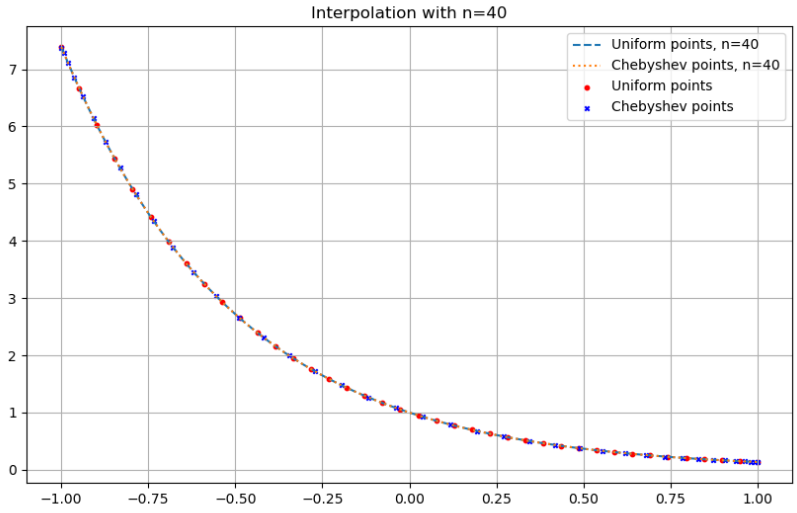


Figure 7: Interpolation with n=40

We can observe that for the function $f(x) = e^{-2x}$, the Runge phenomenon does not manifest itself. As the number of sample points increases, the error in comparison to the original function becomes negligible. In other words, there is no occurrence of the Runge phenomenon.

2. Interpolation Errors and Plot

When n = 10, the result of step 6 is as follows:

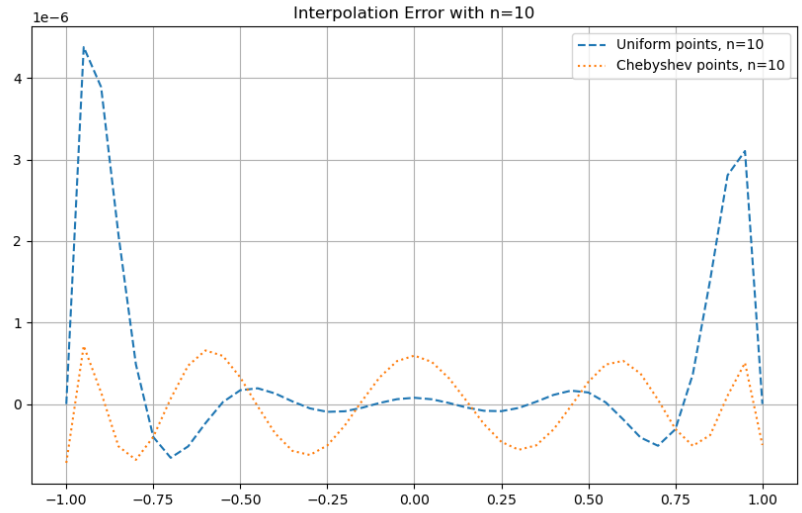


Figure 8: Interpolation errors with n=10

When n = 20, the result of step 6 is as follows:

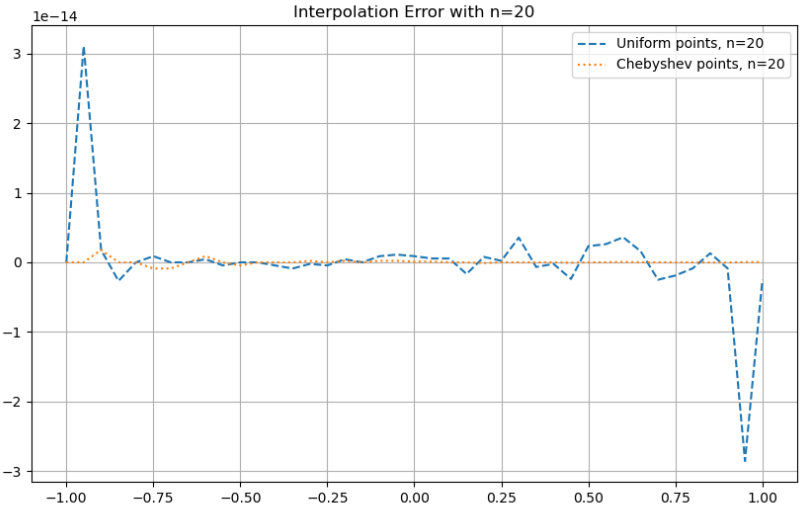


Figure 9: Interpolation errors with n=20

When n = 40, the result of step 6 is as follows:

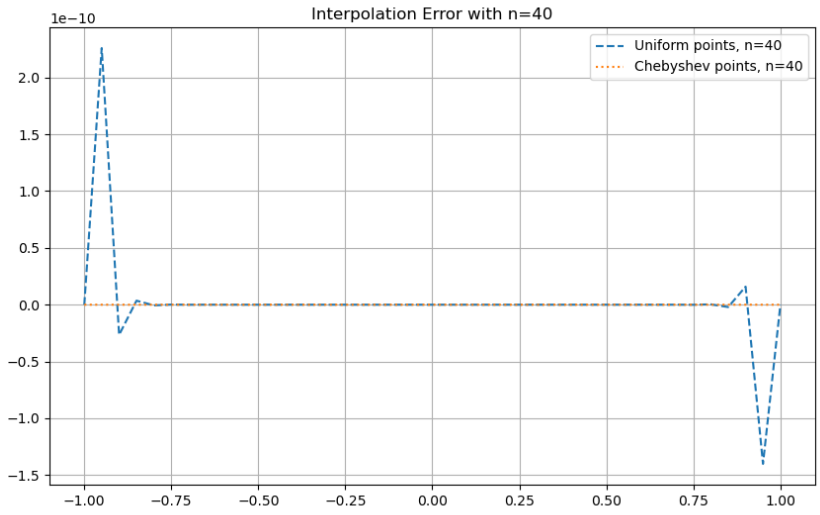


Figure 10: Interpolation errors with n=40

It is evident that the interpolation error using evenly spaced sample points increases significantly at the edges of the interval. In contrast, when employing Chebyshev points for interpolation, the error fluctuation is less pronounced, and the interpolation error is smaller compared to evenly spaced sample points. Notably, when we transition from n being 20 to 40, the maximum error after interpolation using evenly spaced sample points increases by a factor of 10^5.

V. Experimental Summary

In this experiment, we employed the Newton' s divided difference method to obtain the interpolation polynomial $P_n(x)$. We utilized both evenly spaced sample points and Chebyshev points, aiming to compare their effects on the interpolation error.

It's important to note that using Chebyshev points as the sample points yields the best results, ensuring minimal interpolation error. For some functions, employing evenly spaced sample points can lead to the Runge phenomenon, resulting in significant interpolation errors at both ends of the interval. However, in this particular experiment, even when we used evenly spaced sample points for interpolation, the Runge phenomenon did not manifest in $f(x) = e^{-2x}$.

Furthermore, it's worth highlighting that increasing the number of sample points does not necessarily lead to better interpolation performance, as demonstrated by the transition from 20 to 40 sample points.

VI. Appendix: Source Code

1. methods.py

```python
import numpy as np

def nest(coef, x, x_data=None):
    """
    Evaluate Newton's polynomial using Horner's method.
    """
    n = len(coef) - 1
    p = coef[n]
    for i in range(n - 1, -1, -1):
        p = coef[i] + (x - x_data[i]) * p
    return p

def newton_divided_difference(x_data, y_data):
    """
    Generate the Newton interpolating polynomial.
    """
    n = len(x_data)
    F = np.zeros((n, n))
    F[:, 0] = y_data

    for j in range(1, n):
        for i in range(n - j):
            F[i][j] = (F[i + 1][j - 1] - F[i][j - 1]) / (x_data[i + j] -
                x_data[i])

    coef = F[0]

    def polynomial(x):
        return nest(coef, x, x_data)

    return polynomial

def uniform_points(n, a=-1, b=1):
    """
    Generate n uniform points in the interval [a, b].
    """
    return np.linspace(a, b, n)


def chebyshev_points(n, a=-1, b=1):
    """
    Generate n Chebyshev points in the interval [a, b].
    """
    i = np.arange(1, n + 1)
    x_cheb = np.cos((2 * i - 1) * np.pi / (2 * n))
    return 0.5 * (a + b) + 0.5 * (b - a) * x_cheb


def func(x):
    return np.exp(-2 * x)
```

2. Runge-phenomenon.py

```python
from lab2_methods import *
import matplotlib.pyplot as plt


def runge_function(x):
    return 1 / (1 + 25 * x ** 2)


# Generate uniform points and compute interpolating polynomial for various degrees
x_values = np.linspace(-1, 1, 1000)
y_runge = runge_function(x_values)

# Adjusting the range for degrees
degrees = [5, 10, 15, 20]

# Plotting individual graphs for each degree along with the original function
for degree in degrees:
    plt.figure(figsize=(10, 6))

    # Plot original Runge function
    plt.plot(x_values, y_runge, label="Original $f(x) = \\frac{1}{1+25x^2}$", color="black", linewidth=2)

    # Plot interpolating polynomial for the specific degree
    x_data = uniform_points(degree + 1)
    y_data = runge_function(x_data)
    polynomial = newton_divided_difference(x_data, y_data)
    y_poly = [polynomial(x) for x in x_values]
    plt.plot(x_values, y_poly, '--', label=f"Interpolation with n={degree}")

    plt.title(f"Observation of Runge Phenomenon with n={degree}")
    plt.legend()
    plt.grid(True)
    plt.show()
```

3. fitting.py

```python
from lab2_methods import *
import matplotlib.pyplot as plt

# Generate points for n=10, 20, 40
n_values = [10, 20, 40]
uniform_pts = {n: uniform_points(n) for n in n_values}
chebyshev_pts = {n: chebyshev_points(n) for n in n_values}

# Compute function values for the generated points
uniform_y = {n: func(uniform_pts[n]) for n in n_values}
chebyshev_y = {n: func(chebyshev_pts[n]) for n in n_values}

# Generate Newton interpolating polynomials
uniform_polys = {n: newton_divided_difference(uniform_pts[n], uniform_y[n])
    for n in n_values}
chebyshev_polys = {n: newton_divided_difference(chebyshev_pts[n], chebyshev_y
    [n]) for n in n_values}

x_plot = np.linspace(-1, 1, 500)
y_original = func(x_plot)

for n in n_values:
    plt.figure(figsize=(10, 6))

    # Plot original function
    plt.plot(x_plot, [uniform_polys[n](x) for x in x_plot], '--', label=f"
        Uniform points, n={n}")
    plt.plot(x_plot, [chebyshev_polys[n](x) for x in x_plot], ':', label=f"
        Chebyshev points, n={n}")

    # Plot data points
    plt.scatter(uniform_pts[n], [uniform_polys[n](x) for x in uniform_pts[n
        ]], color='red', s=10, marker='o',
                label="Uniform points")
    plt.scatter(chebyshev_pts[n], [chebyshev_polys[n](x) for x in
        chebyshev_pts[n]], color='blue', s=10, marker='x',
                label="Chebyshev points")

    plt.title(f"Interpolation with n={n}")
    plt.legend()
    plt.grid(True)
    plt.show()
```

4. error-comparison.py

```python
from lab2_methods import *
import matplotlib.pyplot as plt

# Generate points for n=10, 20, 40
n_values = [10, 20, 40]
uniform_pts = {n: uniform_points(n) for n in n_values}
chebyshev_pts = {n: chebyshev_points(n) for n in n_values}

# Compute function values for the generated points
uniform_y = {n: func(uniform_pts[n]) for n in n_values}
chebyshev_y = {n: func(chebyshev_pts[n]) for n in n_values}

# Generate Newton interpolating polynomials
uniform_polys = {n: newton_divided_difference(uniform_pts[n], uniform_y[n])
    for n in n_values}
chebyshev_polys = {n: newton_divided_difference(chebyshev_pts[n], chebyshev_y
    [n]) for n in n_values}

# Sampling points and computing the interpolation errors
x_sample = np.arange(-1, 1.05, 0.05)
y_sample_original = func(x_sample)

uniform_errors = {n: [uniform_polys[n](x) - func(x) for x in x_sample] for n
    in n_values}
chebyshev_errors = {n: [chebyshev_polys[n](x) - func(x) for x in x_sample]
    for n in n_values}

for n in n_values:
    plt.figure(figsize=(10, 6))

    plt.plot(x_sample, uniform_errors[n], '--', label=f"Uniform␣points,␣n={n}
        ")
    plt.plot(x_sample, chebyshev_errors[n], ':', label=f"Chebyshev␣points,␣n
        ={n}")
    plt.title(f"Interpolation␣Error␣with␣n={n}")
    plt.legend()
    plt.grid(True)
    plt.show()
```