Program 2

1 Executive Summary

In this report I will use 6 algorithms to interpolate two target functions and evaluate the numerical properties and conditionning of these algorithms.

2 Statement of the Problem

In this assignment, we are supposed to solve the polynomial interpolation problem, that is assume we are given a function $f(x)$ and n distinct points $x_0, x_1, ..., x_n$, we want to find a polynomial $p_n(x) \in \mathbb{P}_n$ s.t. $p_n(x_i) = f(x_i), i = 0, ..., n$.

3 Description of the Mathematics

In this section I will derive the mathematical expression for the algorithm that I will use to interpolate the target functions.

3.1 Barycentric form 2

We first define the characteristic polynomials $l_i(x) \in \mathbb{P}_n$, s.t.

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

Therefore, the desired p_n is simply

$$p_n(x) = \sum_{i=0}^{n} f(x_i) * l_i(x) \tag{2}$$

If we define the nodal polynomial of degree n+1 as

$$w_{n+1}(x) = \prod_{i=0}^{n} (x - x_i) \tag{3}$$

and we have

$$\begin{aligned} l_i(x) &= \prod_{j=0,j\neq i}^{n} \frac{x-x_j}{x_i-x_j} = \frac{\prod_{j=0,j\neq i}^{n}(x-x_j)}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)} = \frac{(x-x_i)\prod_{j=0,j\neq i}^{n}(x-x_j)}{(x-x_i)\prod_{j=0,j\neq i}^{n}(x_i-x_j)} \\ &= \frac{\prod_{j=0}^{n}(x-x_j)}{(x-x_i)\prod_{j=0,j\neq i}^{n}(x_i-x_j)} \\ &= w_{n+1}(x)\frac{w_i}{x-x_i} \end{aligned} \tag{4}$$

where w_i is defined as $w_i = \frac{1}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)}$. Also, if the target function f(x) is a constant function f(x)=1, the interpolating function $p_n(x)$ is just simply $p_n(x) = 1$, and from 2 we have

$$p_n(x) = \sum_{i=0}^{n} f(x_i) * l_i(x) = \sum_{i=0}^{n} l_i(x) = 1 \tag{5}$$

and from 4:

$$1 = \sum_{i=0}^{n} l_i(x) = \sum_{i=0}^{n} w_{n+1}(x)\frac{w_i}{x-x_i} = w_{n+1}(x)\sum_{i=0}^{n}\frac{w_i}{x-x_i} \tag{6}$$

thus

$$w_{n+1}(x) = \frac{1}{\sum_{i=0}^{n}\frac{w_i}{x-x_i}} \tag{7}$$

From 2, 4 and 7, we have

$$\begin{aligned} p_n(x) &= \sum_{i=0}^{n} f(x_i) * l_i(x) \\ &= \sum_{i=0}^{n} f(x_i) * w_{n+1}(x)\frac{w_i}{x-x_i} = w_{n+1}(x)\sum_{i=0}^{n} f(x_i) * \frac{w_i}{x-x_i} \\ &= \frac{\sum_{i=0}^{n}\frac{w_i}{x-x_i}f(x_i)}{\sum_{i=0}^{n}\frac{w_i}{x-x_i}} \end{aligned} \tag{8}$$

Thus we have the second form of the Barycentric interpolation formula 8. In order to evaluate the expression 8, one important task is to obtain $w_i = \frac{1}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)}, i = 0, ..., n$. If the mesh points x_i has some certain distribution, the expression for w_i could be simplified.

3.1.1 Uniform mesh points

Consider the mesh points x_i are uniformly distributed in the interval [a,b], and we have $x_i = a+ih$, where $h = (b-a)/n$. According to [1], the weights w_j can be calculated directly

$$\begin{aligned} w_j &= \frac{(-1)^{n-j}\binom{n}{j}}{h^n n!} = (-1)^j\binom{n}{j} * (\frac{(-1)^n}{h^n n!}) \\ &= \beta_j(\frac{(-1)^n}{h^n n!}) \end{aligned} \tag{9}$$

Since the factor $\frac{(-1)^n}{h^n n!}$ is independent of j, we only need to worry about β_j,that

$$\beta_j = (-1)^j\binom{n}{j} \tag{10}$$

Luckily, we have recursive expression for β_j,

$$\beta_{j+1} = -\beta_j \frac{n-j}{j+1} \tag{11}$$

3.1.2 Chebyshev points of the first kind

The Chebyshev points of the first kind are defined as

$$x_j = cos\frac{(2j+1)\pi}{2n+2}, \qquad j = 0, ..., n. \tag{12}$$

According to [2], the simplified weights β_j after canceling all the common factors are

$$\beta_j = (-1)^j sin\frac{(2j+1)\pi}{2n+2} \tag{13}$$

3.1.3 Chebyshev points of the second kind

The Chebyshev points of the second kind are defined as

$$x_j = cos\frac{j\pi}{n}, \qquad j = 0, ..., n. \tag{14}$$

Also, we the simplified weights β_j canceling all the common factors[3]

$$\beta_j = (-1)^j \delta_j, \qquad \delta_j = \begin{cases} 1/2, & j = 0 \quad or \quad j = n \\ 1, & otherwise \end{cases} \tag{15}$$

3.2 Newton divided difference form

Besides the above Bartcentric form, we can also write the interpolating polynomial in the follow form:

$$p_n(x) = p_{n-1}(x) + q_n(x) \tag{16}$$

where $q_n(x) \in \mathbb{P}_n$ and $p_{n-1}(x)$ interpolate the target function f(x) at $x_0, ..., x_{n-1}$. Since $q_n(x_i) = p_n(x_i) - p_{n-1}(x_i) = 0$ for i = 0, ..., n-1, we can then write $q_n(x)$ in the following the form:

$$q_n(x) = a_n(x - x_0)(x - x_1)...(x - x_{n-1}) = a_n w_n(x) \tag{17}$$

By convention, we also denote the coefficient a_n by $a_n = f[x_0, x_1, ..., x_n]$, and we now have:

$$p_n(x) = p_{n-1}(x) + f[x_0, x_1, ..., x_n]w_n(x) \tag{18}$$

And if we $f(x_0) = f[x_0]$ and $w_0 = 1$, by the above recursive relation, we can obtain the following formula:

$$p_n(x) = \sum_{k=0}^{n} w_k(x) f[x_0, ..., x_k] \tag{19}$$

By some algebraic manipulation from [6], we could have the explicit expresion for the coefficients:

$$f[x_0, ..., x_n] = \sum_{i=0}^{n} \frac{f(x_i)}{w'_{n+1}(x_i)} \tag{20}$$

where $w'_{n+1}(x_i) = \prod_{j=0, j\neq i}^{n}(x_i - x_j)$. Based on 19 and 20, we are able to evaluate the interpolating polynomial $p_n(x)$.

3.3 Bernstein polynomial

The n-th Bernstein polynomial for a real function $f(x)$ which is defined on [0,1] is denoted by $B_n(x)$, such that,

$$\begin{aligned} B_n(x) = B_n(x; f) &= \sum_{k=0}^{n} f(\frac{k}{n}) \binom{n}{k} x^k (1-x)^{n-k} \\ &= \sum_{k=0}^{n} g(k, n; f) * \phi_{n,k}(x) \end{aligned} \tag{21}$$

where $g(k, n; f) = f(\frac{k}{n})\binom{n}{k}$ and $\phi_{n,k}(x) = x^k(1-x)^{n-k}$.

In [6] Bartle showed that if $f(x) in C^{(0)}[0, 1]$ then $B_n(x)$ converges uniformly to $f(x)$ on [0,1], i.e.

$$\lim_{n \to \inf} \|f(x) - B_n(x)\|_{\inf} = 0. \tag{22}$$

Thus, in theory, we can use the Bernstein polynomial $B_n(x)$ to approximate the target function $f(x)$ if n is large enough.

4 Description of the Algorithm and Implementation

4.1 Barycentric form 2

In Barycentric form 2, since the weights w_i and $f(x_i)$, i=0,...,n are only related to the mesh points, to avoid excessive computation, I will store the all the values for w_i and $f(x_i)$, i=0,...,n when evaluating $p_n(x)$, which takes up O(n) space.

First, we need to the compute weight w_i i=0,...,n according to 11, 7 and 15 ,which all have O(n) complexity. Note that for 11, we first compute β_0 which need O(n) operations and then use the recursive relation 11 to compute all the rest β_i i=1,...,n, which also takes O(n) operations. In total, the algorithm for uniform mesh points is also O(n) in operation.

Algorithm 1 Pre-processing with different mesh points

int n
function fun
array x[n+1] ▷ mesh points
array y[n+1] ▷ function value
array w[n+1] ▷ weights
if MeshType==Uniform **then**
    diff=2/n
    factor = $(-1)^n/(\text{diff}^n n!)$
    for i=0 to n **do**
        x[i]= -1 + diff*i
        y[i]= fun(x[i])
        if i==0 **then**
            w[i]=1*factor
        else
            w[i+1] = w[i]*$\frac{n-i}{i+1}$*factor
if MeshType==Chebyshev1 **then**
    for i=0 to n **do**
        x[i] = $cos\frac{i\pi}{n}$
        y[i] = fun(x[i])
        w[i] = $(-1)^i sin\frac{(2i+1)\pi}{2n+2}$
if MeshType==Chebyshev2 **then**
    for i=0 to n **do**
        x[i] = $cos\frac{(2i+1)\pi}{2n+2}$
        y[i] = fun(x[i])
        w[i] = $(-1)^i$
    w[0]=w[0]/2
    w[n]=w[n]/2

Algorithm 2 Evaluating $p_n(x^*)$ in Barycentric form 2

int n
function fun
array x[n+1] ▷ x[n+1], y[n+1] and w[n+1] are already computed.
array y[n+1]
array w[n+1]
if x^* in x[:] **then** ▷ Return the exact function value if evaluated at the mesh point.
    return fun(x^*)
else
    numer=0
    denom=0
    for i = 0 to n **do**
        numer = numer + w[i]*y[i]/(x^* - x[i])
        denom = denom + w[i]/(x^* - x[i])
    return numer/denom

4.2 Newton divided difference form

The algorithm for Newton divided difference form will mainly based on equation 19 and 20. For computing the coefficients, I will adapt the ideas of Smoktunowicz et al [5]:

$$\begin{aligned} f[x_0, ..., x_n] &= \sum_{i=0}^{n} \frac{f(x_i)}{w'_{n+1}(x_i)} \\ &= \{\sum_{i=0}^{n-1} [\frac{f(x_i)}{w'_n(x_i)}] * \frac{1}{x_i - x_n}\} + \frac{f(x_n)}{w'_{n+1}(x_n)} \end{aligned} \tag{23}$$

And we also have the recursive relation for $w'_{n+1}(x_i)$:

$$w'_{n+1}(x_i) = w'_n(x_i)(x_i - x_n), \qquad i < n \tag{24}$$

And the following is the algorithm for generating all the coefficients:

Algorithm 3 Divided Difference Coefficients

```
int n
function fun
array d[n+1]
array x[n+1]
array f[n+1]                          ▷ f[n+1] is the divided difference coefficients
f[0] = fun(x[0])
d[0] = f[0]
for i = 1 to n do
    p = (x[0]-x[i])
    d[0] = d[0]/p
    s = d[0]
    for j = 1 to (i-1) do
        t = x[j] - x[i]
        d[j] = d[j]/t
        p = p * t
        s = s + d[j]
    d[i] = (−1)^i fun(x[i])/p
    f[i] = d[i] + s
```

Note that the above algorithm is $O(n)$ in space and $O(n^2)$ in operations. After obtain the coefficients, we could apply the following algorithm to compute $p_n(x^*)$, which is directly from 19,

Algorithm 4 Evaluating $p_n(x^*)$ in Newton divided difference form

```
array f[n+1]                          ▷ divided difference coefficients.
array x[n+1]                          ▷ mesh points
w = 1
s = f[0] * w
for i = 1 to n do
    w = w * (x* - x[i-1])
    s = s + w * f[i]
return s
```

Note that the above algorithm 4 require O(n) in operations and O(1) in space.

4.3 Bernstein polynomial

By the definition of Bernstein polynomial $B_n(x)$, it only aims to approximate smooth functions defined on [0,1], and in our problem, we need to approximate functions defined on [-1, 1]. For $x \in [-1, 0]$, we have $-x \in [0, 1]$ and $f(-x)$ would be the target function for approximating when $x \in [-1, 0]$. Thus, we divide the original interpolation problem into two separated problems i.e. interpolating $f(x)$ for $x \in [0, 1]$ and interpolating $f(-x)$ for $x \in [-1, 0]$.

When evaluating 21, $g(k, n; f) = f(\frac{k}{n})\binom{n}{k}$ is independent of the variable x, thus we can compute $g(k, n; f), \quad k = 0, ..., n$ and store these values. So when evaluating $B_n(x)$ we only need to compute $\phi_{n,k}(x) = x^k(1 - x)^{n-k}$ for different value of x. In general, computing $g(k, n; f), \quad k = 0, ..., n$ requires $O(n^2)$ operations and computing $B_n(x)$ require extra $O(n)$ operations.

Algorithm 5 Algorithm for the coefficients of $B_n(x)$ i.e. $g(k, n; f)$

```
function fun
int n
array g[n+1]                                   ▷ g(k, n; f)
array binomial[n+1]                            ▷ Binomial coefficients
for i = 0 to ceil(n/2) do
    temp = 1
    temp = temp*(n - i)/(i + 1)
    binomial[i] = temp
    binomial[n-i] = temp
for i = 0 to n do
    g[i] = binomial[i] * fun(i/n)
return g[:]
```

Algorithm 6 Algorithm for evaluating $B_b(x)$ given values $g(k, n; f)$

```
int n
initialize g_pos[n+1]                                   ▷ coefficients for fun(x)
initialize g_neg[n+1]                                   ▷ coefficients for fun(-x)
initialize g[n+1]
if x >= 0 then      ▷ Determine whether x in positive and choose the corresponding coefficients
    g = g_pos[n+1]
else
    g = g_neg[n+1]
    x = -x                                              ▷ make x positive
if x == 0 then                                          ▷ Evaluating the end points.
    return g[0]
if x == 1 then
    return g[n]
sum = 0
d = x/(1 − x)
temp = (1 − x)^n/d
for i = 0 to n do
    temp = temp * d
    sum = sum + g[i]*temp
return sum
```

5 Description of the Experimental Design and Results

In this section, I will use two different target functions on [-1,1] to test the numerical behaviour of our algorithms. And the two functions are:

$$f_1(x) = |x| + \frac{x}{2} - x^2 \tag{25}$$

and

$$f_2(x) = \frac{1}{1 + x^2} \tag{26}$$

For each algorithm, I will evaluate the value of $p_n(x)$ at 1000 equally spaced points on [-1,1] and compare these result with the true function value $f(x)$. Note that, the Bernstein polynomial divide the interpolation on [-1,1] into two separated problems, interpolation on [-1,0] and interpolation on [0,1]. To make the Bernstein polynomial comparable to other algorithms, I only use the Bernstein polynomial of degree n/2 in each interval.

5.1 Stability analysis

I will use the difference between the double and single version of the same algorithms to roughly measure the stability of the algorithms. For each functions, I test 3 degrees which are 10, 30 and 70 to explore the numerical stability for different algorithms at different degree. The a-axis is the value for which $p_n(x)$ is evaluated and y-axis is the error in log scale.

Figure 1: Error of single algorithm at degree of 10 (function1)

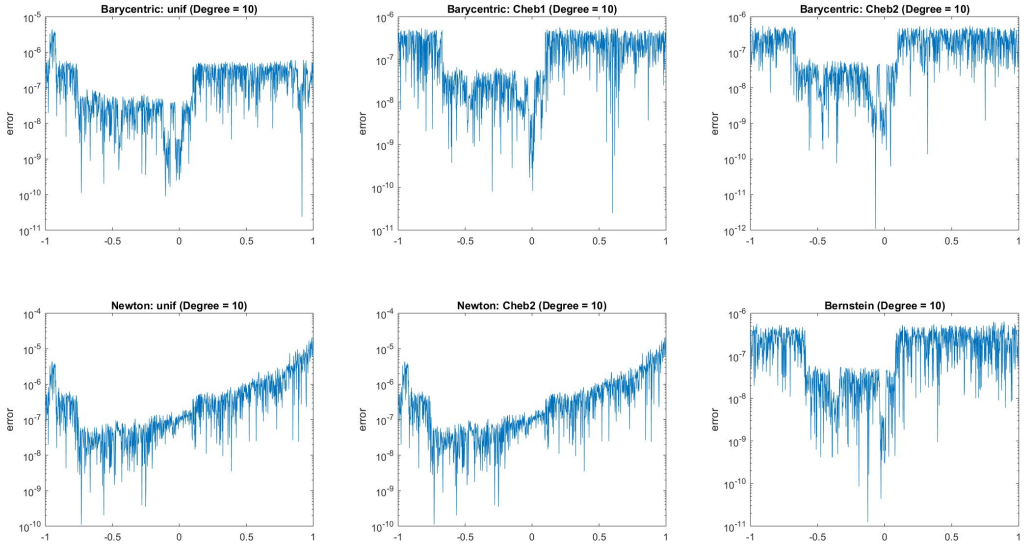


Figure 2: Error of single algorithm at degree of 30 (function1)

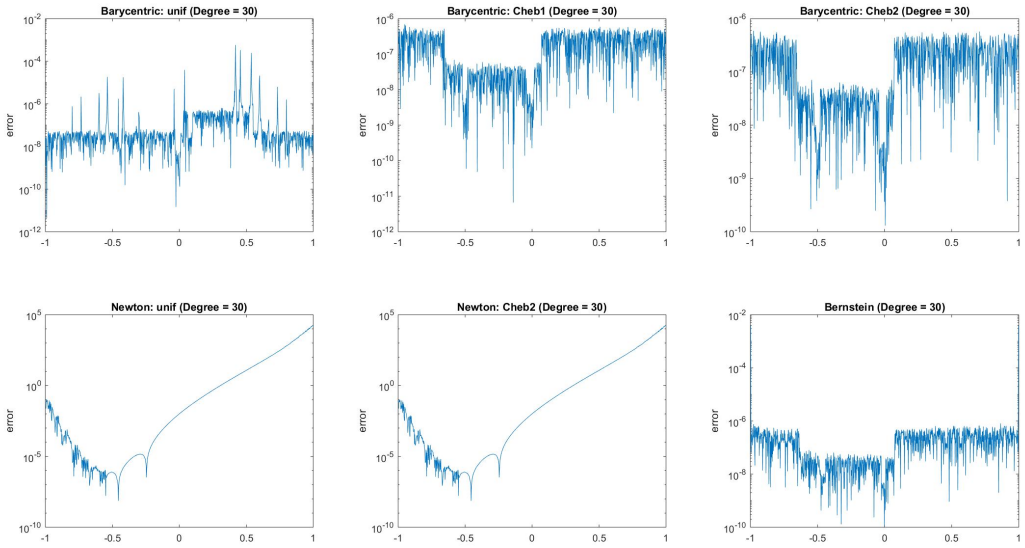


Figure 3: Error of single algorithm at degree of 70 (function1)

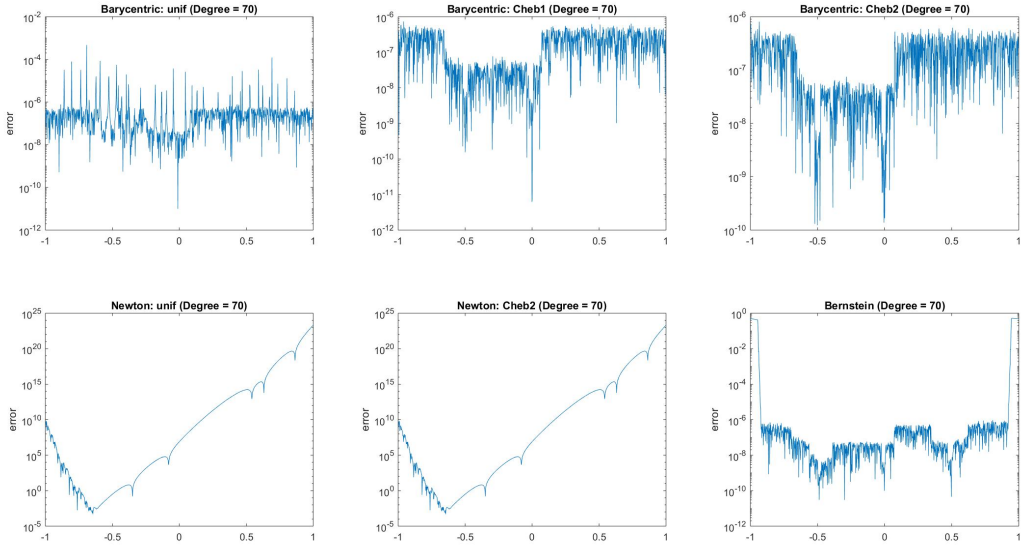


Figure 4: Error of single algorithm at degree of 10 (function2)

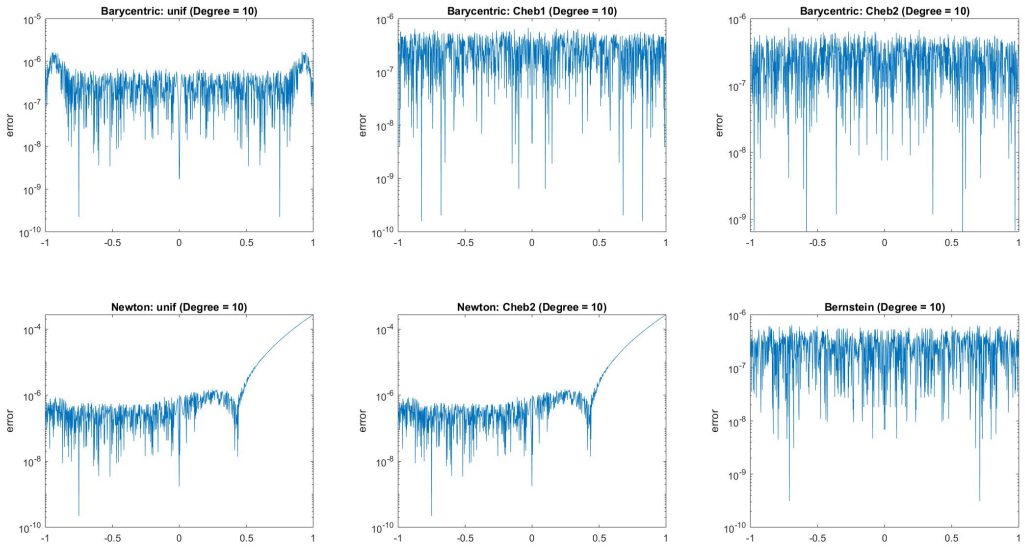


Figure 5: Error of single algorithm at degree of 30 (function2)

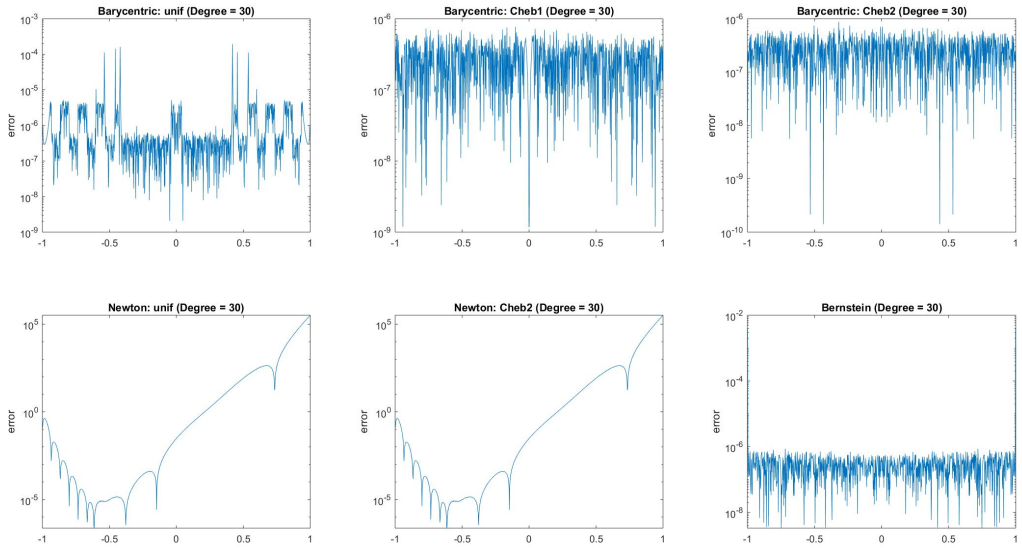
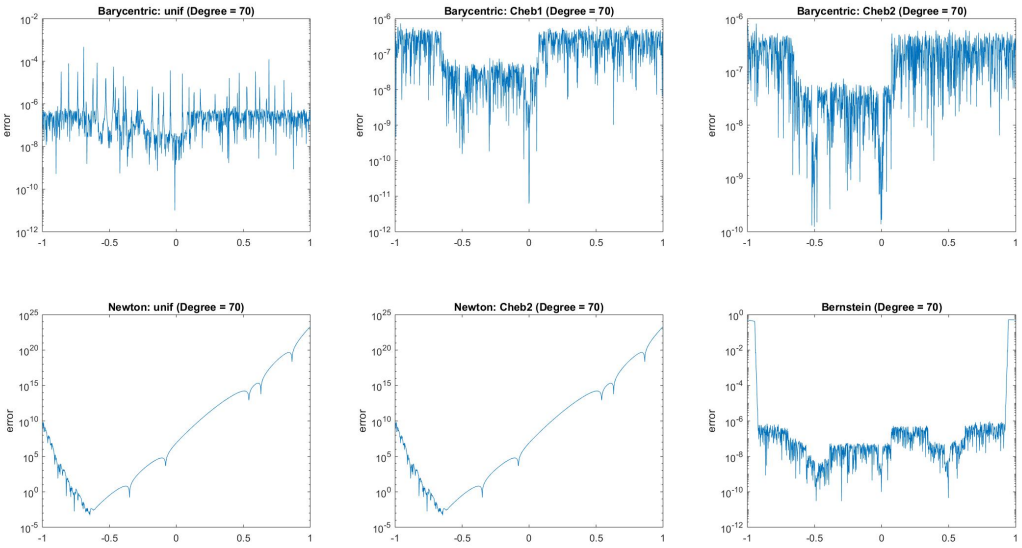


Figure 6: Error of single algorithm at degree of 70 (function2)



From the above results, it is obvious that the Newton divided difference form become extremely unstable when the degree is large as the error blow up when the degree is high in both functions.

The Barycentric form 2 with Chebyshev points is best performer in terms of the stability as the error is bounded by 10^{-6} in both functions even when the degree is very big.

For Barycentric form 2 with uniform points, the algorithm behaves good when the degree is small, but when the degree increases, for a small number of points the error blow up in scale, while most of the points have error that is bounded by 10^{-6}

For the Bernstein polynomial, the stability is good in general, but the only problem is the error will blow up near 1 when the degree becomes large.

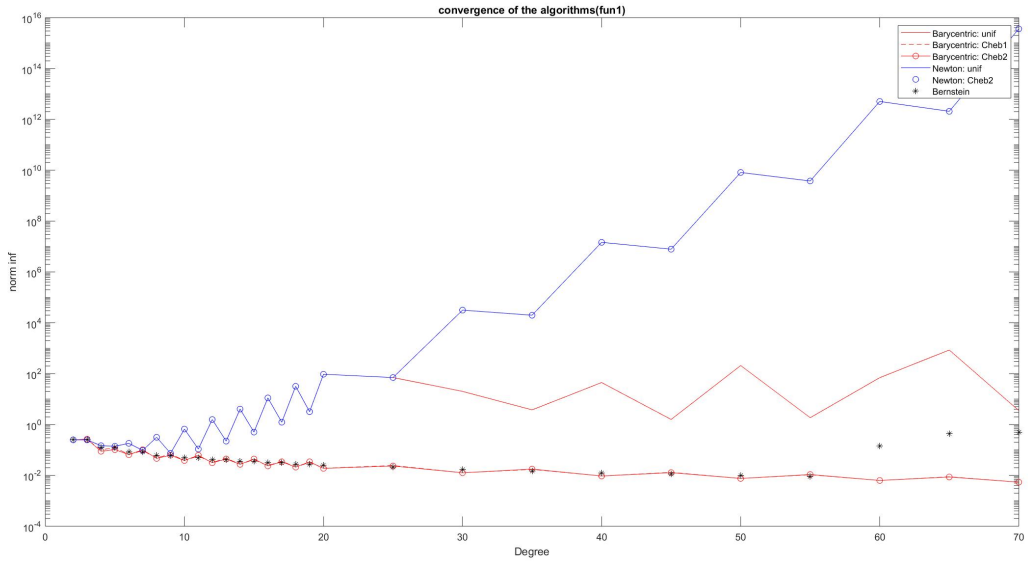
5.2 Convergence analysis

For the following convergence analysis and the conditioning analysis, I will only consider the double precision algorithms. To analysis the convergence, I use the infinity norm to define the interpolation error:

$$error = max(|f(x) - p_n(x)|), \quad x \in [-1, 1] \tag{27}$$

which is basically the largest difference between $p_n(x)$ and $f(x)$. The following results shows the relation between the degree and the error.

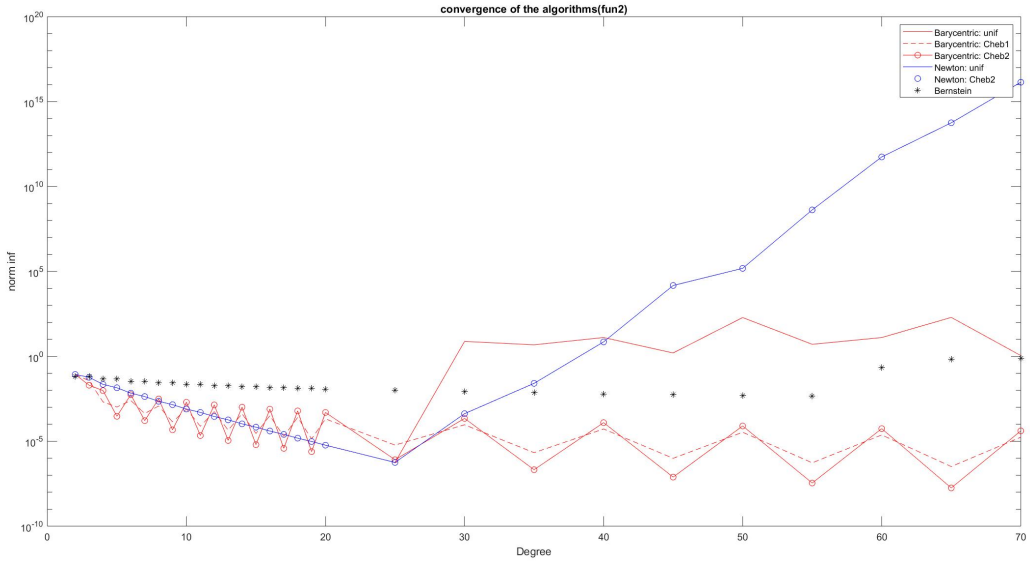
Figure 7: Maximum interpolation error V.S. degree (function1)



For function 1, the Bartcentric form 2 with Chebyshev points is also the best performer as the error decrease with the degree which indicates that interpolating function converge to the target function as degree increases. And they also have the fastest decrease in the error. The Bernstein polynomial also has good convergence when degree is below 55 as the Bartcentric form 2 with Chebyshev points, but when the degree is above 60, the error seems blow up and destroy the convergence.

Newton divided difference form and Bartcentric form 2 with uniform points behave okay when the degree in below 10, when the degree is above 10, the error begin to grow. For Newton divided difference form, the error grow exponentially, and for Bartcentric form 2 with uniform points the error seems growing much slower that Newton divided difference form. And they all have similar behaviour when degree is below 25.

Figure 8: Maximum interpolation error V.S. degree (function2)



For function 2, the Bernstein polynomial has the slowest rate of convergence while all other methods converge much faster. Similarly, when degree is above 25, the Newton divided difference form and Bartcentric form 2 with uniform points begin to diverge with similar behaviour that one is faster than the other. Bernstein polynomial begin to diverge when degree is above 55.

As function 2 is smooth, we could observe a faster convergence rate of the Newton and Barycentric form. When a target function is not smooth like function 1, the the Newton and Barycentric form would converge slowly as the Bernstein polynomial because the error at the break point cannot be eliminate completely by the smooth polynomial functions. And the following figures show the where the function value $p_n(x)$ converge or diverge in different degree.

Function 1: $(f_1(x) = |x| + \frac{x}{2} - x^2)$

Figure 9: $p_{50}(x)$ and true function value (degree of 50) (function1)

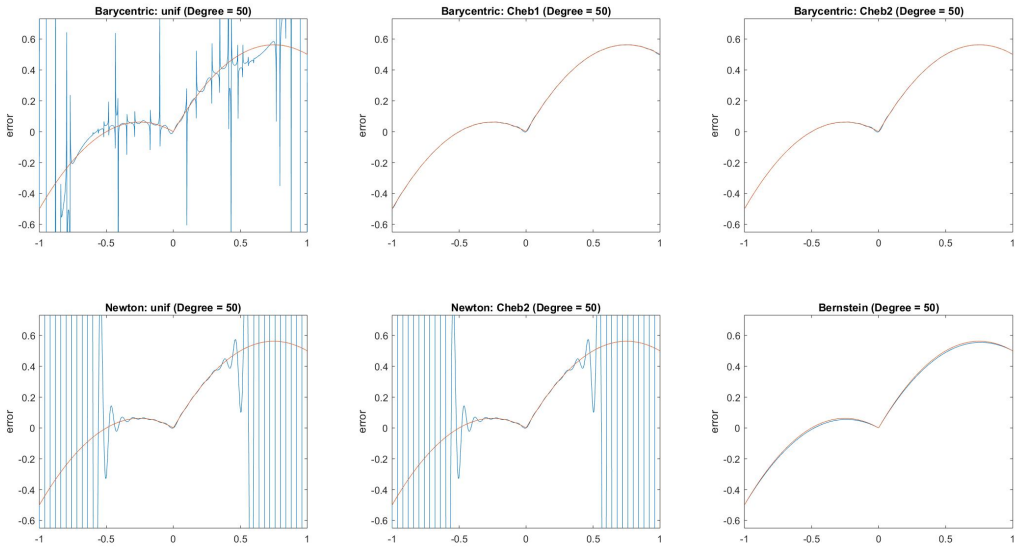


Figure 10: Interpolation error(log scale) of $p_{50}(x)$ (degree of 50) (function1)

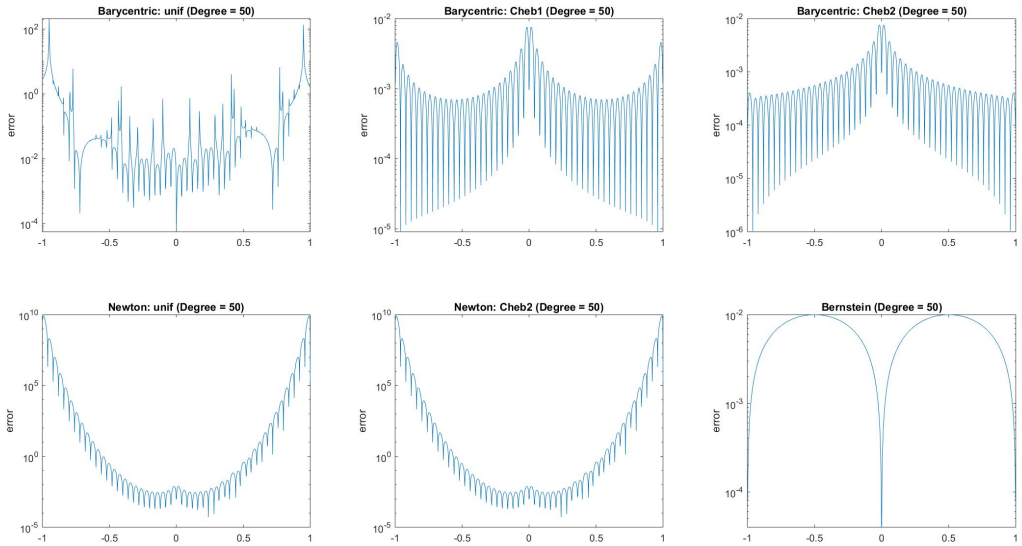


Figure 11: $p_{70}(x)$ and true function value (degree of 70) (function1)

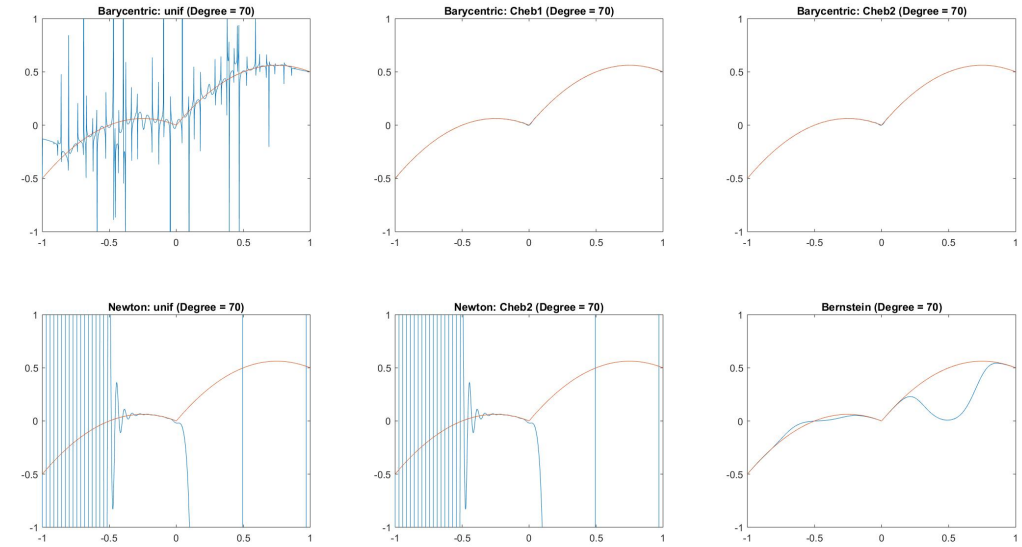
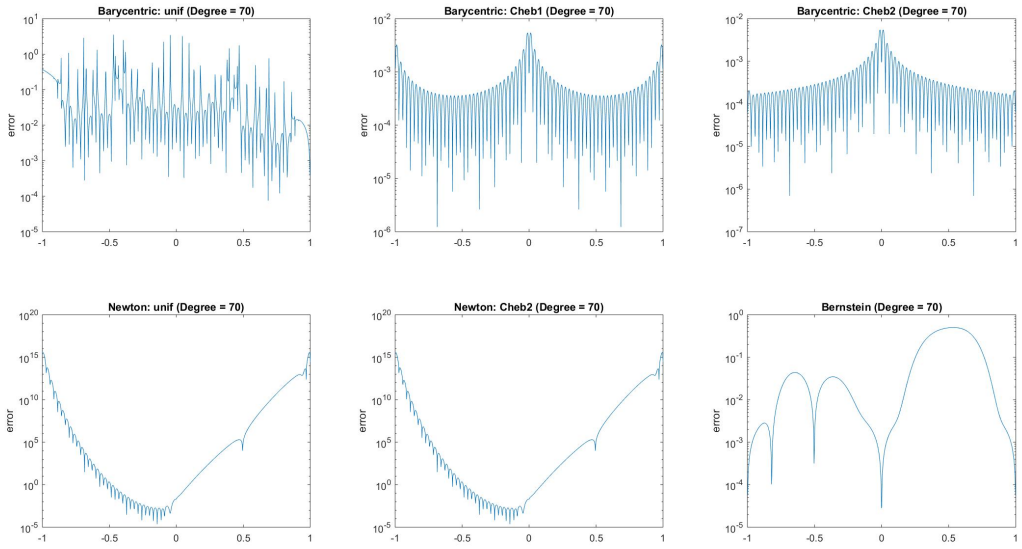


Figure 12: Interpolation error(log scale) of $p_{70}(x)$ (degree of 70) (function1)



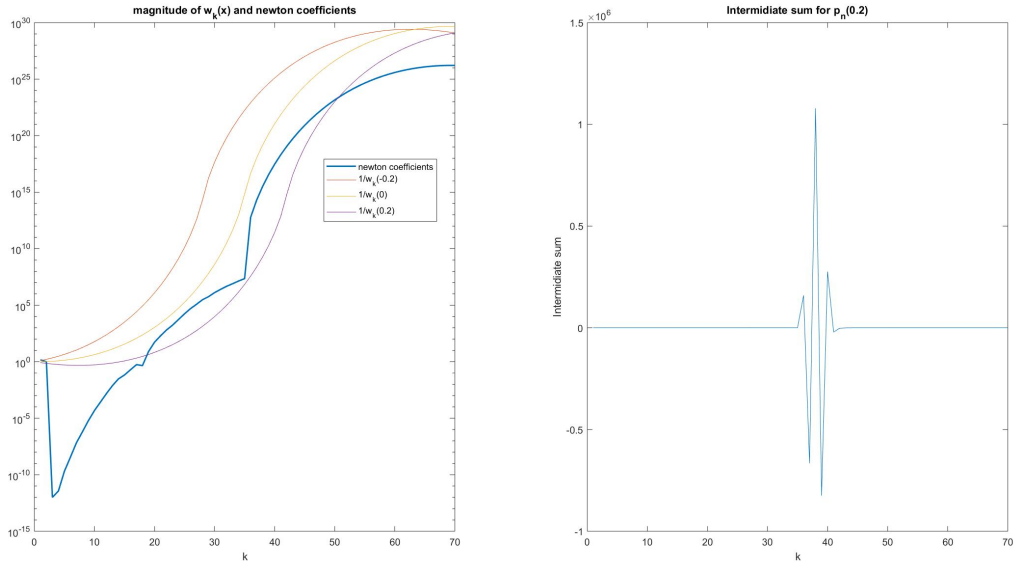
For Barycentric form 2 with uniform mesh strategy, a number of points have significant interpolation error and the error is relatively large near the end points. When the degree increase from 50 to 70, the maximum error decreases from about 10^2 to below 10, but the overall magnitude of error is not decreased.

For Barycentric form 2 with Chebyshev points, the interpolating polynomial converge very well. In both Chebyshev points, the maximum error is achieved at 0, because the target function is not differentiable at 0 ,i.e. not smooth at near 0, using smooth global polynomial strategy is slow and difficult to eliminate the error at the break point.

The Chebyshev points of first kind do not include the end points, and the information of the target function values at the end point is not accessed in the interpolating process. So the Chebyshev points of first kind have the error increased at the end points and also the error have larger range of fluctuations near the end point. The Chebyshev points of second kind, basically have monotonic decreasing error $p_n(x)$ is evaluated away from 0.

For Newton divided difference form, when degree is 50, the $p_n(x)$ is still around the target function near 0, but when the degree increase to 70, $p_n(x)$ only close to the target function when x is negative and also close to 0, which accord the Runge's phenomenon discussed in the lecture that the interpolation polynomial diverge at the end points of the interval. For degree 70, the results seems not follow Runge's phenomenon exactly, as nearly all x above 0 diverges and I will dig into that matter. The following figure shows the magnitude of the Newton divided coefficients, $w_k(x)$ k=1,...,n and intermediate sum for evaluating $p_n(x)$

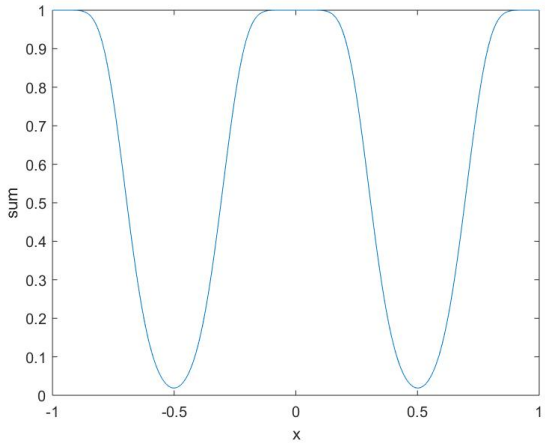
Figure 13: Intermediate result of Newton divided difference form $p_{70}(x)$ (degree of 70) (function1)



Note that in the figure I plotted the magnitude of $1/w_k(x)$ to compare the magnitude with the coefficients. The left figure shows that when x increase from -0.2 to 0.2, the magnitude of $w_k(x), k = 1, ..., 70$ increases. When x = -0.2, the magnitude of $w_k(x)$ is small enough to cancel the magnitude of the Newton coefficients and magnitude of the intermediate sum stay small. When x = 0.2, the magnitude of $w_k(x)$ become too big to cancel the magnitude of the Newton coefficients, thus the intermediate sum blows up, which result in severe cancellation and round off error. In the right figure, we can see that the intermediate sum grow up to 10^6 and then go back to normal size number and this process generate huge round off error and numerical instability, which also accord with the previous stability analysis.

For Bernstein polynomial, the error grow up when $p_n(x)$ is evaluated away from the end points -1, 0 and 1. The value of Bernstein polynomial goes to 0 around -0.5 and 0.5 when degree is high, which produce error around -0.5 and 0.5. Since the target function is close to 0 at -0.5, error near -0.5 is smaller that the error at 0.5. The following plot shows the value of $\sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}$ for $x \in [-1, 1]$.

Figure 14: $\sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}$ for $x \in [-1, 1]$ of $B_{70}(x)$ (degree of 70)



Note that $\sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}$ show be identically 1 for $x \in [-1, 1]$ in exact sum. As the weights goes to 0 near -0.5 and 0.5, the function value $B_n(x)$ will also goes to 0 near -0.5 and 0.5. Since the $B_n(x)$ work well for x near -1, 0 and 1, we binomial coefficients and target function work correctly. Thus the problem lie in the terms $x^k(1-x)^{n-k}$ for $k = 0, ..., n$. When x = 0.5, the term $x^k(1-x)^{n-k}$ is identically small for all k if n is large. The magnitude of $x^k(1-x)^{n-k}$ is about 10^{-22} and the largest magnitude for $\binom{n}{k}$ is about 10^{20}. So similarly, severe cancellation happens in the $\sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}$, which is the main source of the error.

Function 2: ($f_2(x) = \frac{1}{1+x^2}$)

Figure 15: $p_{50}(x)$ and true function value (degree of 50) (function2)

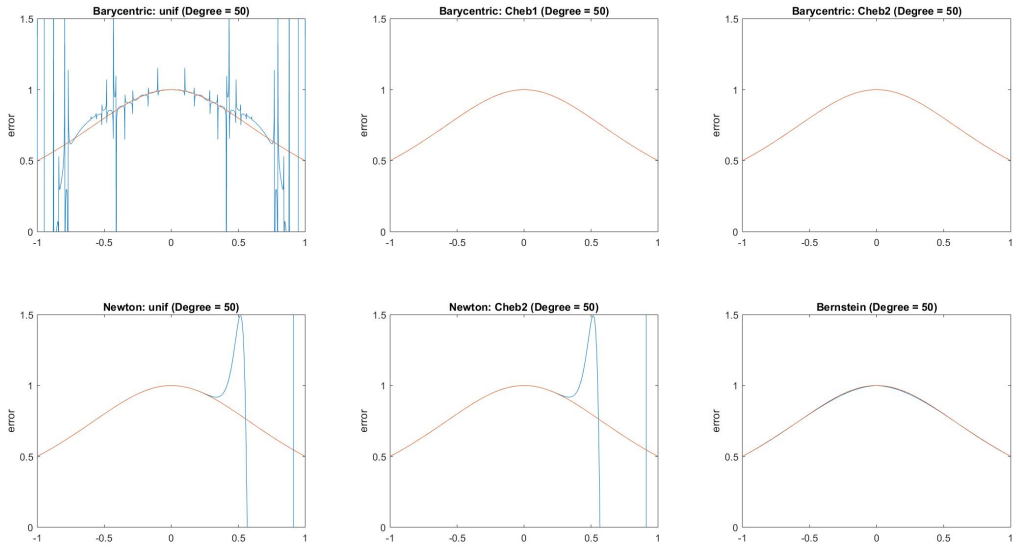


Figure 16: Interpolation error(log scale) of $p_{50}(x)$ (degree of 50) (function2)

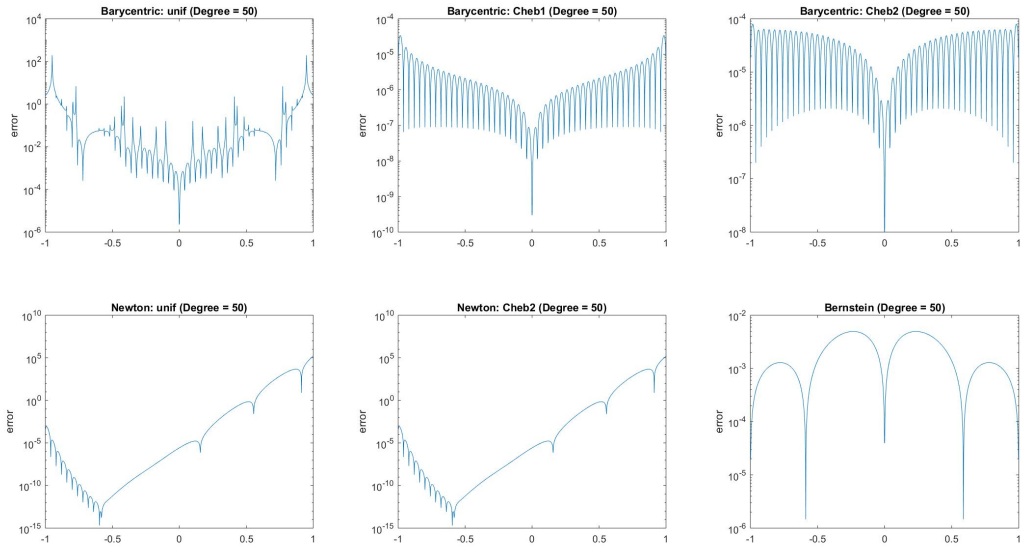


Figure 17: $p_{70}(x)$ and true function value (degree of 70) (function2)

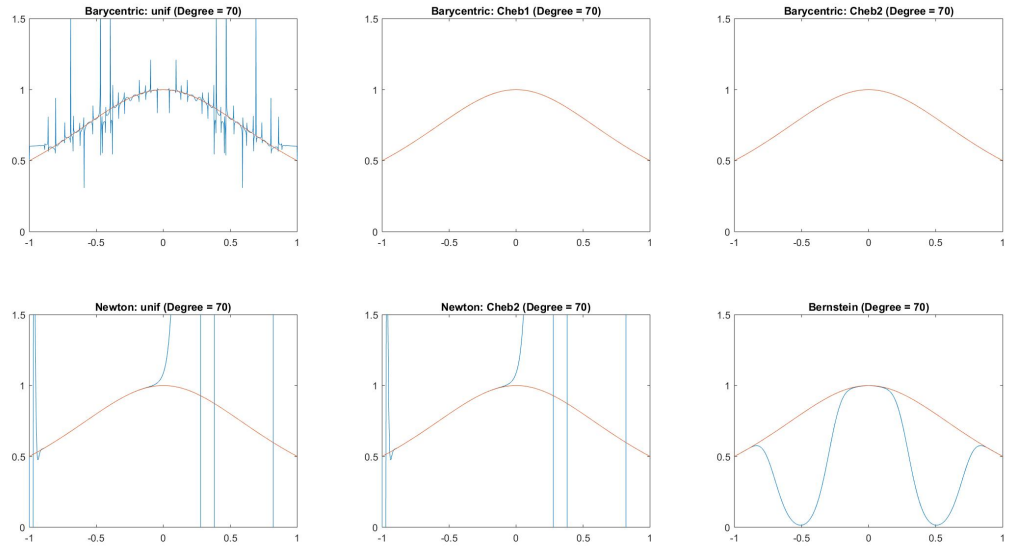
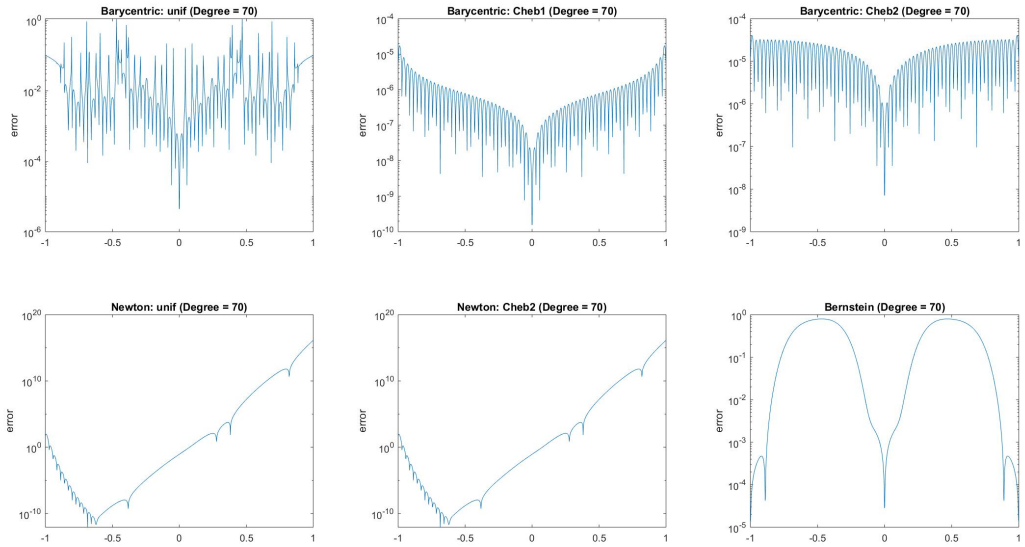


Figure 18: Interpolation error(log scale) of $p_{70}(x)$ (degree of 70) (function2)



For Bartcentric form 2, now the error mainly mainly exist in the end points which accord with the Runge's phenomenon. Similarly, the Chebyshev points of first kind have more error near the end points compared with the Chebyshev points of second kind. And other behaviours are similar with the behaviour discussed above, e.g. the divergence of Newton divided difference form for when x increase from -1 to 1 and Bernstein polynomial converge to 0 near -0.5 and 0.5 when degree is high. However, some of the above numerical instability can be improved by other summation strategies like the sorted sum.

5.3 Conditioning analysis

I define my perturbed functions as the following:

$$\tilde{f}_1(x) = |x| + \frac{x}{2} - x^2 + M * sin(100x) \tag{28}$$

and

$$\tilde{f}_2(x) = \frac{1}{1+x^2} + M * sin(100x) \tag{29}$$

where M is the perturbation size. In the following result, I choose the value of M to be 1 and 10 times of the machine epsilon in single precision(10^{-7}) to test the conditioning. By this setup, the change in the input $f(x_i) - \tilde{f}(x)$ is bounded by M, and I also use the infinity norm to measure the difference in $p_n(x)$ and $\tilde{p}_n(x)$.

Figure 19: Perturbation: 1 machine epsilon(float)

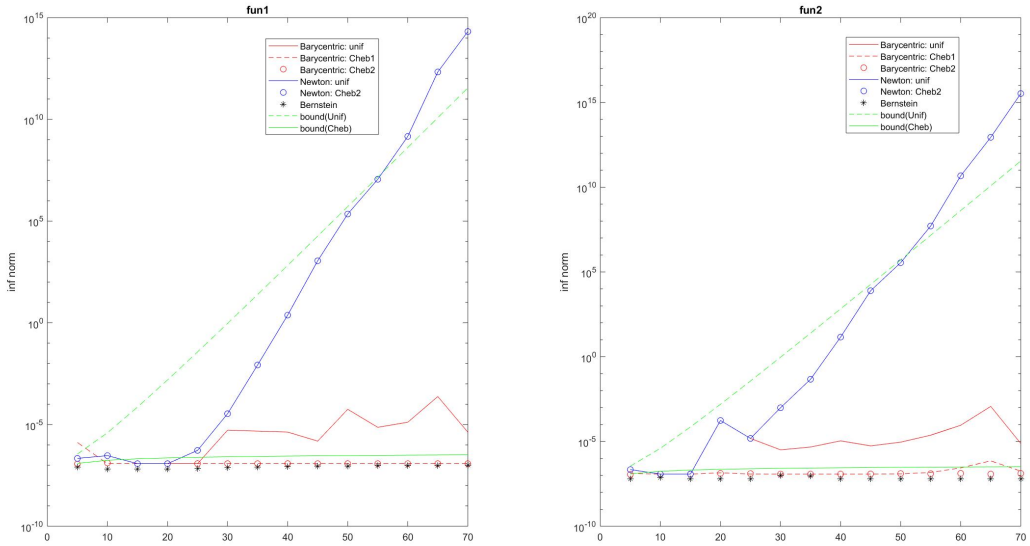
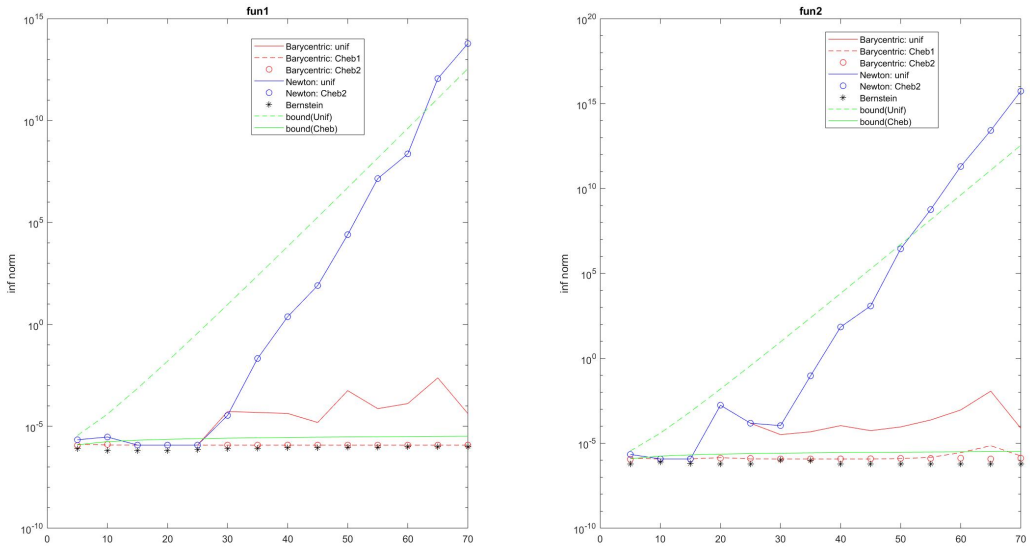


Figure 20: Perturbation: 10 machine epsilon(float)



First of all, the conditioning for Newton divided difference form is not good when degree is large as a small change in the input could result in huge change in the output. But, the change in output is still within the upper bound defined by the Lebesgue constant s.t. $\Lambda_n(X) \approx \frac{2^{n+1}}{enlog(n)}$. And the value outside of the bound might come from the instability of the algorithm which have been showed earlier. Bartcentric form 2 with uniform mesh is also not ideal as the difference in result is about 100 time bigger than the error of other methods. Also is is in the upper bound for equally spaced mesh.

For Bernstein polynomial and Bartcentric form 2 with Chebyshev points, which has stable condition number across different degree, about 10^{-7} perturbation in the input could have 10^{-7} change in the output, and similarly, 10^{-6} perturbation in the input result in 10^{-6} change in the output. So the actual condition number for these three interpolation form is close to 1, which is also in within the bound for Chebyshev points defined by $\Lambda_n(X) \approx \frac{2}{\pi} log(n)$.

The conditioning for Newton divided difference form and Bartcentric form 2 with uniform mesh have similar behaviour in conditioning when degree is not too high. Moreover, for function 2, the condition number deteriorate earlier in degree that function 1 for this two methods.

6 Conclusions

In terms of number of operations, Barycentric form 2 need $O(n)$ operations for pre-processing and $O(n)$ operations to evaluate,the Newton divided difference form require $O(n^2)$ for pre-processing and $O(n)$ operations to evaluate, and finally the Bernstein polynomial take $O(n)$ for pre-processing and $O(n)$ operations to evaluate.

For stability, Barycentric form 2 with Chebyshev points is stable even when the degree is large. Barycentric form 2 with uniform mesh is stable for most of the points, but a small number of unstable values when degree is large. Bernstein polynomial is stable in general, but is become unstable at the end points when degree is large. Newton divided difference is very unstable when degree is large.

For convergence, Barycentric form 2 with Chebyshev points and Bernstein polynomial will converge when degree goes up, but when degree is too big (above 55)the Bernstein polynomial

will begin to diverge. Newton divided difference and Barycentric form 2 with uniform points begin to diverge when the degree is above 10 in function 1 and 25 in function 2, and Newton divided difference diverge faster while the divergence of Barycentric form 2 with uniform points does not grow significantly.

For the convergence rate in degree such that all the algorithms converge, if the target function is smooth, Bernstein polynomial is the slowest and if the target function is not smooth they have similar rate of convergence i.e. all other algorithm are slowed down due to the break points.

For conditioning, Barycentric form 2 with Chebyshev points and Bernstein polynomial are nearly perfectly conditioned for this 2 target functions. Barycentric form 2 with uniform points is well conditioned when degree is small, and the condition number deteriorate when degree increase but the worse case is about 1000. The condition number for Newton divided difference is good when degree is small, when degree goes up, the condition number grow exponentially.

7 Program Files

Compile c++ code first to generate all the .csv files for analysis, and then use the Matlab code the do the plot.

References

[1] H. R. Schwarz, *Numerische Mathematik*, 4th ed., Teubner, Stuttgart, 1997; English translation of the 2nd edition, *Numerical Analysis: A Comprehensive Introduction*, Wiley, New York, 1989.

[2] P. Henrici, *Essentials of Numerical Analysis*, Wiley, New York, 1982.

[3] H. E. Salzer, *Lagrangian interpolation at the Chebyshev points xn,,= cos(/n), = 0(1)n; some unnoted advantages*, Comput. J., 15 (1972), pp. 156159.

[4] Quarteroni, A., Sacco, R., & Saleri, F. (2010). Numerical mathematics (Vol. 37). Springer Science Business Media.

[5] Smoktunowicz, A., Wrbel, I., Kosowski, P. (2007). *A new efficient algorithm for polynomial interpolation*. Computing, 79(1), 33-52.

[6] Bartle, Robert Gardner, and Robert G. Bartle. *The elements of real analysis*. Vol. 2. New York: Wiley, 1964.