Programming Assignment 5

Jenny Petrova

February 23rd, 2025

1 Executive Summary

This report demonstrates several methods of polynomial interpolation. For a given set of $n+1$ distinct nodes (x_0, y_0), ..., (x_n, y_n), we want to construct a degree-n polynomial $p_n(x) \in \mathbb{P}_n$ such that

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n,$$

where $p_n(x_0) = y_0$, ..., $p_n(x_n) = y_n$.

We construct routines to perform interpolation by the Barycentric Form 1 and Barycentric Form 2 of the Lagrange interpolating polynomial, and calculate the Newton divided differences, using the adapted Horner's rule to evaluate the Newton interpolating polynomial. The stability, accuracy, and computational complexity of each method is assessed.

2 Statement of the Problem

Theorem 1. *Given $n+1$ distinct points (x_i, y_i), there exists a unique interpolating polynomial $p_n(x) \in \mathbb{P}_n$ such that $p_n(x_i) = y_i$ for all $0 \leq i \leq n$.*

There exists a unique polynomial that interpolates a given set of distinct nodes, hence the solution to our interpolation problem should, theoretically, remain constant regardless of the method we apply. This theorem allows us to compare the efficiency and accuracy between different methods of polynomial interpolation. As each algorithm varies in computational complexity and may introduce different degrees of floating point error throughout the polynomial construction process, we must assess the level of interpolation error between the true and interpolated polynomial. To do this, we test problems for which we know the answer. We generate a set of values $x_0, \ldots, x_n$, using known functions $f(x)$ to obtain the corresponding values $y_0, \ldots, y_n$, where $f(x_i) = y_i$. Our goal is to minimize the interpolation error. We attempt to solve this problem by varying our initial choice of mesh points. We develop a routine to generate the initial mesh points, which can be either a set of uniform points from a predetermined interval $[a, b]$, Chebyshev points of the first kind, or Chebyshev points of the second kind. Additionally, a routine to order the distinct mesh points is constructed, such that we can order the set of x_i values in descreasing order, increasing order, or by Leja ordering. The choice of ordering and distinct mesh points can help us properly condition the problem such that our resulting polynomial interpolates any new points x_j to a high degree of accuracy. We assess the conditioning, stability, and accuracy of interpolating polynomials of varying degrees for various functions, for varying mesh points and intervals.

3 Description of the Mathematics

3.1 Lagrange Interpolating Polynomial

Lagrange Interpolation consists of computing the Lagrange basis functions $l_0(x), \ldots, l_n(x)$ given by

$$l_i(x) = \frac{\prod_{j=0}^{n} \quad (x - x_j)}{\prod_{j=0, j \neq i}^{n} (x_i - x_j)}, \tag{1}$$

where $l_i(x)$ is a degree n polynomial and

$$l_i(x_j) = \begin{cases} 0 & \text{if} \quad i \neq j \\ 1 & \text{if} \quad i = j. \end{cases} \tag{2}$$

The Lagrange Form of the unique interpolating polynomial is constructed by

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x), \tag{3}$$

where $y_i = f(x_i)$. Now consider the constant function $f(x) = 1$, which is the unique interpolating polynomial for points $(x_0, 1), \ldots, (x_n, 1)$. Equation (3) gives

$$p_n(x) = \sum_{i=0}^{n} (1) l_i(x) = f(x).$$

Since $p_n(x) = f(x) = 1$, the above equation simplifies to

$$\sum_{i=0}^{n} l_i(x) = 1. \tag{4}$$

This result is necessary for the derivations that follow.

3.1.1 Barycentric Form 1

From the Lagrange interpolating polynomial (1), define the following:

$$w_{n+1}(x) = \prod_{j=0}^{n} (x - x_j) \tag{5}$$

and

$$w'_{n+1}(x_i) = \prod_{j=0, j\neq i}^{n} (x_i - x_j). \tag{6}$$

Equation (1) becomes

$$l_i(x) = \frac{w_{n+1}(x)}{(x - x_j)w'_{n+1}(x_i)} \tag{7}$$

and the **Barycentric Form 1** interpolation formula is given by

$$p_n(x) = w_{n+1}(x) \sum_{i=0}^{n} \frac{y_i}{(x - x_j)w'_{n+1}(x_i)} \tag{8}$$

$$= w_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\gamma_i}{(x - x_j)}, \tag{9}$$

where

$$\gamma_i^{-1} = w'_{n+1}(x_i). \tag{10}$$

3.1.2 Barycentric Form 2

By result (4) and equation (8),

$$\sum_{i=0}^{n} l_i(x) = w_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\gamma_i}{(x - x_j)} = 1. \tag{11}$$

Then $w_{n+1}(x)$ can be written as

$$w_{n+1}(x) = \left[\sum_{i=0}^{n} \frac{\gamma_i}{(x - x_j)} \right]^{-1}. \tag{12}$$

The true form of the **Barycentric Form 2** interpolation formula is thus defined as

$$p_n(x) = \frac{\sum_{i=0}^{n} y_i \frac{\gamma_i}{(x - x_j)}}{\sum_{i=0}^{n} \frac{\gamma_i}{(x - x_j)}}. \tag{13}$$

3.1.3 Uniform Nodes

Suppose we take an interval $[a, b]$ of equally spaces nodes $x_i = a + ih$ for $h = (b - a)/n$. Our equation for γ_i^{-1} (10) can be expanded such that

$$\begin{aligned}
\gamma_i^{-1} &= \prod_{j=0}^{i-1}(x_i - x_j) \prod_{j=i+1}^{n}(x_i - x_j) \\
&= \prod_{j=0}^{i-1}(a + ih - (a + jh)) \prod_{j=i+1}^{n}(a + ih - (a + jh)) \\
&= \prod_{j=0}^{i-1}(i - j)h \prod_{j=i+1}^{n}(i - j)h.
\end{aligned}$$

Using factorial notation,

$$\prod_{j=0}^{i-1}(i - j) = i!,$$

and

$$\prod_{j=i+1}^{n}(j - i) = (n - i)!,$$

we obtain

$$\gamma^{-1} = (-1)^{n-i} h^n (i!)(n - i)!. \tag{14}$$

Then

$$\begin{aligned}
\gamma_i &= \frac{(-1)^{n-i}}{h^n (i!)(n - i)!} \\
&= \frac{(-1)^{n-i}}{h^n n!}\binom{n}{i} \\
&= \frac{(-1)^n}{h^n n!}\left[(-1)^i \binom{n}{i}\right] \\
&= \frac{(-1)^n}{h^n n!}\beta_i
\end{aligned}$$

such that

$$\beta_{i+1} = -\beta_i \frac{n - i}{i + 1}. \tag{15}$$

Therefore the Barycentric Form 1 interpolation formula (8) can be adapted such that

$$\begin{aligned}
p_n(x) &= w_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\gamma_i}{(x - x_j)} \\
&= \tilde{w}_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\beta_i}{(x - x_j)}.
\end{aligned}$$

3.1.4 Chebyshev Nodes

Chebyshev polynomials of the first kind (T_n) are defined by

$$T_n(\cos\theta) = \cos(n\theta).$$

Chebyshev nodes of the first kind are given by the zeros of polynomial T_n, where

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n+2}\right), \quad 0 \le i \le n. \tag{16}$$

Therefore the coefficients (β_i) for the Barycentric form interpolating polynomial (16), for these nodes, are

$$\beta_i = (-1)^i \sin\left(\frac{(2i+1)\pi}{2n+1}\right). \tag{17}$$

Similarly, Chebyshev polynomials of the second kind (U_n) are defined by

$$U_n(\cos\theta)\sin\theta = sin((n+1)\theta).$$

Chebyshev nodes of the second kind are given by the zeros of polynomial U_n, where

$$x_i = \cos\left(\frac{i\pi}{n}\right), \quad 0 \le i \le n. \tag{18}$$

The coefficients (β_i) for the Barycentric form interpolating polynomial (16), for these nodes, are thus

$$\beta_i = (-1)^i \delta_i, \tag{19}$$

where

$$\delta_i = \begin{cases} \frac{1}{2} \text{ if } j = 0 \text{ and } j = n \\ 1 \text{ otherwise.} \end{cases}$$

3.2 Newton Interpolation

The **Newton form** of the interpolating polynomial is a linear combination of the Newton basis polynomials $w_i(x)$, such that

$$p_n(x) = \sum_{i=0}^{n} \alpha_i w_i(x), \tag{20}$$

given

$$w_0(x) = 1, \quad w_i(x) = \prod_{j=0}^{i-1}(x - x_j).$$

The coefficients α_i are the divided differences of the data points (x_i, y_i), $0 \le i \le n$.

3.2.1 Divided Differences

The formula for the **Newton divided differences** is given by

$$f\left[x_0, \ldots, x_n\right] = \frac{y\left[x_1, \ldots, x_n\right] - y\left[x_0, \ldots, x_{n-1}\right]}{x_n - x_0}. \tag{21}$$

Consider a set of nodes $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$ with distinct values x_i. For the given nodes, the divided difference formula is used to construct the following table:

$$\begin{array}{cccc}
x_0 & x_1 & x_2 & x_3 \\
y[x_0] & y[x_1] & y[x_2] & y[x_3] \\
y[x_0, x_1] & y[x_1, x_2] & y[x_2, x_3] & \\
y[x_0, x_1, x_2] & y[x_1, x_2, x_3] & & \\
y[x_0, x_1, x_2, x_3] & & &
\end{array}, \tag{22}$$

where

$$\begin{aligned}
y[x_0] &= y_0, \\
y[x_0, x_1] &= \frac{y_1 - y_0}{x_1 - x_0}, \\
y[x_0, x_1, x_2] &= \frac{y[x_1, x_2] - y[x_0, x_1]}{x_2 - x_0}, \\
y[x_0, x_1, x_2, x_3] &= \frac{y[x_0, x_1, x_2] - y[x_1, x_2, x_3]}{x_3 - x_0}.
\end{aligned}$$

We construct the Newton interpolating polynomial using the recursive derivation of the Newton form. The above table yields

$$\begin{aligned}
p_n(x) =& y[x_0] + y[x_0, x_1](x - x_0) + y[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&+ y[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2).
\end{aligned}$$

The divided differences are the coefficients of the interpolating polynomial. Hence equation (13), the definition of the Newton interpolating polynomial, becomes

$$p_n(x) = \sum_{i=1}^{n} w_i(x) y[x_0, \ldots, x_n]. \tag{23}$$

3.3 Horner's Rule

Given the polynomial $p_n(x) = \sum_{i=0}^{n} a_n x^n$, **Horner's Rule** is a method for evaluating a polynomial $p_n(x)$ in nested form, such that

$$p_n(x) = \alpha_0 + x(\alpha_1 + x(\alpha_2 + x(\alpha_3 + \cdots + x(\alpha_{n-1} + x\alpha_n) \ldots))). \tag{24}$$

3.4 Conditioning of Interpolation

Given distinct values $x_0, \ldots, x_n$, we aim to find a condition number κ that bounds the difference between the true polynomial $p_n(x)$ interpolating $y_0, \ldots, y_n$, and the polynomial $\tilde{p}_n(x)$ interpolating perturbed values $\tilde{y}_0, \ldots, \tilde{y}_n$. The infinity norm is used on $\mathbb{P}_n$ for the change in polynomials:

$$||p_n(x) - \tilde{p}_n(x)||_\infty = \max_{x\in[a,b]} |p_n(x) - \tilde{p}_n(x)|$$

and on $\mathbb{R}^{n+1}$ for the change in parameters y_i:

$$||y_i - \tilde{y}_i||_\infty = \max_{0\leq i\leq n} |y_i - \tilde{y}_i|.$$

Hence the condition number κ bounds

$$||p_n(x) - \tilde{p}_n(x)||_\infty \leq \kappa_n ||y_i - \tilde{y}_i||_\infty. \tag{25}$$

The Lebesgue constant, Λ_n, defined as

$$\Lambda_n = || \sum_{i=0}^{n} |l_i^{(n)}(x)| \, ||_\infty \tag{26}$$

can be used as a condition number with respect to the infinity norm, such that

$$\begin{aligned} ||p_n(x) - \tilde{p}_n(x)||_\infty &= \max_{0\leq i\leq n} |\sum_{i=0}^{n} (y_i - \tilde{y}_i) l_i^{(n)}(x)| \\ &\leq \left(\max_{x\in[a,b]} \sum_{i=0}^{n} |l_i(x)|\right)\left(\max_{0\leq i\leq n} |y_i - \tilde{y}_i|\right) \\ &= \Lambda_n ||y_i - \tilde{y}_i||_\infty. \end{aligned}$$

The relative form of the condition number is given by

$$\frac{||p_n(x) - \tilde{p}_n(x)||_\infty}{||p_n(x)||_\infty} \leq \Lambda^{(rel)} \frac{||y - \tilde{y}||_\infty}{||y||_\infty}, \tag{27}$$

where

$$\Lambda^{(rel)} = \frac{\Lambda_n ||y||_\infty}{||p_n(x)||_\infty}. \tag{28}$$

The initial selection of nodes $\{x_i\}_{i=0}^n$, affects the value of $\Lambda_n(X)$. For uniform (equally spaced) nodes:

$$\Lambda_n(X) \approx \frac{2^{n+1}}{e(n \log n)}. \tag{29}$$

For Chebyshev points:

$$\Lambda_n(X) \approx \frac{2}{\pi} \log n. \tag{30}$$

We also define the condition number for the value of the interpolating polynomial $p_n(x)$ evaluated at a particular node x, for a given set of nodes $\{x_i\}_{i=0}^n$, by

$$\kappa(x,n,y) = \frac{\sum_{i=0}^n |l_i(x)y_i|}{|p_n(x)|} = \frac{\sum_{i=0}^n |l_i(x)y_i|}{|\sum_{i=0}^n l_i(x)y_i|}, \tag{31}$$

Suppose we take the constant function $f(x) = 1$. Then the condition number becomes

$$\kappa(x,n,1) = \sum_{i=0}^n |l_i(x)|. \tag{32}$$

Hence

$$1 \leq \kappa(x,n,1) \leq \Lambda_n = \max_{x\in[a,b]} \kappa(x,n,1) \tag{33}$$

and we can also define

$$1 \leq \kappa(x,n,y) \leq H_n = \max_{x\in[a,b]} \kappa(x,n,y). \tag{34}$$

4 Description of the Methods, Algorithms and Implementation

For the construction and testing of the polynomial interpolation methods, we develop several routines using the Python programming language. We present the relevant pseudocode below, and discuss the complexity of the algorithms with respect to time and space. Note that each function may be modified such as to use 64-bit (double) precision or 32-bit (single) precision of the IEEE standard floating point forms.

4.1 Polynomial Functions $f(x)$ for Testing

The functions we are interested in evaluating are:

1. $f_1(x) = (x-2)^9$
2. $f_2(x) = \prod_{i=0}^d (x-i)$
3. $f_3(x) = l_n(x)$, where $l_n(x)$ is the Lagrange basis function.
4. $f_4(x) = \frac{1}{1+25x^2}$

The following routines regard the above functions. For the experiments described in Section 5, these functions accept an array of values x for testing and return an array of the corresponding $f(x)$ values, so we may compare the polynomial evaluation to the true function evaluation.

Algorithm 1 Evaluate $f_1(x)$

```
function $f_1(x, d = 9, \rho = 2)$
    $y \leftarrow (x - \rho)^d$
    return $y$
end function
```

Algorithm 2 Evaluate $f_2(x)$

```
function $f_2(x, d)$
    $y \leftarrow 1$
    for $i \leftarrow 0$ to $d$ do
        $y \leftarrow y \cdot (x - i)$
    end for
    return $y$
end function
```

Algorithm 3 Evaluate $f_3(x)$

```
function $f_3(x, x_i, n)$
    for $i \leftarrow 0$ to $n$ do
        for $j \leftarrow 0$ to $n$ do
            if $i \neq j$ then
                $l \leftarrow l \cdot (x - x_i[j])/(x_i[i] - x_i[j])$
            end if
        end for
    end for
    return $l$
end function
```

Algorithm 4 Evaluate $f_4(x)$

```
function $f_4(x)$
    $y \leftarrow 1/(1 + 25x^2)$
    return $y$
end function
```

4.2 Methods for Polynomial Interpolation

The following routine evaluates the coefficients $\gamma_i(x)$ (10) of the Barycentric form 1 for interpolating values x_i, $i = 0, \ldots, n$.

Algorithm 5 Compute Barycentric 1 Weights

```
1: procedure BARY1_WEIGHTS($x_i, y_i$)
2:     $n \leftarrow |x_i|$
3:     $\gamma \leftarrow [1, 1, \ldots, 1]$ (length $n$)
4:     for $i \leftarrow 0$ to $n - 1$ do
5:         prod $= 1$
6:         for $j \leftarrow 0$ to $n - 1$ do
7:             if $i \neq j$ then
8:                 prod $\leftarrow$ prod $\cdot (x_i[i] - x_i[j])$
9:             end if
10:            $\gamma = 1/$prod
11:        end for
12:    end for
13:    return $\gamma$
14: end procedure
```

The following routine evaluates the polynomial $p_n(x)$ in Barycentric form 1.

Algorithm 6 Barycentric Form 1 Interpolation

```
procedure BARY1_INTERPOLATION($x, x_i, \gamma, y_i$)
    $n \leftarrow |x_i|$
    $p \leftarrow$ array of zeros (length $x$)
    $w \leftarrow$ array of ones (length $x$)
    $\kappa_1 \leftarrow$ array of ones (length $x$)
    $\kappa_y \leftarrow$ array of ones (length $x$)
    for $k \leftarrow 0$ to $|x| - 1$ do
        $sum\_abs\_1 \leftarrow 0$
        $sum\_abs\_y \leftarrow 0$
        for $j \leftarrow 0$ to $n - 1$ do
            $w[k] \leftarrow w[k] \times (x[k] - x_i[j])$
        end for
        for $i \leftarrow 0$ to $n - 1$ do
            if is_close($x[k], x_i[i]$, atol $= 10^{-12}$) then
                $p[k] \leftarrow y_i[i]$
                break
            end if
            $p[k] \leftarrow p[k] + \dfrac{w[k] \cdot y_i[i] \cdot \gamma[i]}{x[k] - x_i[i]}$
            $sum\_abs\_1 \leftarrow sum\_abs\_1 + \left|\dfrac{w[k] \cdot \gamma[i]}{x[k] - x_i[i]}\right|$
            $sum\_abs\_y \leftarrow sum\_abs\_y + \dfrac{|w[k] \cdot y_i[i] \cdot \gamma[i]|}{x[k] - x_i[i]}$
        end for
        if $p[k] = 0$ then
            $\kappa_y[k] \leftarrow 1$
        else
            $\kappa_y[k] \leftarrow \dfrac{sum\_abs\_y}{|p[k]|}$
        end if
        $\kappa_1[k] \leftarrow |sum\_abs\_1|$
    end for
    $\Lambda_n \leftarrow \max(\kappa_1)$
    $H_n \leftarrow \max(\kappa_y)$
    return $p, \Lambda_n, H_n$
end procedure
```

Since the above algorithm requires division by $(x - x_i)$, we must anticipate and handle situations where $x \approx x_i$, meaning an evaluation node x is approximately (or precisely) equal to any of the interpolation nodes x_i. To avoid potential numerical instability or a "division by zero" error, we directly assign y_i to the polynomial $p(x)$. The value of y_i is known a-priori. We also formulate this algorithm such that we can directly compute the condition numbers $\kappa(x, n, 1)$ and $\kappa(x, n, y)$ in order to find the condition bounds Λ_n (33) and H_n (34), respectively. This algorithm requires $O(kn^2) \approx O(n^2)$ complexity, since we evaluate k test nodes x. Our output is the polynomial of length k, hence $O(k)$ space is required for this routine.

The following routine produces a set of interpolation nodes $\{x_i\}_{i=0}^n$ on a set interval $[-1, 1]$ and evaluates the corresponding coefficients $\beta_i(x)$ of the Barycentric form 2 for the x_i.

Algorithm 7 Compute Barycentric 2 Weights

```
1:  procedure BARY2_WEIGHTS($flag, n, a = -1, b = 1$)
2:      $\beta \leftarrow [1, 1, \dots, 1]$ (length $n + 1$)
3:      $x_i \leftarrow [0, 0, \dots, 0]$ (length $n + 1$)
4:      if $flag = 1$ then                                  ▷ Uniform Nodes
5:          $x_i \leftarrow \text{linspace}(a, b, n + 1)$
6:          $\beta[0] \leftarrow 1$
7:          for $i \leftarrow 1$ to $n$ do
8:              $\beta[i] \leftarrow \beta[i-1] \cdot (-1) \cdot \frac{(n-i+1)}{i}$
9:          end for
10:     else if $flag = 2$ then                             ▷ Chebyshev First Kind
11:         for $i \leftarrow 0$ to $n$ do
12:             rad $\leftarrow \frac{(2i+1)\pi}{2(n+1)}$
13:             $x_i[i] \leftarrow \cos(\text{rad})$
14:             $\beta[i] \leftarrow \sin(\text{rad}) \cdot (-1)^i$
15:         end for
16:     else if $flag = 3$ then                             ▷ Chebyshev Second Kind
17:         for $i \leftarrow 0$ to $n$ do
18:             rad $\leftarrow \frac{i\pi}{n}$
19:             $x_i[i] \leftarrow \cos(\text{rad})$
20:             if $i = 0$ or $i = n$ then
21:                 $\beta[i] \leftarrow 0.5 \cdot (-1)^i$
22:             else
23:                 $\beta[i] \leftarrow (-1)^i$
24:             end if
25:         end for
26:     end if
27:     return $\beta, x_i$
28: end procedure
```

The above routine accepts a "flag" to indicate the type of interpolation nodes that will be produced, coefficients β_i are computed with respect to the interpolation nodes used: uniform nodes (15), Chebyshev nodes of the first kind (17), or Chebyshev nodes of the second kind (19). The coefficients of the Barycentric form 2 routine require $O(n)$ time and space complexity.

The following routine evaluates the polynomial $p_n(x)$ in Barycentric form 2.

Algorithm 8 Barycentric Form 2 Interpolation

```
procedure BARY2_INTERPOLATION($x, x_i, \beta, y_i$)
    $n \leftarrow$ len($x_i$)
    num $\leftarrow$ array of zeros with length $|x|$
    denom $\leftarrow$ array of zeros with length $|x|$
    $p \leftarrow$ array of zeros with length $|x|$
    for $k \leftarrow 0$ to len($x$) - 1 do
        for $i \leftarrow 0$ to $n - 1$ do
            if is_close($x[k], x_i[i]$) then
                $p[k] \leftarrow y_i[i]$
                break
            else
                term $\leftarrow \beta[i]/(x[k] - x_i[i])$
                num[$k$] $\leftarrow$ num[$k$]+ term $\cdot y_i[i]$
                denom[$k$] $\leftarrow$ denom[$k$]+ term
            end if
        end for
        if denom[$k$] $\neq 0$ then
            $p[k] \leftarrow$ num[$k$]/ denom[$k$]
        end if
    end for
    return $p$
end procedure
```

As with the Barycentric form 1 interpolation, the Barycentric form 2 interpolation also requires a division by $(x - x_i)$, so we again must anticipate and handle situations where $x \approx x_i$. To avoid the potential numerical instability or "division by zero" error, we again assign y_i directly to the polynomial $p(x)$. However, this routine is more time efficient than Barycentric form 1, as the Barycentric form 2 requires only $O(kn) \approx O(n)$ computations. The interpolating polynomial $p_n(x)$ evaluated at k points x again requires $O(k)$ space.

This next routine computes the divided differences, i.e. the coefficients, for the Newton basis polynomial.

Algorithm 9 Compute Newton Divided Difference

```
procedure NEWTON_DIVIDED_DIFF($x_i, y_i$)
    $n \leftarrow |x_i|$
    $y\_diff \leftarrow$ copy of $y_i$
    $coeffs \leftarrow [\, y\_diff[0] \,]$
    for $j \leftarrow 1$ to $n-1$ do
        for $i \leftarrow 0$ to $n-j-1$ do
            $y\_diff[i] \leftarrow \dfrac{y\_diff[i+1] - y\_diff[i]}{x_i[i+j] - x_i[i]}$
        end for
        Append $y\_diff[0]$ to $coeffs$
    end for
    $divided\_diff \leftarrow$ array formed from $coeffs$
    return $divided\_diff$
end procedure
```

The above algorithm computes the divided difference table recursively (22) and stores it in array form. The routine returns only the first column of the divided difference table, which corresponds to the coefficients of the interpolating polynomial. Construction of the divided differences table requires $O(n^2)$ computations, and the output requires only $O(n)$ storage. We save space complexity by only storing the coefficients we need, rather than the complete table.

The following routine uses Horner's rule to evaluate the Newton interpolating polynomial $p_n(x)$ using a nested form (25).

Algorithm 10 Horner's Rule for Polynomial Evaluation

```
procedure HORNER($x, x_i, y\_\text{diff}$)
    $n \leftarrow |x_i|$
    $p \leftarrow y\_\text{diff}[n-1]$
    for $i \leftarrow n-2$ to $0$ do
        $p \leftarrow (p \cdot (x - x_i[i])) + y\_\text{diff}[i]$
    end for
    return $p$
end procedure
```

By evaluating the interpolating polynomial using this method, we are able to construct an algorithm that computes the polynomial for k values x, using only $O(kn)$ operations. The interpolated polynomial output requires $O(k)$ space.

We develop an algorithm to order the interpolation nodes $\{x_i\}_{i=0}^n$ in either increasing order, decreasing order, or Leja ordering.

Algorithm 11 Ordering of Points with Leja Ordering Option

```
1:  procedure ORDERING($x_i$, flag)
2:      if flag $= 1$ then
3:          sort $x_i$ in increasing order
4:      else if flag $= 2$ then
5:          sort $x_i$ in increasing order
6:          reverse $x_i$
7:      else if flag $= 3$ then
8:          $x_{\text{remaining}} \leftarrow$ copy of $x_i$
9:          $index_0 \leftarrow \arg\max(x_{\text{remaining}})$
10:         $x_{\text{leja}} \leftarrow [\, x_{\text{remaining}}[index_0]\,]$
11:         Remove $x_{\text{remaining}}[index_0]$ from $x_{\text{remaining}}$
12:         while $|x_{\text{remaining}}| > 0$ do
13:             for each $x \in x_{\text{remaining}}$, compute
                    $$\text{prod}(x) \leftarrow \prod_{y \in x_{\text{leja}}} |x - y|$$
14:             $index \leftarrow \arg\max(\{\text{prod}(x) : x \in x_{\text{remaining}}\})$
15:             Append $x_{\text{remaining}}[index]$ to $x_{\text{leja}}$
16:             Remove $x_{\text{remaining}}[index]$ from $x_{\text{remaining}}$
17:         end while
18:         $x_i \leftarrow x_{\text{leja}}$
19:     end if
20:     return $x_i$
21: end procedure
```

Leja ordering is useful for minimizing the floating point error between the interpolating polynomial $p_n(x)$ and the true polynomial value $f(x)$ for a node x. The algorithm for the Leja ordering of points satisfies the following conditions for set of nodes $\{x_i\}_{i=0}^n$:

$$x_0 = \max_i |x_i|, \quad x_j = \max_{i \geq j} \prod_{k=0}^{j-1} |x_i - x_k|. \tag{35}$$

The follow algorithms perform several error analysis computations comparing the results of the interpolating polynomial evaluated at a value x to the chosen function $f(x)$ evaluated at a value x.

Algorithm 12 Evaluate Interpolation Error

```
1: procedure EVALUATE_ERROR($p, f, x$)
2:     $r \leftarrow p - f(x)$
3:     $norm \leftarrow \|r\|_\infty$
4:     $avg \leftarrow \text{mean}(r)$
5:     $var \leftarrow \text{variance}(r)$
6:     return $norm, avg, var$
7: end procedure
```

Algorithm 13 Relative Error Calculation

```
1: procedure RELATIVE_ERROR($p$, $y\_true$)
2:     $rel\_error \leftarrow$ array of zeros like $p$
3:     for $i \leftarrow 0$ to $|y\_true| - 1$ do
4:         if is_close($y\_true[i]$, 0, tol = $10^{-12}$) then
5:             $rel\_error[i] \leftarrow |p[i] - y\_true[i]|$
6:         else
7:             $rel\_error[i] \leftarrow \dfrac{|p[i] - y\_true[i]|}{|y\_true[i]|}$
8:         end if
9:     end for
10:    return $rel\_error$
11: end procedure
```

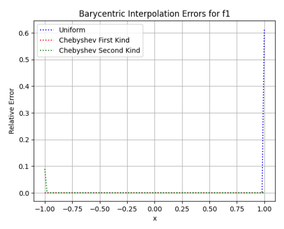
The algorithm computes the relative error $\frac{|p_n(x)-f(x)|}{f(x)}$, for all tested values x. In the next section, we examine norm, average, and variance outputs and plot the error terms to assess the behavior and stability of our algorithms.

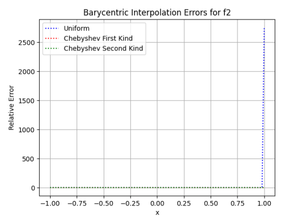
5 Description of Experimental Design and Results

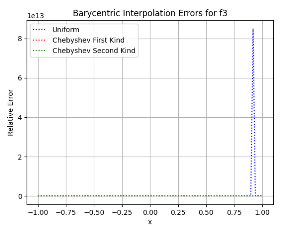
For the following tests, we generate a set of 100 uniform test nodes $\{x\}$ on the interval $[-1+(10^3 \times eps), -1-(10^3 \times eps)]$. For each function, we evaluate the true values y such that we have 100 test nodes (x, y). We examine the behavior of the relative errors for the Barycentric form 2 and the Newton form of the interpolating polynomials. For each method, we examine the behavior of each function when interpolated using uniform nodes, Chebyshev nodes of the first kind, and Chebyshev nodes of the second kind. Additionally, for the Newton form, we examine the difference in behavior, for each type of ordering: increasing order, decreasing order, and Leja order. The corresponding condition numbers, Λ_n and H_n, are summarized in a table for each combination of function, node type, and order type tested.

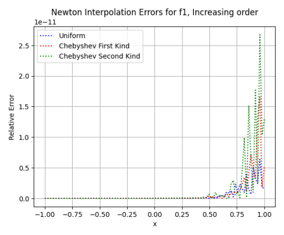
5.1 Testing Functions $f_1(x)$, $f_2(x)$, and $f_3(x)$

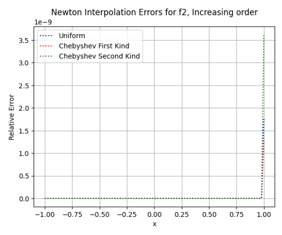
The following plots are the result of tests performed in IEEE standard double-precision floating point representation:

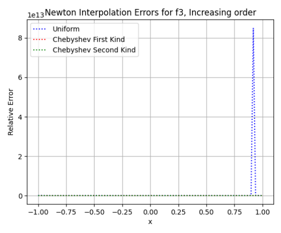


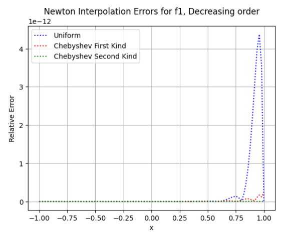


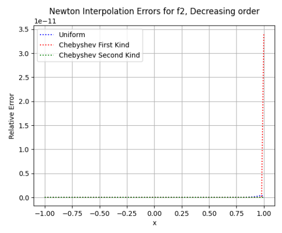


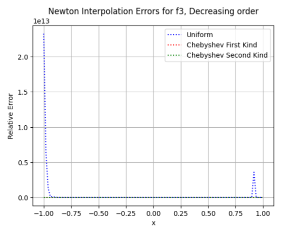


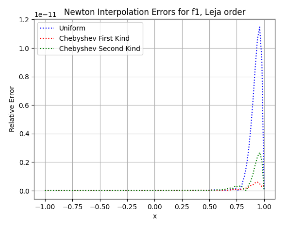


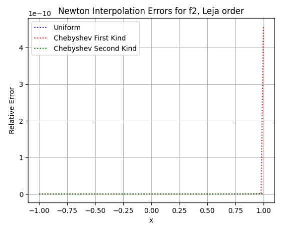


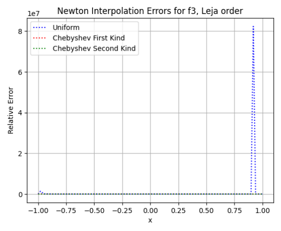












The above plots provide a strong result for our algorithms regarding the above functions tested. For $f_1(x)$, the relative error for the uniformly nodes increases as the test values x approach 1, and remains close to 0 for both Chebyshev kind nodes. We see a similar result for the Newton interpolation for $f_1(x)$. Particularly, for the nodes ordered in increasing order, we see the Chebyshev second kind nodes display larger relative error compared to the uniform and Chebyshev second kind. For the descreasing and Leja ordering, the largest relative error is seen from the uniform nodes, which increase in relative error as the test nodes approach 1. For $f_2(x)$, the Barycentric interpolation interpolation results in a large relative error as the uniform nodes approach 1. For the Newton interpolation for $f_2(x)$, the Chebyshev nodes of the second kind display the largest relative error, for all three ordering types. For $f_3(x)$, the relative errors increase as x approaches -1 on the Newton interpolation plot for x_i in decreasing order. For the remaining Newton plots and the Barycentric plots, the relative error increases as x approaches 1.

Consider the following table:

Function	Order	Flag	n	Method	Λ_n	H_n
f1		Uniform	10	Barycentric	29.897	13379.6
f1	Increasing	Uniform	10	Newton	29.897	13379.6
f1		Chebyshev First Kind	10	Barycentric	2.48943	1134.96
f1	Increasing	Chebyshev First Kind	10	Newton	2.48943	1134.96
f1		Chebyshev Second Kind	10	Barycentric	2.41955	482.719
f1	Increasing	Chebyshev Second Kind	10	Newton	2.41955	482.719

Function $f_1(x)$ has larger maximum condition numbers Λ_n and H_n for uniform interpolation nodes, meaning the problem is more numerical stable when interpolated using Chebyshev nodes of the first and second kind.

Function	Order	Flag	n	Method	Λ_n	H_n
f2		Uniform	10	Barycentric	29.897	2125.16
f2	Increasing	Uniform	10	Newton	29.897	2125.16
f2		Chebyshev First Kind	10	Barycentric	2.48943	2.27238e+13
f2	Increasing	Chebyshev First Kind	10	Newton	2.48943	2.27099e+13
f2		Chebyshev Second Kind	10	Barycentric	2.41955	101
f2	Increasing	Chebyshev Second Kind	10	Newton	2.41955	101

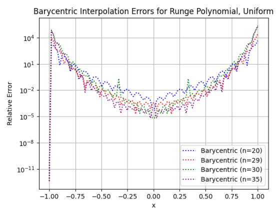
The condition numbers for $f_2(x)$ follow a similar pattern to the condition numbers for $f_1(x)$. We see $f_2(x)$ has a larger condition number Λ_n for uniform interpolation nodes, meaning the problem is more numerical stable when interpolated using Chebyshev nodes of the first and second kind.

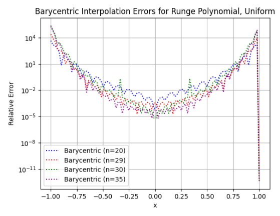
Function	Order	Flag	n	Method	Λ_n	H_n
f3		Uniform	30	Barycentric	6.27694e+06	1
f3	Leja	Uniform	30	Newton	6.27694e+06	1
f3		Chebyshev First Kind	30	Barycentric	3.14871	1.02264
f3	Leja	Chebyshev First Kind	30	Newton	3.14871	1.00004
f3		Chebyshev Second Kind	30	Barycentric	3.12391	1.14825
f3	Leja	Chebyshev Second Kind	30	Newton	3.12391	1

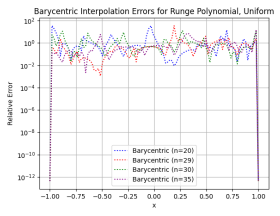
The function $f_3(x)$ displays a large relative error for the uniform interpolation nodes, resulting in a large maximum condition Λ_n for the Leja ordering, compared to the Chebyshev interpolation nodes, which are more numerically stable nodes for interpolation, for this function.

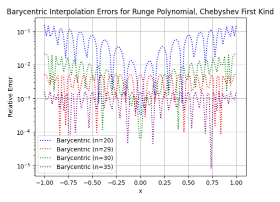
5.2 Testing Function $f_4(x)$

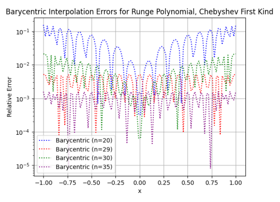
For the following plots, we test several sizes of interpolation nodes: $n = [20, 29, 30, 35]$. We show the results for the Barycentric form 2 polynomial interpolation. Each row is ordered by increasing, decreasing, and Leja ordering:

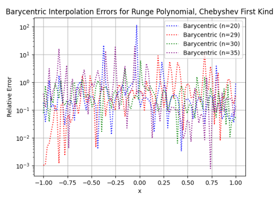


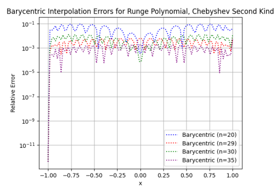


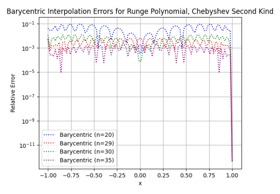


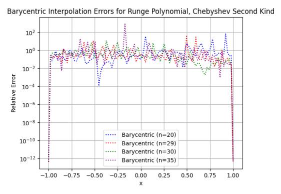




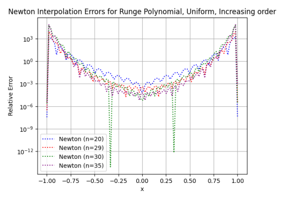


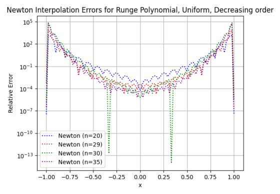


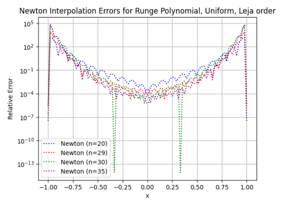


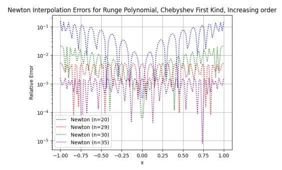


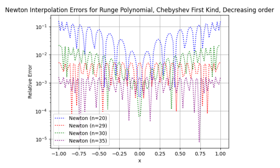
We show the results for the Newton polynomial interpolation:

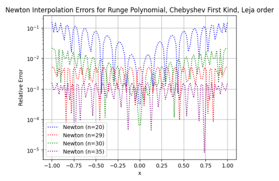


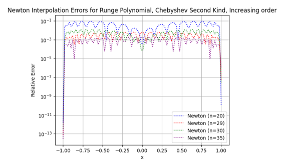


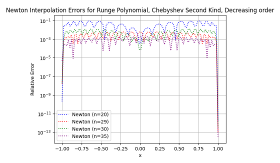


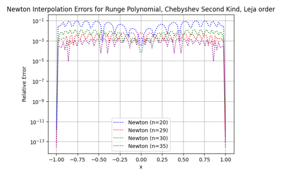












For both interpolation strategies displayed above, when $x = 0$, the function $f_4(x)$ becomes $f_4(0) = \frac{1}{1} = 1$. For the uniform interpolation nodes, we see the relative error decreasing around $x = 0$, and increase as x approaches both ends of the interval, for all three ordering types. The relative error is significantly less in value for the Chebyshev first kind and Chebyshev second kind interpolation nodes. The best result, with the lowest relative error across the interval $[-1, 1]$, appears in the Barycentric plot for the Chebyshev nodes of the second kind, ordered using Leja ordering. Across all of the plots, as the polynomial approach 30, we see a better approximation of the polynomial, with less variation in the relative errors. When $n = 35$, the polynomial experiences overfitting, and displays more error than when the polynomial is underfit to the graph.

6 Conclusions

We examined large relative error in our outputs. Considering we tested problems for which we knew the true function output, further examination must be done to examine the structure of the routines.

References

[1] Alfio Quarteroni and Fausto Saleri, “Numerical Methods for Scientists and Engineers,” *Springer*, 2000. https://link.springer.com/book/10.1007/978-3-662-04166-0, [Online; accessed 16-February-2025].

[2] Nicholas J. Higham, “The numerical stability of barycentric Lagrange interpolation ,” *IMA Journal of Numerical Analysis*, 2004.

[3] Jean-Paul Berrut and Lloyd N. Trefethen, “Barycentric Lagrange Interpolation,” *SIAM Review*, 2004.

[4] Dr. Kyle A. Gallivan, “Set 6: Polynomial Interpolation: Part 1”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University, February 2025.

[5] Dr. Kyle A. Gallivan, “Set 7: Polynomial Interpolation: Part 2”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University, February 2025.

[6] Dr. Kyle A. Gallivan, “Set 8: Polynomial Interpolation: Part 3”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University, February 2025.

[7] Dr. Kyle A. Gallivan, “Set 9: Polynomial Interpolation: Part 4”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University, February 2025.

[8] Dr. Kyle A. Gallivan, “Set 12: Interpolation Error, Conditioning, and Stability: Part 2”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University, February 2025.

[9] “Newton’s divided differences,” *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Divided_differences, [Online; accessed 16-February-2025].

[10] “Lagrange interpolation,” *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Lagrange_polynomial, [Online; accessed 16-February-2025].

[11] ‘Chebyshev nodes,” *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Chebyshev_nodes, [Online; accessed 16-February-2025].