# FCM 1 HW 2

Ryan Bausback

October 2020

## 1 Executive Summary

Two different interpolation strategies, Lagrange Barycentric Form 2 and Newton, were evaluated for their accuracy and stability when applied to four different functions, utilizing 2 different sets of mesh points each. A third mesh and the convergence was also investigated for the final function. Our results indicate that the interpolation methods are able to mimic the output of the true functions to single precision unit roundoff, except in extreme cases. We also found that the theoretical stability bound for Barycentric 2 holds.

## 2 Statement of Problem

Our goal is to evaluate two different polynomial interpolation strategies: the Lagrange Barycentric Form 2 and the Newton Form, and confirm empirically their theoretical accuracy and convergence for different functions, as well as that the theoretical bound for stability holds. To do this, four different functions will be evaluated, each selected to have properties that might make interpolation difficult, possibly disproving theory.

Three different meshes will be used to construct the interpolating polynomials. Those meshes consist of a uniform mesh of points with a specified interval [a,b], the Chebyshev points of the 1st kind, and the Chebyshev points of the 2nd kind. For consistence of comparison, all three meshes will use the same interval [a,b]. Differing orderings of the mesh points, as well as differing the degree of the interpolating polynomial will also be investigated. The results for specific functions are expected to align with the results of the numerical interpolation experiments in Dr. Nicholas Higham's paper and so specific parameterizations will be selected in order to compare direct with his results.[1]

## 3 Description of Mathematics

### 3.1 Lagrange Barycentric 2 Form

The first basis on which the interpolating polynomial will be constructed is on the Lagrange Barycentric 2 form. The basic Lagrange interpolation form is given as:

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

where $y_i$ is the output of the function (although it may have originated from data) and $l_i(x)$ is the Lagrange basis.[2] For our purposes, the Lagrange basis will be rewritten in terms of $\gamma_i$'s, by first interpolating the constant 1 as follows:

$$l_i(x) = \frac{\omega_{n+1}(x)}{(x - x_i)\omega'_{n+1}(x_i)}, \quad \omega_j = \prod_{i=0}^{j-1}(x - x_i)$$

$$\sum_{i=0}^{n} l_i(x) = \omega_{n+1}(x) \sum_{i=0}^{n} \frac{\gamma_i}{(x - x_i)} = 1$$

$$\omega_{n+1}(x) = \sum_{i=0}^{n} \frac{1}{\frac{\gamma_i}{(x - x_i)}}$$

If we then plug this $\omega_{n+1}(x)$ back into the orignal Lagrange form we are given:

$$p_n(x) = \omega_{n+1}(x) \sum_{i=0}^{n} \frac{y_i \gamma_i}{(x - x_i)} = \frac{\sum_{i=0}^{n} \frac{y_i \gamma_i}{(x - x_i)}}{\sum_{i=0}^{n} \frac{\gamma_i}{(x - x_i)}}$$

which is the Barycentric form 2.[2] For interpolation of the Chebyshev points, the $\gamma_i$'s will be replaced with $\beta_i$'s that can be evaluated in $O(n)$ or simpler. For the Chebyshev points of the first kind, given by $x_j = \cos\frac{(2j+1)\pi}{2n+2}$, $0 \le j \le n$, the $\beta_j$'s are defined as: $\beta_j = (-1)^j \sin\frac{(2j+1)\pi}{2n+2}$. For the Chebyshev points of the second kind, given by $x_j = \cos\frac{j\pi}{n}$, $0 \le j \le n$, the $\beta_j$'s are defined as: $\beta_j = (-1)^j \delta_j$, where $\delta_j = 0.5$ if $j = 0$ or $j = n$ and 1 otherwise. [2]

## 3.2 Newton Form

The Newton interpolating form is defined as:

$$p_n(x) = \sum_{i=0}^{n} y[x_0, ..., x_i]\omega_i(x)$$

where $\omega_i(x)$ is as defined above in the Lagrange form, and $y[x_0, ..., x_i]$ is the divided difference. The divided difference can be expressed in various ways but the form specified below will be utilized since we will already have to compute the $\gamma_i$'s to construct the Barycentric form 2 polynomial.[3]

$$y[x_0, ..., x_k] = \sum_{i=0}^{k} \frac{f(x_i)}{\omega'_{k+1}(x_i)} = \sum_{i=0}^{k} f(x_i)\gamma_i$$

In order to evaluate the Newton form, Horner's rule will be implemented to simplify computation.[3] It proceeds in the following way:

$$p_n(x) = (((\alpha_n(x - x_{n-1}) + \alpha_{n-1})(x - x_{n-2}) + ... + \alpha_1)(x - x_0) + \alpha_0$$

In this way, each $x - x_i$ only has to evaluate once, reducing the chance of errors from cancellation.

## 3.3 Stability and Conditioning

The absolute conditioning of the interpolation problem is defined in the following way:

$$||p_n(x) - \hat{p}_n(x)||_\infty \leq \Lambda_n ||y - \hat{y}||_\infty$$

where $\hat{p}_n(x)$ and $\hat{y}$ are the perturbed versions of the interpolant and the real function value at $x$ respectively, and $\Lambda_n$ is the Lebesgue constant, which is used as the condition number with respect to the max norm[4]. The Lebesgue constant is defined in terms of the Lagrange basis (as seen below) but, since conditioning is an analytical problem, can be used to evaluate interpolations on any basis.

$$\Lambda_n = ||\sum_{j=0}^{n} |l_j^{(n)}(x)|||_\infty$$

However, although the Lebesgue constant is independent of basis, it's form does depend on the set of mesh points used resulting in $\Lambda_n \approx \frac{2^{n+1}}{e*n*\log n}$ on a uniform mesh and $\Lambda_n \approx \frac{2}{\pi}\log n$ on the Chebyshev points. [5]

For the stability of the algorithm, the usual relative error form will be used to compute the difference between the "exact" double precision interpolating polynomial and the single precision polynomial. The error bound for Lagrange Barycentric Form 2 is defined in terms of a second, tighter conditioning number, defined as

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

which is the same form as the condition number of summation. When $y$ is identically 1 for all i, Then this condition number becomes the Lebesgue constant. The error is then bounded by:

$$\frac{|p_n(x) - \hat{p}_n(x)|}{|p_n(x)|} \leq (3n+4)\kappa(x,n,y)u + (3n+2)\kappa(x,n,1)u + O(u^2)$$

where u is the unit roundoff. [5] The error bound for the Newton interpolating polynomial is not well understood and therefore stability will be evaluated empirically instead of in relation to a bound.

## 4 Description of Algorithm and Implementation

Since the Barycentric Form 2 can be defined in terms of $\beta$'s, which can reduce computation to $O(n)$ or less for $n$ $x_i$'s, if-statements were utilized with a flag to determine the type of input mesh with which to compute the $\beta$'s. However, to reduce program complexity associated with evaluating the factorial in the $\beta$ for the uniform mesh, $\gamma_i$'s were used instead, as those will have to be computed for the Newton interpolation form anyways, resulting in a single computation. It's important to note that since the Chebyshev points are defined on interval [-1,1] as they are outputs of the cos function, that their corresponding $\beta$'s will also be defined on [-1,1]. In order to evaluate the interpolation on a larger interval such as [-a,a], the cos or sin output is multiplied by $a$ in order scale it over the entire interval. Then, after all of the $\beta$'s (or $\gamma$'s), the computed function values ($f(x_i)$'s), and the $x_i$'s themselves are computed in individual for-loops, they are passed into a 2-D array for output. This array is subsequently input into the evaluation function, along with an array of evaluation points (denoted as "xValues"), where two nested for-loops compute the values of the interpolant as described in 3.1. This requires $O(n)$ operations for each evaluation point.

Problems arise during evaluation when the evaluation point $x$ is identical to the interpolation point $x_i$, as Barycentric 2 requires division by $(x - x_i)$. However, we know that the output of the interpolating polynomial is supposed to be identical to $f(x_i)$ at $x_i$. Therefore, an if-statement / for-loop combination is used to check if $x = x_i$, and if so, $f(x_i)$ is output instead of the normal evaluation algorithm output.

The first step to compute the divided differences involves the computation of $\omega_{k+1}(x_i)$ as described in 3.1 for the $k$th divided difference. This is done in 2 nested for-loops, where the product is recursively multiplied on itself for each $x_i$, thus requiring $O(n^2)$ operations and $O(n)$ storage. Since we are using the $\gamma_i$ definition of the divided difference, an if-statement is utilized to determine whether or not to directly output the $\gamma_i$'s at this point. If not, then the true function values $f(x_i)$ are divided by these $\omega_{k+1}(x_i)$ in a for-loop to produce the divided differences as seen in 3.2, which are then output.

This output is then used to compute the Newton form $p_n(x)$'s using Horner's rule, as described in 3.2. For each evaluation point $x$ passed into the function, a for-loop recursively multiplies $(x - x_i)$ by the current product and then adds the divided difference computed above. This requires $O(n)$ operations for each evaluation point.

To compute the increasing and decreasing orderings to be used with the Newton form interpolation, simple bubble sort was used. However, the Leja ordering is more complicated. Step 1 involves putting the max value in the $x_i$'s in the first array slot. Within a for-loop, start with the current first value and if subsequent values are larger, switch that larger value with the one in the first position. Steps 2 and up involve finding the value that is furthest away from the values that have already been ordered. This is achieved by multiplying the absolute value of the difference between each unordered value and the ordered values and finding the max, which is placed in the next array slot. Two nested for-loops were utilized for this, thereby requiring $O(n^2)$ computations.

## 5 Experimental Design and Results

### 5.1 Accuracy Test

In order to validate the correctness of the programmed algorithm, the function $f(x) = x^2 + 1$ was used. This function was chosen because it is a very low degree polynomial without any roots, and since the interpolation methods to be tested also produce polynomials, we hypothesized that the error between the computed interpolant and the exact function to be almost negligible. The degree was set to $n = 2$ on the interval [-10,10] with a uniform, increasing mesh.

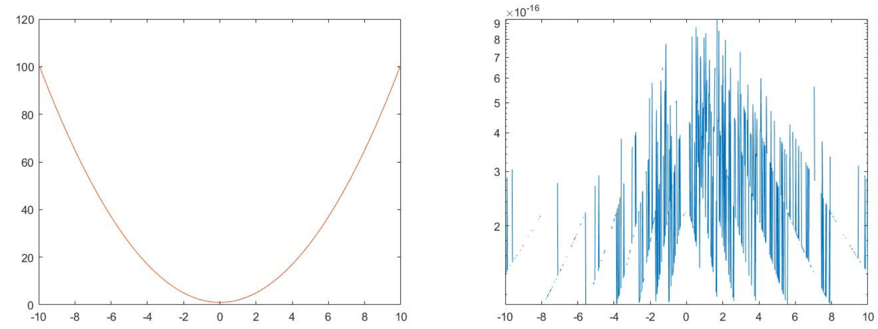

Figure 1: Uniform Mesh Barycentric 2 polynomial v. $f(x) = x^2 + 1$ and Relative Pointwise Error

This is exactly what happened. As Figures [1] and [2] indicate, the relative pointwise error between the interpolating polynomial and the true function value was of the order of $10^{-16}$ for Barycentric 2 and $10^{-14}$ for Newton, so close

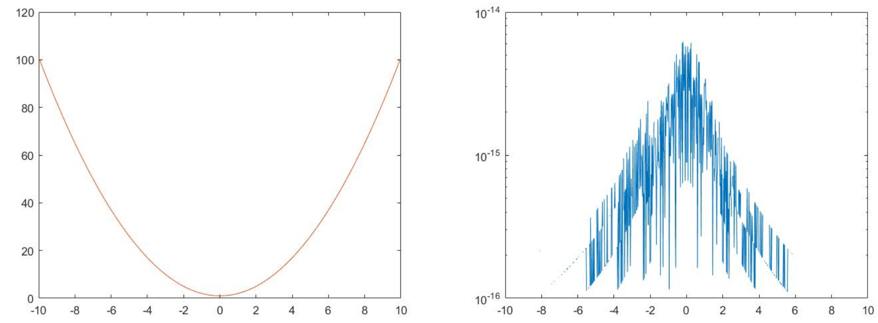

Figure 2: Increasing Uniform Mesh Newton polynomial v. $f(x) = x^2 + 1$ and Relative Pointwise Error

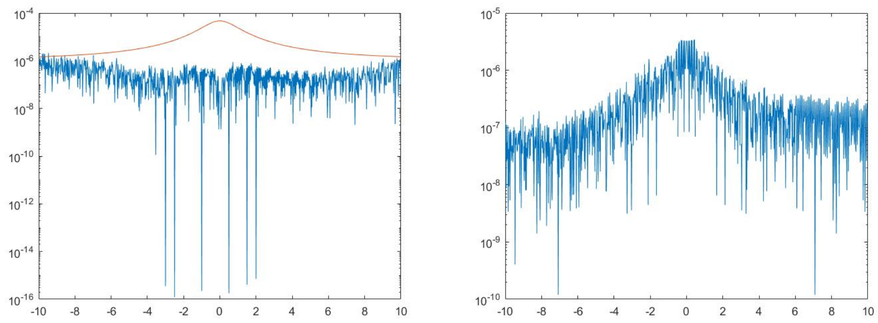

Figure 3: Double v Float Relative Pointwise Error: Lagrange(left), Newton(right) for $f(x) = x^2 + 1$

that the two polynomials are indiscernible from the real function in the plots. This illustrates that the algorithm is correctly interpolating. The results from stability tests were as expected as well. The relative error between the double and float versions of Barycentric 2 stayed well within the bound (in red in Figure3) described in 3.3. The Newton polynomial was also fairly stable, with the error less than $10^{-5}$ for all points. Overall, the algorithms performed as expected, illustrating the correctness of the code used to implement them.

## 5.2 Function 1: $f_1(x) = (x - 2)^9$

For functions 1, 2, and 3, it is impossible to determine the accuracy of the codes by plotting the true function values against the interpolant's values since they both include huge oscillations. In order to view any plot at all, they must be view from $10^{19}$ scale or greater, making it impossible to visually determine deviations. Therefore, only relative error plots will be used.

For the first experiment, the function $f(x) = (x-2)^9$ was chosen. Since this is a polynomial itself, the two interpolation strategies should be able to mimic and evaluate relatively easily with degree $n = 9$. However, some difficulty is predicted to occur near the root, since the root of 2 is repeated nine times. It will also be evaluated on $[-10, 10]$ as was the case for the accuracy test since $2 \in [-10, 10]$.

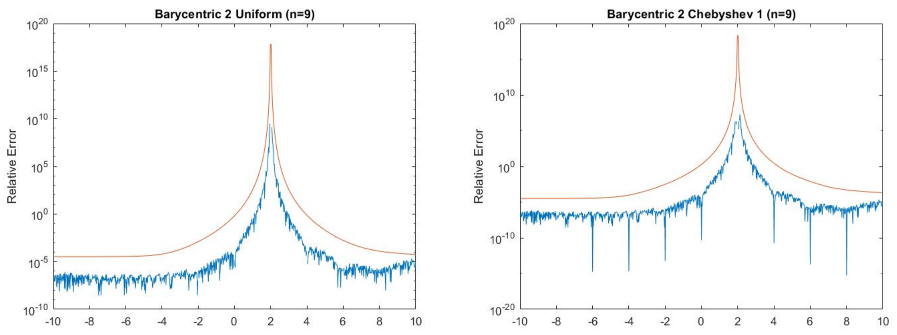

Figure 4: Barycentric 2 Pointwise Relative Error and Error Bound

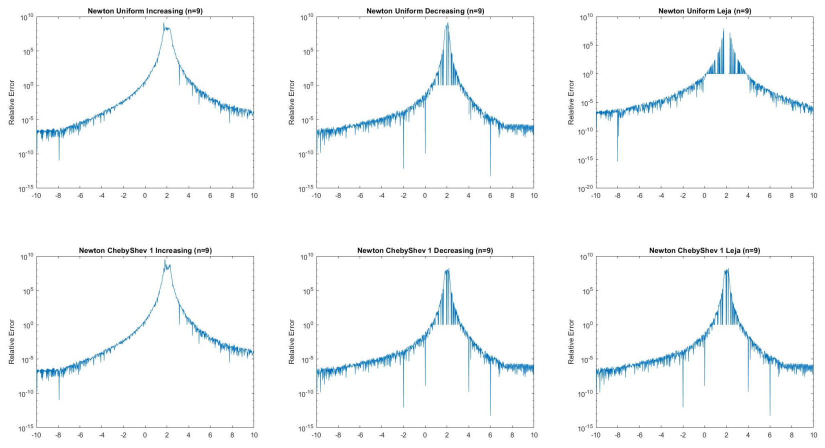

Figure 5: Newton Pointwise Relative Error

As expected, both interpolating polynomials become extremely unstable near the root, as characterized by the huge spikes in error in Figures 4 and 5. Neither performed significantly better than the other, however changing the mesh from a uniform mesh to the Chebyshev points of the first kind did result in marginal improvements for both interpolation strategies. Changing the order to the Leja ordering also saw marginal improvements for the Newton polynomial, with many fluctuations near the root. This was also the case for changing the ordering to decreasing. It is important to note that the error in the Barycentric form 2 polynomial remained bounded by the theoretical bound (in red), despite the increased instability near the root.

## 5.3 Function 2: $f(x) = \prod_{i=1}^{d}(x-i)$

The next function to be interpolated is $f(x) = \prod_{i=1}^{d}(x-i)$. Two different degrees for $d$ were tested: $d = 21$ and $d = 29$.

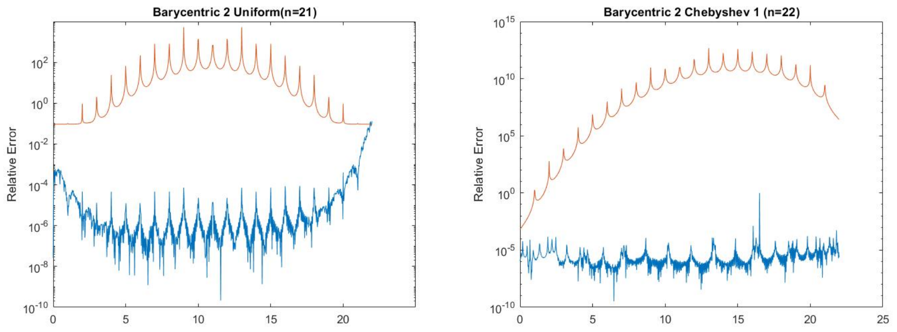

Figure 6: Barycentric 2 Pointwise Relative Error and Error Bound (N=21)

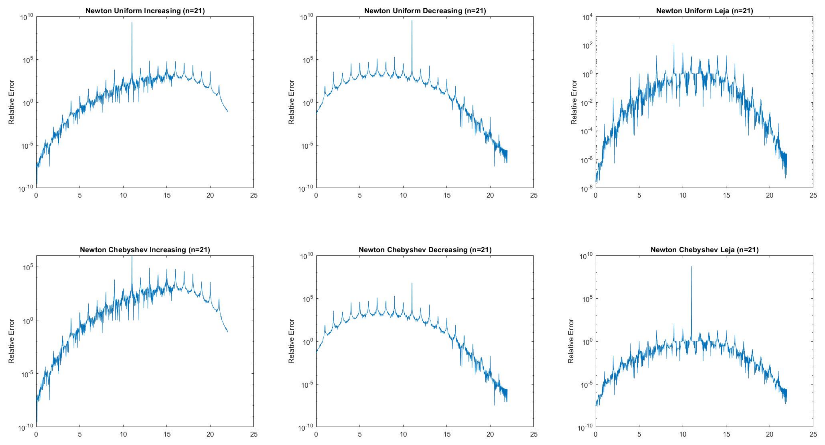

Figure 7: Newton Pointwise Relative Error (N=21)

This time, it is obvious that Barycentric Form 2 was much more stable than Newton, with the error remaining below $10^{-2}$ on both meshes. Changing the

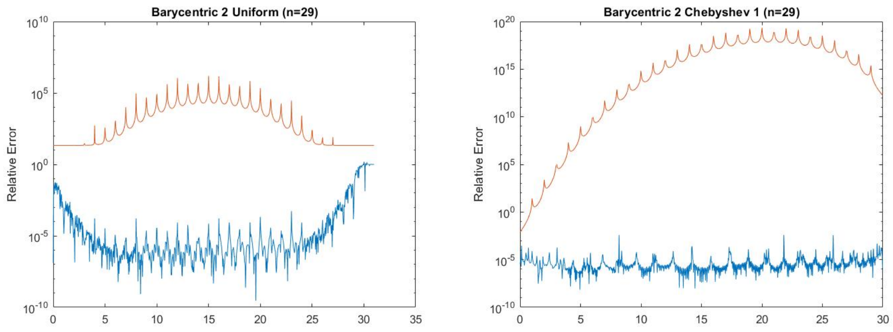

Figure 8: Barycentric 2 Pointwise Relative Error and Error Bound (N=29)

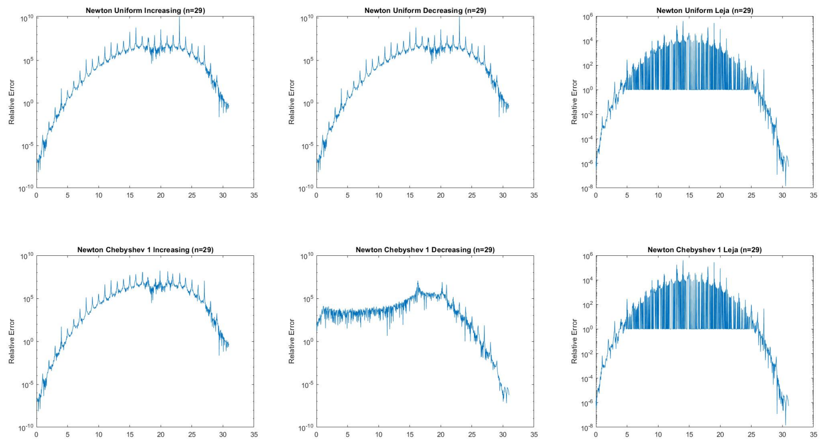

Figure 9: Newton Pointwise Relative Error (N=29)

ordering of the mesh did produce any significant difference in error for the Newton polynomial and neither did changing changing the mesh. The Leja ordering proved to be the most stable, but only marginally. It is also apparent that we are seeing the same behavior as in Dr. Higham's paper for Barycentric 2 on the uniform mesh, where the interpolation is more unstable near the endpoints.[1] This is especially apparent for degree $n = 29$, which is the same degree that he used, although the function itself is different.

Another thing of note is that there was increased instability near the roots of the polynomial for bother interpolation strategies, further emphasizing our observations from Function 1 in 5.2. This is characterized by the jagged appearance of the error and that it was generally much greater overall than with Function 1.

## 5.4 Function 3: $f_3(x) = l_n(x)$

For the third function, the same function that Dr. Higham used in his numerical experiments was chosen.[1] Degree $n = 29$ was chosen as well to conform to the paper's specifications and for easy comparison of results. Degree $n = 49$ was also chosen.

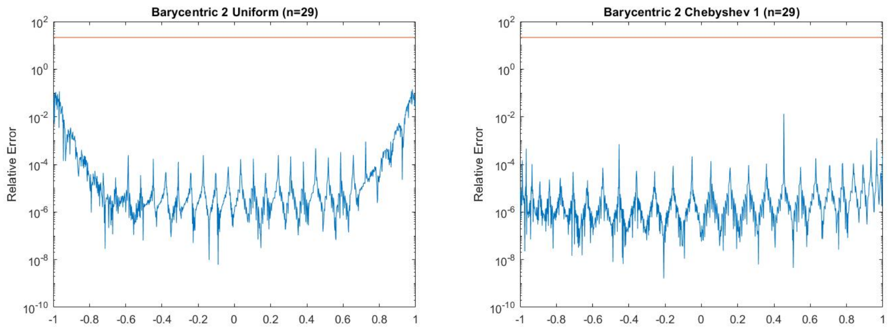

Figure 10: Barycentric 2 Pointwise Relative Error and Error Bound (N=29)

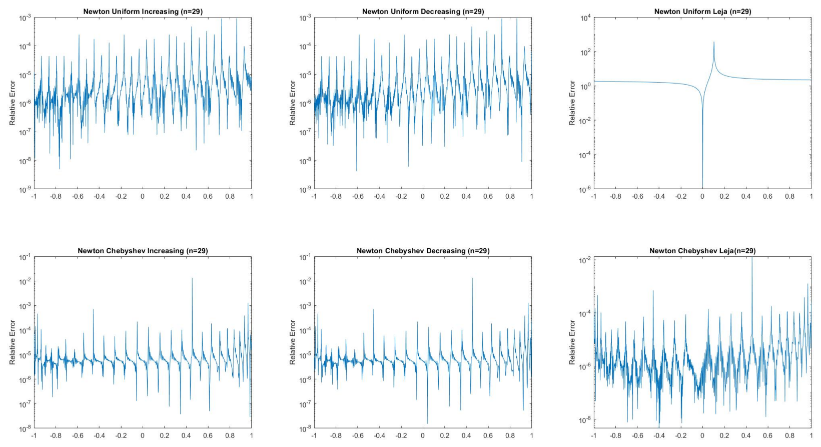

Figure 11: Newton Pointwise Relative Error (N=29)

The results for Barycentric 2 uniform mesh in Figures 10 and 12 show a close resemblance to the results in Dr. Higham's paper, as was also true for function 2.[1] The algorithm is much more stable near the center of the mesh than near the endpoints. The error for Barycentric 2 on the Chebyshev mesh also shows a close resemblance as well, hovering close to the unit roundoff. The difference is that since Dr. Higham was comparing the double precision output against the

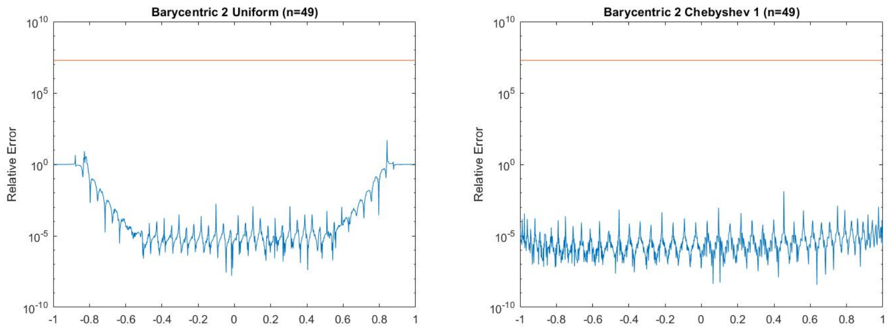

Figure 12: Barycentric 2 Pointwise Relative Error and Error Bound (N=49)

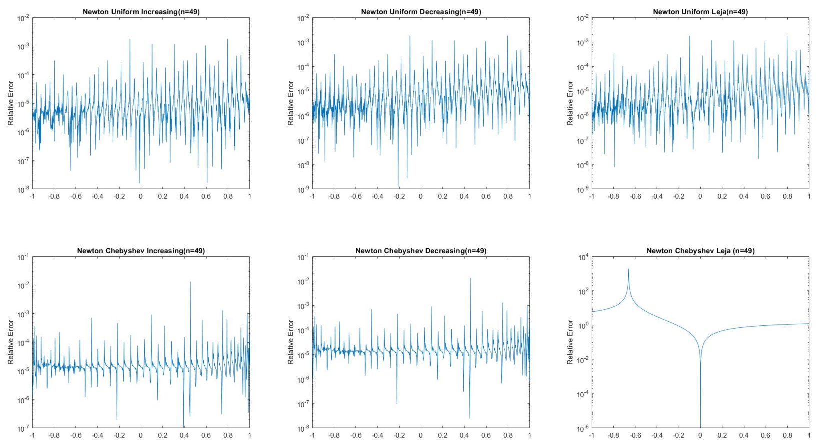

Figure 13: Newton Pointwise Relative Error (N=49)

matlab output, his error hovered around $10^{-16}$ or approximately unit roundoff for double precision.[1] In this case, since single precision is being compared against double precision, the error hovers around $10^{-7}$ or approximately unit roundoff for single precision.

For the the Newton polynomial, the relative error remained fairly constant for all orderings, fluctuating near unit roundoff for single precision. It failed to take on the same behavior seen in Dr. Higham's paper for the decreasing uniform ordering.[1] Interestingly, the pointwise error took on very strange shapes with the Leja ordering, spiking and dropping in unpredictable ways for both degrees of the polynomial tested.

## 5.5 Function 4: $f_4(x) = \frac{1}{(1+25x^2)}$

The final function on which the interpolation strategies will be tested is $f(x) = \frac{1}{(1+25x^2)}$. Degree $n = 29$ polynomials on interval $[-1, 1]$ will again be used to interpolate for easy comparison.

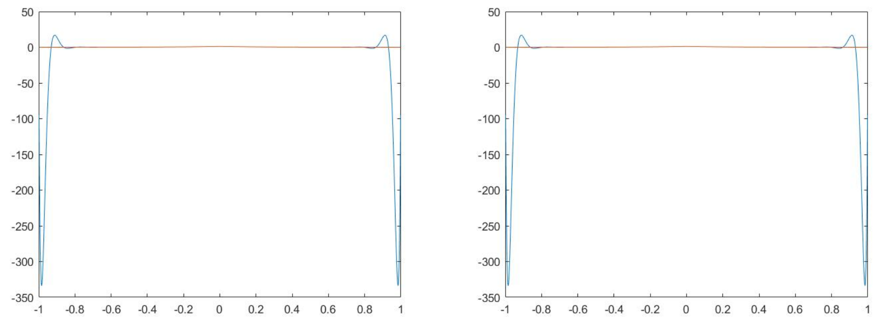

Figure 14: Barycentric 2(left) and Newton(right) Interpolating Polynomials on Uniform Mesh Plotted Against $f_4(x)$

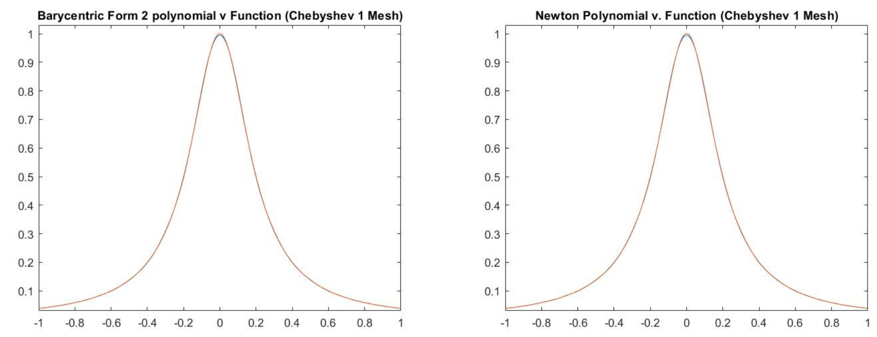

Figure 15: Interpolating Polynomials on Chebyshev 1 Mesh Plotted Against $f_4(x)$

When a rational function is being interpolated on a uniform mesh, the interpolant is known to experience Runge's Phenomenon, characterized by extreme fluctuations and deviations from the true function values near the endpoints. This is visible in Figure 14 for both the Barycentric Form 2 and Newton interpolation strategies. However, this is where the advantage of using the Chebyshev points of the first kind is apparent. By clustering many of the $x_i$'s near the endpoints, the approximation becomes much more accurate, visible in Figure 15 where the only slight deviation is noticeable near the function's peak at $x = 0$.

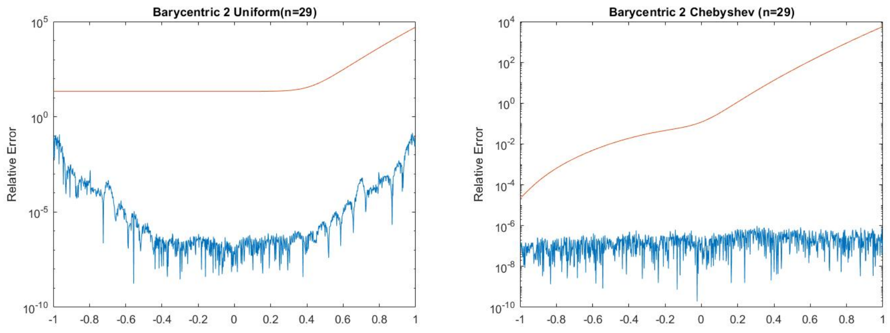

Figure 16: Barycentric Form 2 Pointwise Relative Error and Error Bound

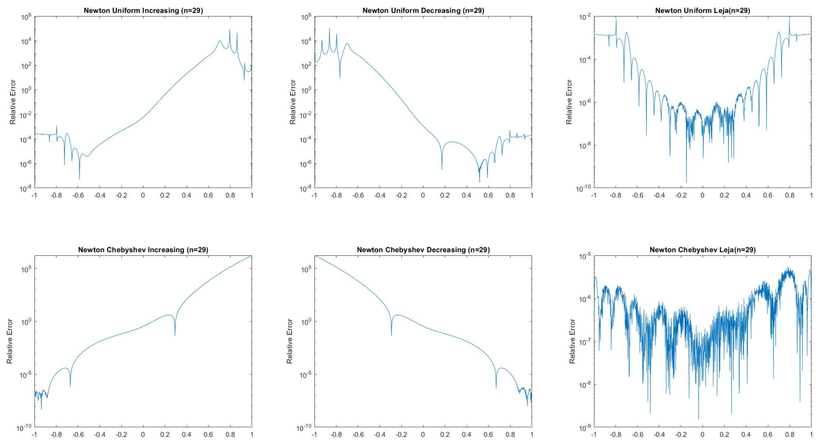

Figure 17: Newton Pointwise Relative Error

The relative pointwise error for both interpolation strategies seemed to closely follow the expected results from Dr. Higham's paper.[1] Barycentric Form 2 exhibited the same increased instability near the endpoints on the uniform mesh, with approximately constant error close to single precision unit roundoff on the Chebyshev 1 mesh. Both errors remained well below the theoretical error bound, continuing to confirm theory.

For the Newton polynomials, the behavior of the error was flipped by changing the order of the points from increasing to decreasing on both meshes, as was expected. Ordering the $x_i$'s according to the Leja ordering increased the stability and resulted in error close to single precision unit roundoff. On the uniform mesh, the error when Leja ordered also exhibited behavior similar to that of Barycentric 2 on the uniform mesh, with increased instability near the endpoints. Since this is the same region that Runge's phenomenon manifested itself for both polynomials, this error could coincide with it.

For this particular function, both interpolation strategies were also evaluated on the Chebyshev points of the second kind as well. The error between double and single precision behaved much the same way as with Chebyshev 1, as Figures 19 and 20 show. One notable difference is that the Barycentric form 2 polynomial does not approximate $f_4(x)$ quite as well when plotted for accuracy. However, the substantial decrease in computational complexity ($O(1)$) may be worth it in this case.

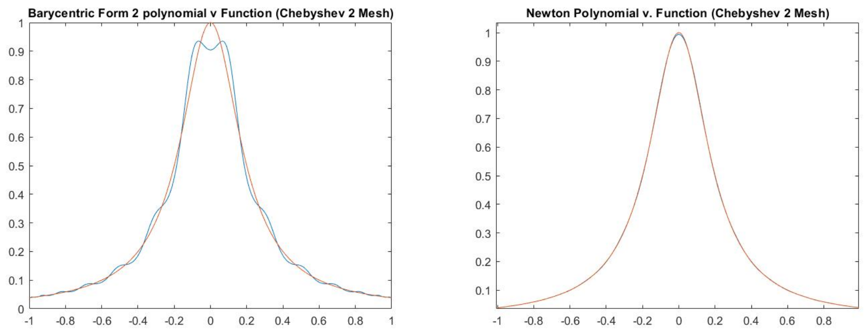

Figure 18: Interpolating Polynomials on Chebyshev 2 Mesh Plotted Against $f_4(x)$

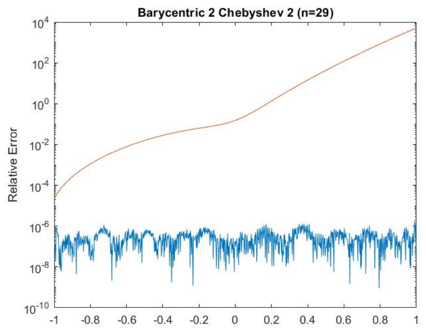

Figure 19: Barycentric Form 2 Pointwise Relative Error and Error Bound

The convergence of the Barycentric form 2 polynomial to $f_4(x)$ as $n$ increases was also investigated. To accomplish this, interpolating polynomials were computed using the normal algorithm and the max norm computed. This was then plotted against the degree of the interpolant to empirically determine if a limit exists.

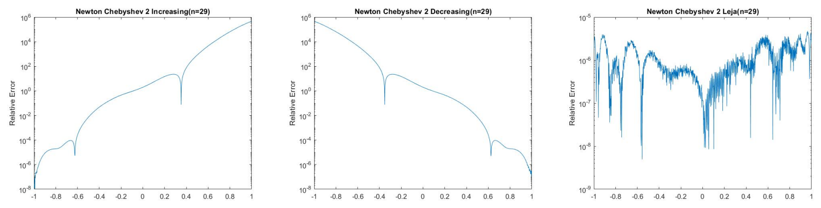

Figure 20: Newton Pointwise Relative Error

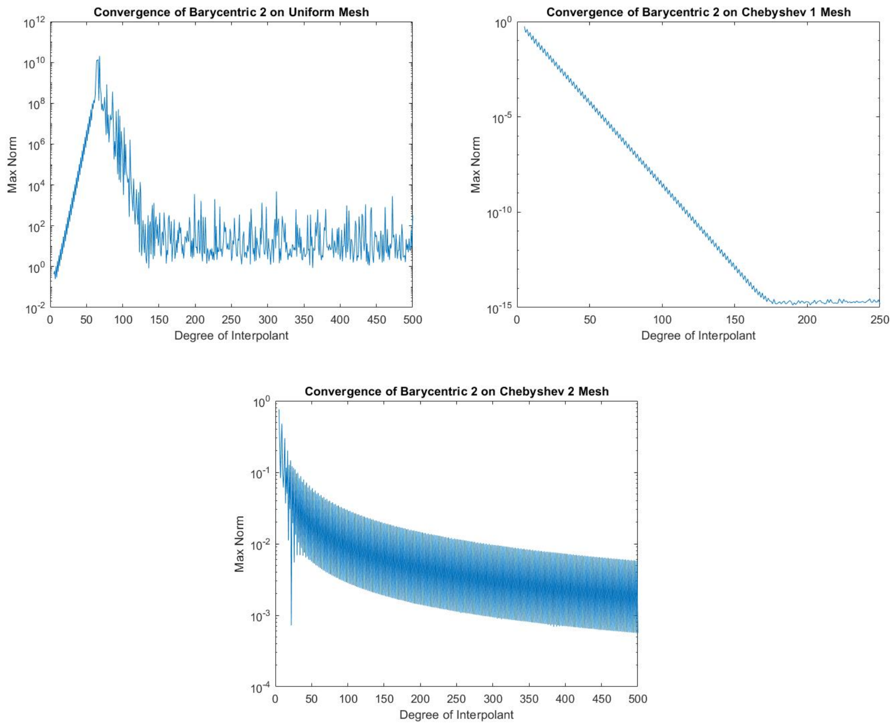

Figure 21: Convergence of Barycentric Form 2 as $n \nearrow$

From Figure 21, we can see that on the uniform mesh, the max norm continues to fluctuate around magnitudes greater than $10^2$ even as the degree of $p_n(x)$ increases towards 500. We can therefore say that Barycentric 2 fails to converge to the true function on the uniform mesh. On the Chebyshev points of the 1st kind, however, the max norm steadily decreases until it reaches $10^{-16}$ or unit roundoff for double precision. Past approximately $n = 175$, all of the values output by Barycentric form 2 will be identical within the machine to the real function values. The finite precision of the computer is the major limitation that prevents further convergence. In order to achieve "perfect" convergence within the machine, the degree of $p_n(x)$ should be set to around 200. For the Chebyshev points of the second kind, there appears to be convergence as well but not as quickly or as dramatically as with Chebyshev 1. The max norm oscillates as the degree increases, similar to the cos function that generated the mesh and $\beta$'s. In order to achieve output from $p_n(x)$ that is confidently within $10^{-2}$ of $f_4(x)$ at every single point, a polynomial of degree$> 250$ is required, while for $10^{-1}$, only degree$> 50$.

## 6 Conclusions

Overall, the results indicate that Barycentric form 2 is more stable than the Newton form for potentially difficult-to-interpolate functions. In cases were the relative pointwise error for Newton skyrocketed, it remained fairly level for Barycentric Form 2. Far from difficult points, the relative error tended to fluctuation around single precision unit roundoff or $10^{-7}$ for both methods, and then tended to significantly increase near those points, such as the roots of Functions 1 and 2. The pointwise relative error for Barycentric form 2 also remained less than the theoretical error bound in all cases, further confirming theory. The results from the numerical experiments in Dr. Higham's paper were also able to be duplicated for functions 2, 3, and 4.[1]

As anticipated the Chebyshev points of the 1st kind tended to be more stable than the uniform mesh. This was seen most effectively in Function 4, were Runge's phenomenon presented itself on the uniform mesh but was absent for Chebyshev 1 and 2. This was the case with functions 2 and 3 as well, were there was increased instability near the endpoints with Barycentric form 2 on the uniform mesh but not with the Chebyshev points.

The ordering of the $x_i$'s used to create $p_n(x)$ in the Newton method did not appear to have much of an impact on stability. It mainly appeared to influence which side of the evaluation interval $[a, b]$ the error was propagated. One notable exception is that the Leja ordering appeared to significantly reduce the relative pointwise error of function 4. This is in contrast to function 3, were more error appear with the Leja ordering when combined with paricular meshes and degree of the interpolant. It is therefore difficult to make a definitive conclusion about its impact.

## 7 Program Files

The program files were originally compiled in Java SE 9 on my local machine using Windows Powershell. To compile, "javac hw2.java" and then "java hw2". Then import the .txt file that is output into matlab and plot. All hard-coded values that must be altered to perform different experiments are marked with "- - - - - - - -".

## References

[1] Higham, N. J. “The Numerical Stability of Barycentric Lagrange Interpolation.” IMA Journal of Numerical Analysis, vol. 24, no. 4, 2004, pp. 547–556. https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Papers/HighamIMA2004.pdf

[2] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set7.pdf

[3] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set8.pdf

[4] https://mathworld.wolfram.com/L-Infinity-Norm.html

[5] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set11.pdf