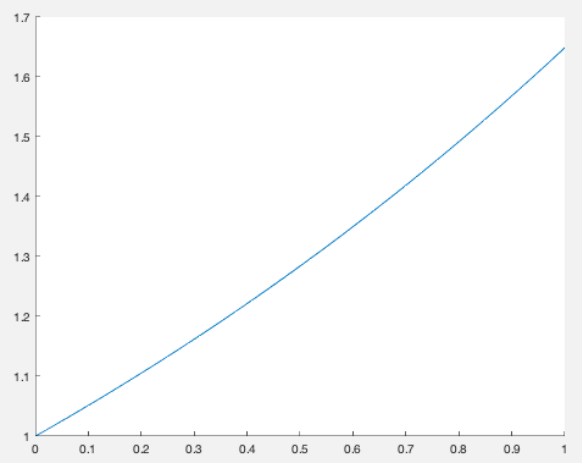
Corey Schulz
CS 3200 – Spring 2020
January 19, 2020

Homework 1 Report

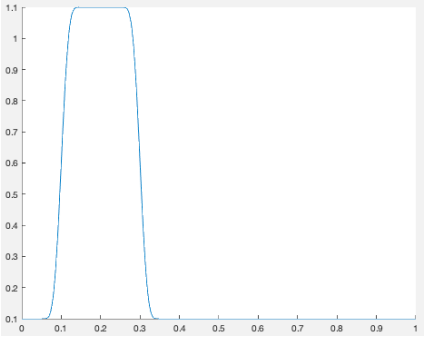
Question 1

I plotted the exponential function on the interval [0, 1] and it looks as follows:

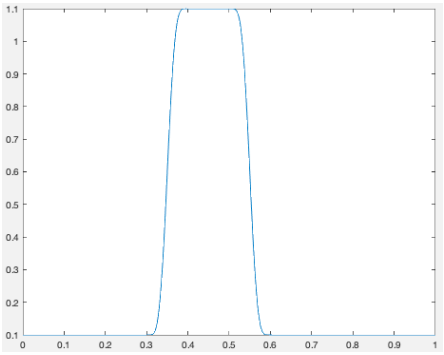


Note that you can't see too much curvature given the limited range of the window.

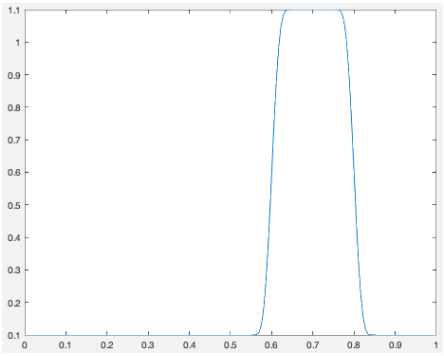
The Hubbard function, meanwhile, can be plotted with varying time values as such:



Time = 0.25



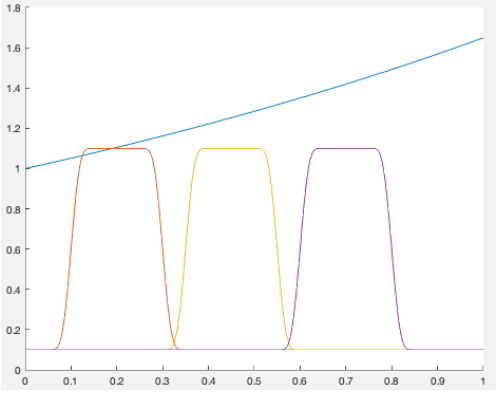
Time = 0.5



Time = 0.75

The time value is an offset to where the Hubbard function starts. The greater the time value, the greater the shift on the X axis the function takes. This shift scales linearly with time.

Finally, here's the exponential function and Hubbard all on one plot:



Question 2

Starting off with this question, I used Matlab's built-in `linspace` function to get 1001 point values. I used the Hubbard function on those points to get a 1001 point approximation of the Hubbard function to use as a baseline for the rest of the problem. With a polynomial of variable degree, I calculated Hubbard using the polyinterp method. Finally, I calculated the L1, L2 and L Infinity errors as follows:

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	2.202691e+03	9.568528e+03	6.819600e+04
36	1.644392e+05	8.185172e+05	5.970239e+06
44	5.090175e+06	2.876898e+07	2.536539e+08
52	5.094791e+08	3.221990e+09	3.302924e+10
60	2.495564e+10	1.694086e+11	1.746184e+12

Equal Spacing Interpolation error

In general, the accuracy with polynomial interpolation over polynomials of varying degree gets worse as the degree increases. As the polynomial grows sufficiently, the error becomes greater – on a different order of magnitude than a polynomial of a lesser degree. However, even here, the error is much greater than I expected to see from the interpolation function. The errors here are simply too much to reasonably graph, but they *are* visible in this table and in the graphs at the end of this question.

Doing the same data-collection method with the Chebyshev points, derived from the equation in the assignment, produces the following error data:

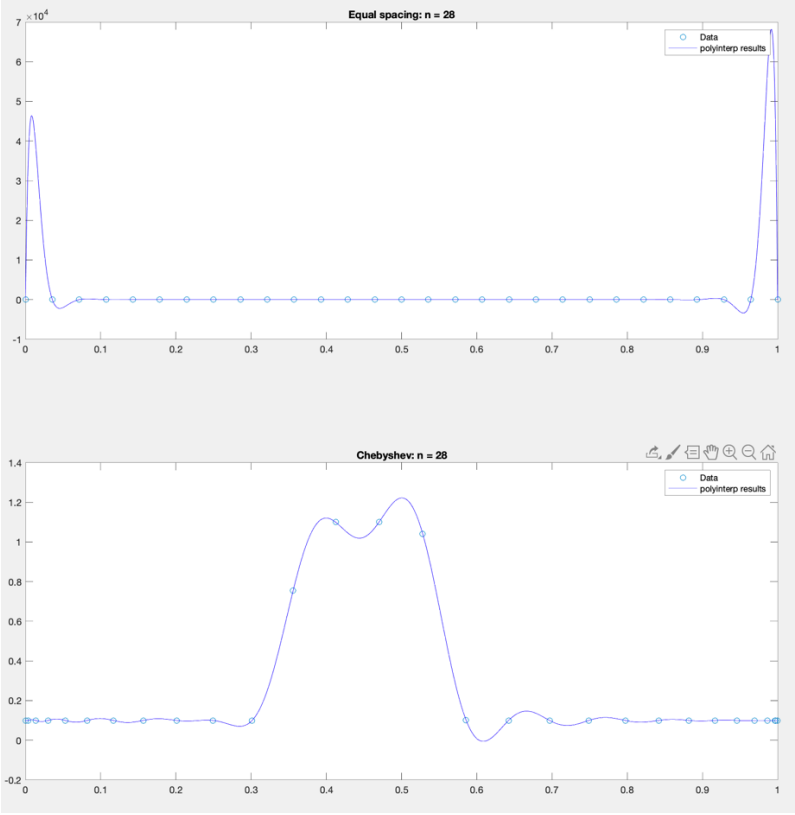
Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	2.793633e-02	4.822941e-02	1.811896e-01
36	2.496597e-02	4.480548e-02	1.776224e-01
44	1.068216e-02	1.798399e-02	8.134253e-02
52	8.331598e-03	1.290606e-02	5.111712e-02

60	2.840626e-03	4.589178e-03	1.401506e-02

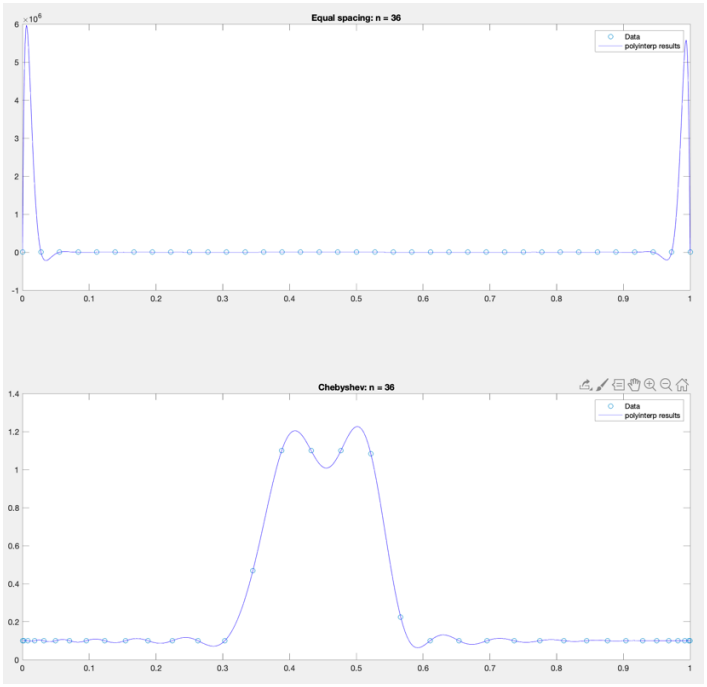
Chebyshev Interpolation error

The errors produced from using the Chebyshev points as a baseline were drastically lower – many orders of magnitude lower (and all less than one!) – than the errors produced from the equal spacing polyinterp baseline above. Here, as the polynomial degree increases, the errors actually trend *lower*, not higher which is another key difference from equal spacing.

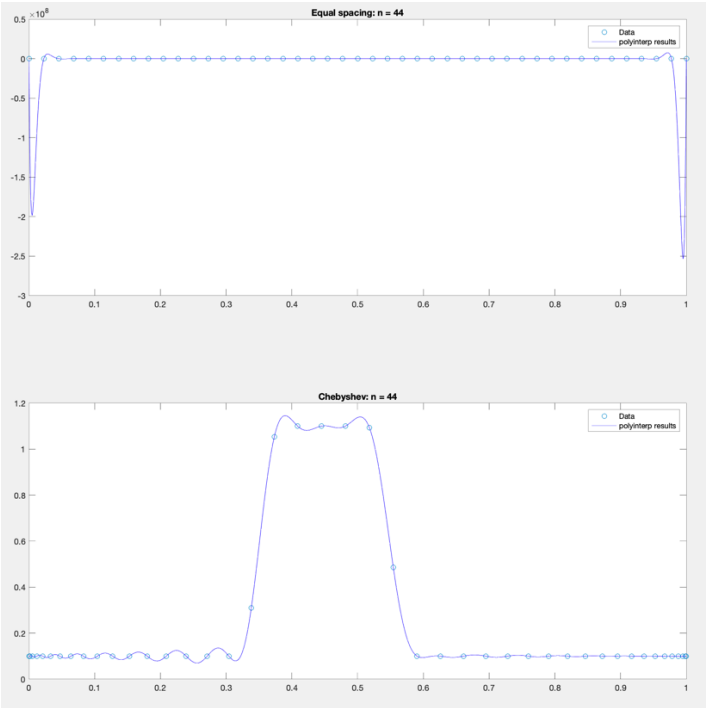
Below are images of the data points for each case, plotted along the polyinterp data. The equal spacing points and data are on top, while the Chebyshev points and data are on the bottom of each image:



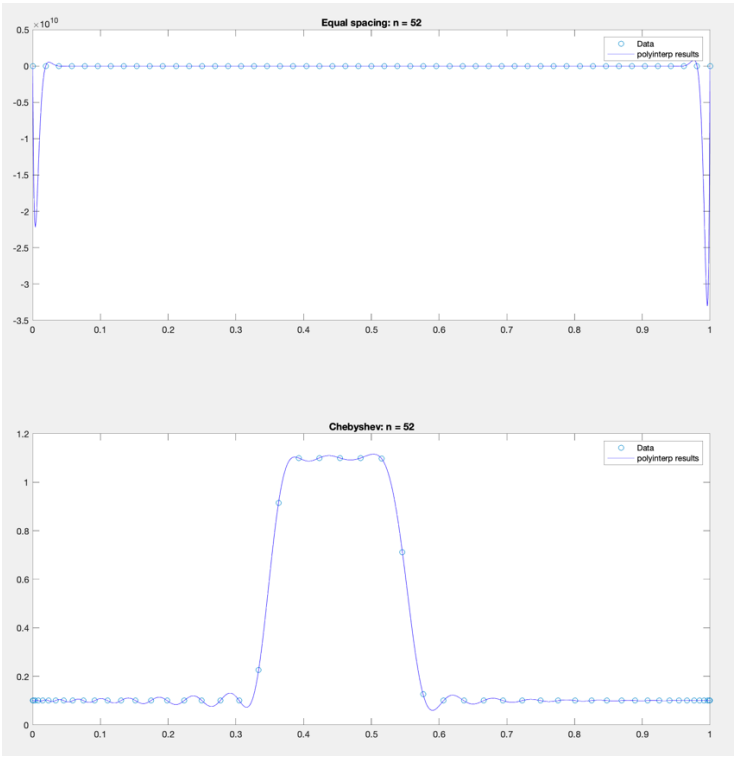
N = 28


Equal spacing: n = 36
Data
polyinterp results
Chebyshev: n = 36
Data
polyinterp results

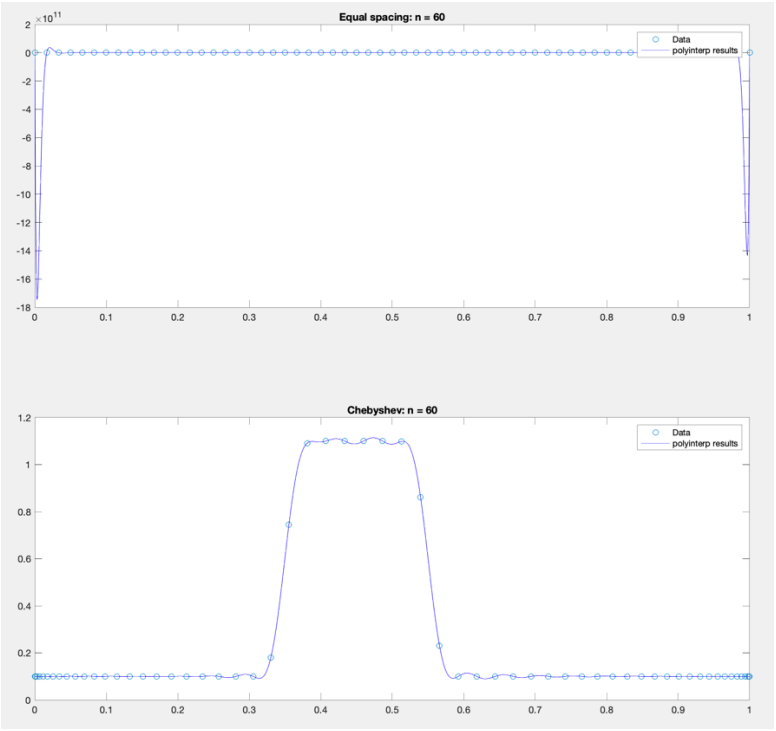
N = 36


Equal spacing: n = 44
Data
polyinterp results
Chebyshev: n = 44
Data
polyinterp results

N = 44


Equal spacing: n = 52
×10^10
Data
polyinterp results
Chebyshev: n = 52
Data
polyinterp results

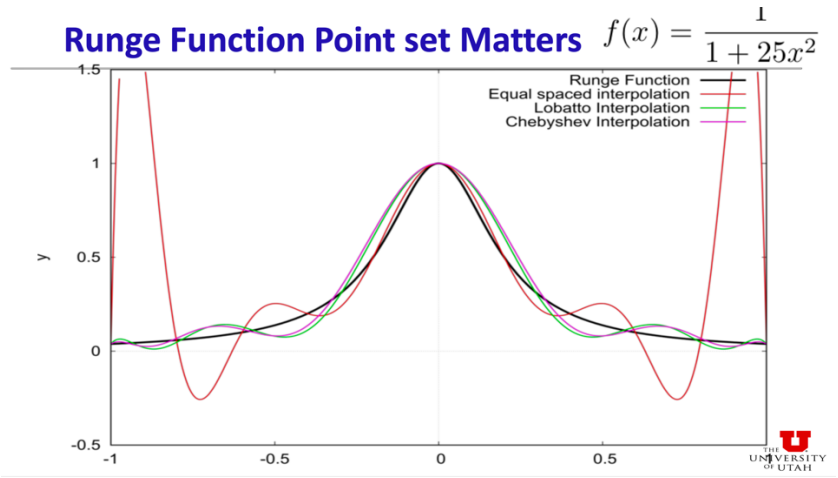
N = 52


Equal spacing: n = 60
×10^11
Data
polyinterp results
Chebyshev: n = 60
Data
polyinterp results

N = 60

In the case of the equally spaced points, they are completely off-base in all scenarios which is reasonable considering the large errors for this method. However, the Chebyshev points fared much better, increasingly approximating the Hubbard function more correctly the more polynomial degrees that are given.

The results here match that of the Runge example:



…With the equal spaced interpolation exploding on the ends, and the Chebyshev points getting closer to the objective function.

Question 3

Modifying my script from Question 2, I changed it to work on the Gelb Tanner function. Using *most* of the same code as before, I first found the errors for the equal spacing interpolation.

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	5.699925e+01	1.986716e+02	1.164863e+03
36	3.756142e+03	1.511017e+04	1.021848e+05
44	2.308951e+05	1.042506e+06	7.900500e+06
52	1.445461e+07	7.188209e+07	5.993082e+08

60	9.329619e+08	5.041385e+09	4.533108e+10

Equal Spacing Interpolation error

Here, as in Question 2, the error rises by degrees of magnitude as the polynomial degree is increased. Like in the Runge example in class, the error blows up at the ends of the function – so much so that you can't even see the original approximation in the graphs because of it.

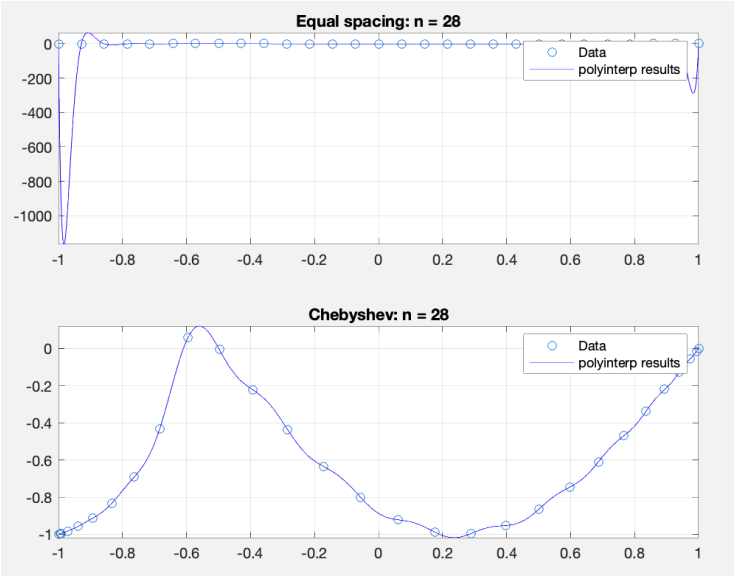
I also calculated the errors for the Chebyshev points, again like in Question 1. This time, the only real change required was making the points over the interval [-1, 1] instead of [0, 1]. Here are the errors reported for Chebyshev:

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	5.921873e-02	1.508701e-01	9.654801e-01
36	6.546557e-02	1.031836e-01	7.666819e-01
44	7.427252e-02	1.003205e-01	5.967959e-01
52	4.963277e-02	1.207002e-01	9.506042e-01
60	4.647436e-02	7.846171e-02	6.982986e-01

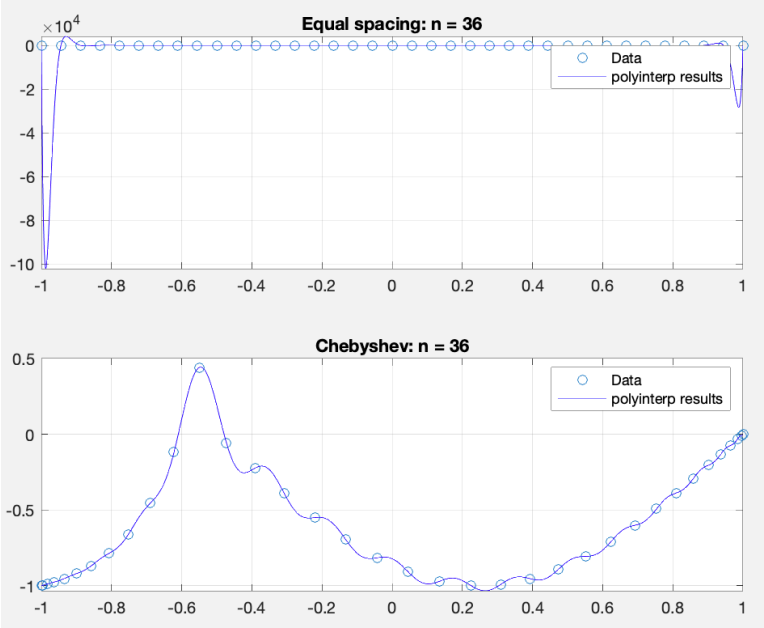
Chebyshev Interpolation error

This time, the errors for the Chebyshev interpolation remained roughly in the same range, with the L2 error remaining about the same each time, but the L1 and L-Inf errors fluctuating about roughly the same point. All errors remained on the same order of magnitude. The Chebyshev points remain clear winners of low error, far exceeding the accuracy of equal spaced points.

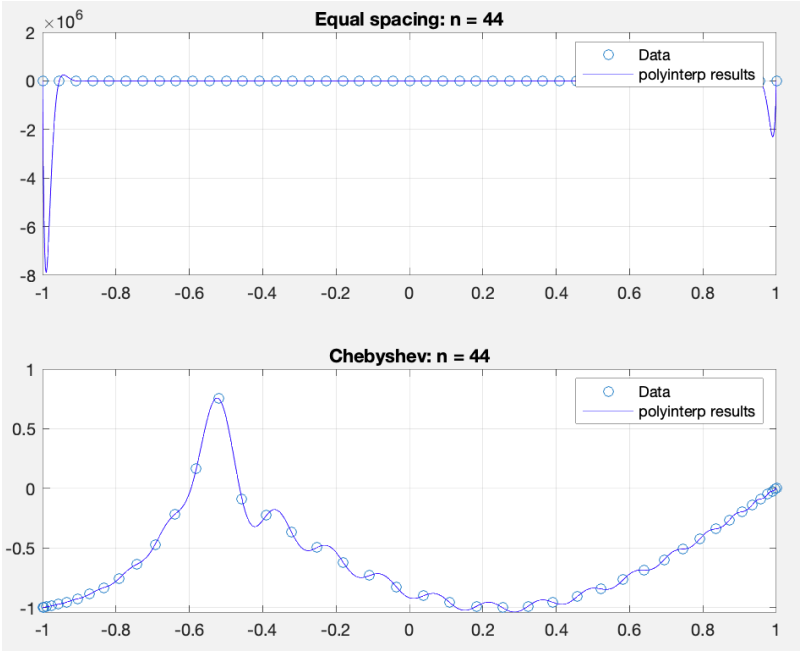
Finally, below are images of the data points for each case, plotted along the polyinterp data. The equal spacing points and data are on top, while the Chebyshev points and data are on the bottom of each image:


Equal spacing: n = 28
Data
polyinterp results
Chebyshev: n = 28
Data
polyinterp results

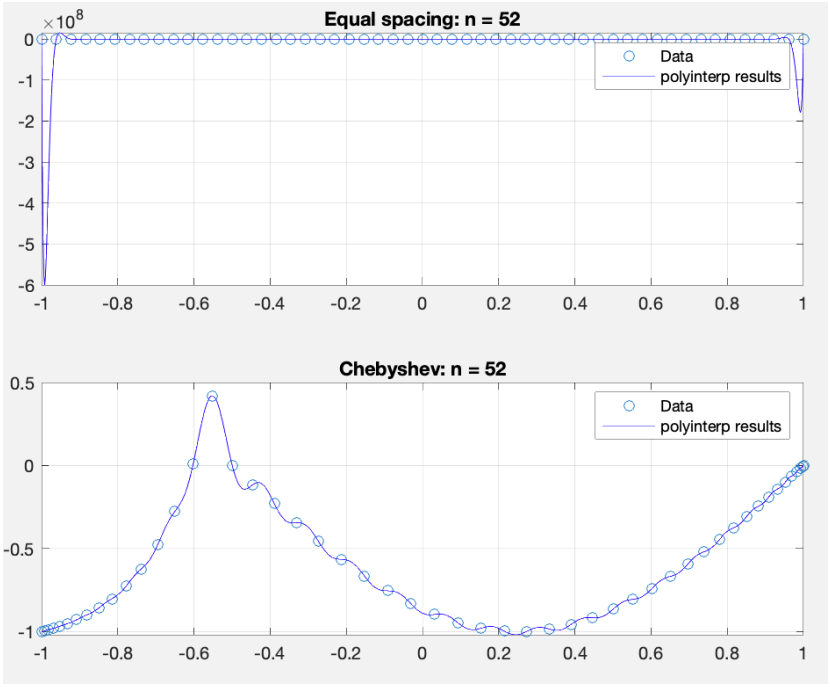
N = 28


Equal spacing: n = 36
×10^4
Data
polyinterp results
Chebyshev: n = 36
Data
polyinterp results

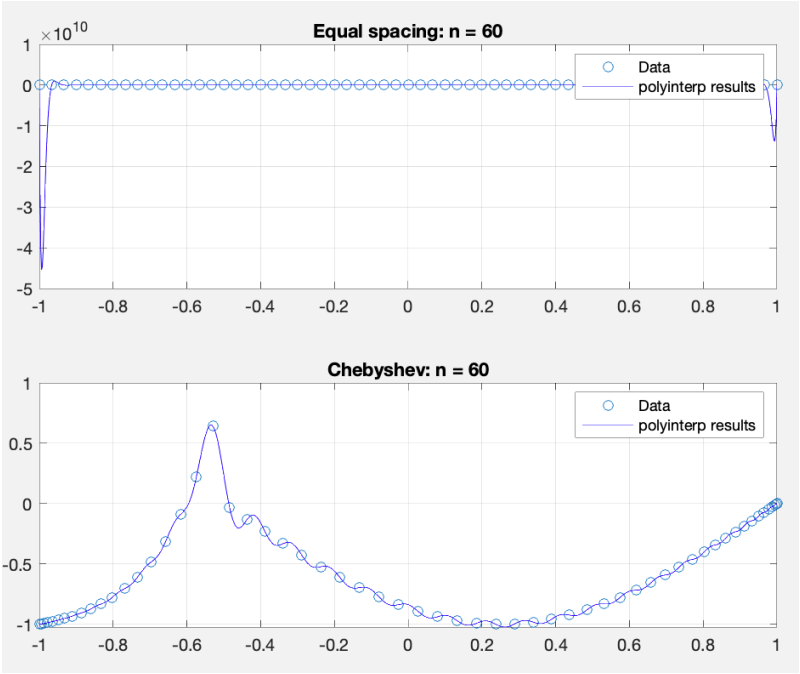
N = 36


Equal spacing: n = 44
×10^6
2
0
-2
-4
-6
-8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Data
polyinterp results
Chebyshev: n = 44
1
0.5
0
-0.5
-1
Data
polyinterp results

N = 44


Equal spacing: n = 52
×10^8
0
-1
-2
-3
-4
-5
-6
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Data
polyinterp results
Chebyshev: n = 52
0.5
0
-0.5
-1
Data
polyinterp results

N = 52



N = 60

Here, again, the Chebyshev points better define the Gelb Tanner function the larger N is. However, the error remains roughly equivalent. But, again like in Question 2, the equal spacing method does not come near approximating the Gelb Tanner function. The Chebyshev method is the clear winner.

Question 4

Finally, modifying roughly the same code as from the last two questions, this time I used Matlab's builtin `pchip` and `spline` functions and calculated the interpolation error for both the Hubbard and Gelb Tanner functions, as noted below.

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	5.544406e-03	1.639597e-02	7.994698e-02
36	3.039971e-03	9.081455e-03	4.641342e-02
44	1.732534e-03	5.238436e-03	2.592760e-02

52	1.060150e-03	3.150398e-03	1.574250e-02
60	7.037201e-04	2.110651e-03	1.009527e-02

PCHIP Interpolation Error: Hubbard

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	7.680448e-03	1.778022e-02	9.091054e-02
36	3.240715e-03	7.695324e-03	3.771522e-02
44	1.387949e-03	3.484214e-03	1.900067e-02
52	6.865046e-04	1.689338e-03	8.885648e-03
60	2.872988e-04	7.201683e-04	3.383913e-03

Spline Interpolation Error: Hubbard

In general, in the case of the Hubbard function, the spline interpolant beats out the PCHIP interpolant, but at some points the margin is rather narrow.

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	3.575458e-02	1.429931e-01	9.684368e-01
36	2.781968e-02	1.283722e-01	9.664743e-01
44	2.261858e-02	1.173082e-01	9.648037e-01
52	1.896955e-02	1.084838e-01	9.630169e-01
60	1.628238e-02	1.012210e-01	9.609920e-01

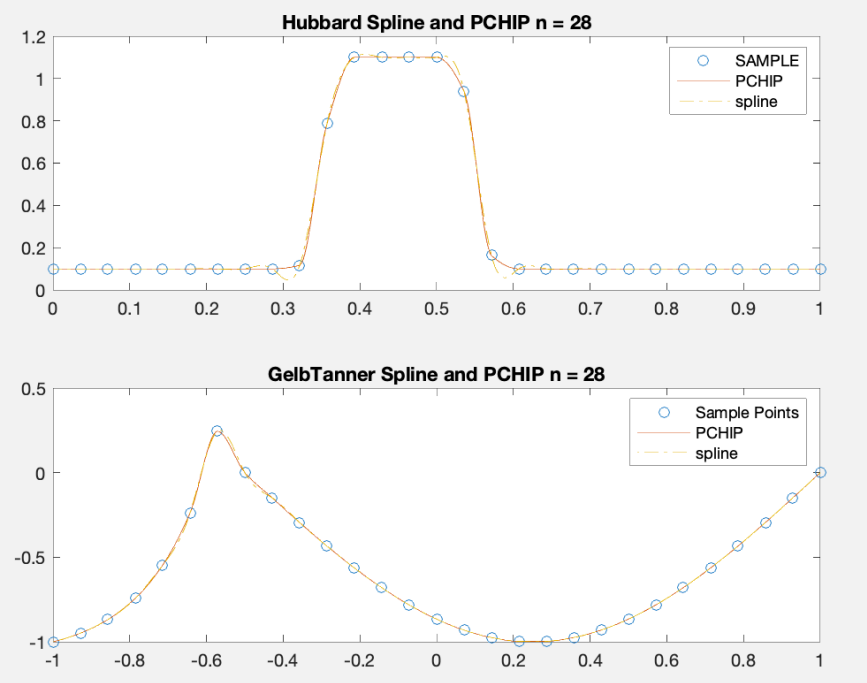
PCHIP Interpolation Error: Gelb Tanner

Polynomial Degree	L1 Error	L2 Error	L-Inf Error
28	3.659825e-02	1.386975e-01	9.653119e-01
36	3.016903e-02	1.250642e-01	9.602655e-01
44	2.551388e-02	1.144332e-01	9.549577e-01
52	2.202602e-02	1.058865e-01	9.494491e-01
60	1.933519e-02	9.883175e-02	9.437700e-01

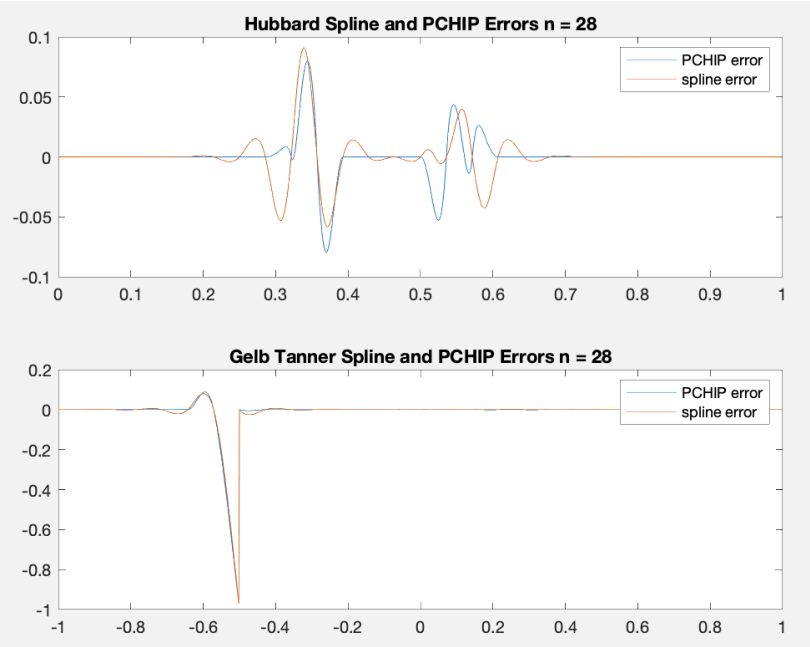
Spline Interpolation Error: Gelb Tanner

Unlike with the Hubbard function, with the Gelb Tanner function, the PCHIP interpolant narrowly beats out the spline interpolant. This is because “spline produces a smoother result, such that S″(x) is continuous. spline produces a more accurate result if the data consists of values of a smooth function. pchip has no overshoots and less oscillation if the data is not smooth,” (citation: MathWorks). Here, with these functions this is indeed the case and the differences in errors turned out as expected following this logic.

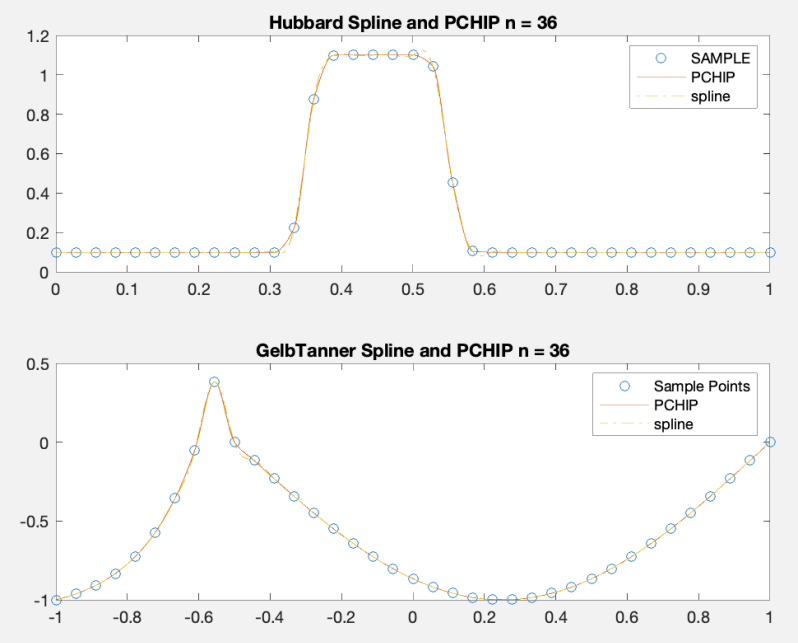
Visually graphing the functions, again as in the previous questions (and this time the errors, since they’re now meaningful to compare to each other!), yields the following:



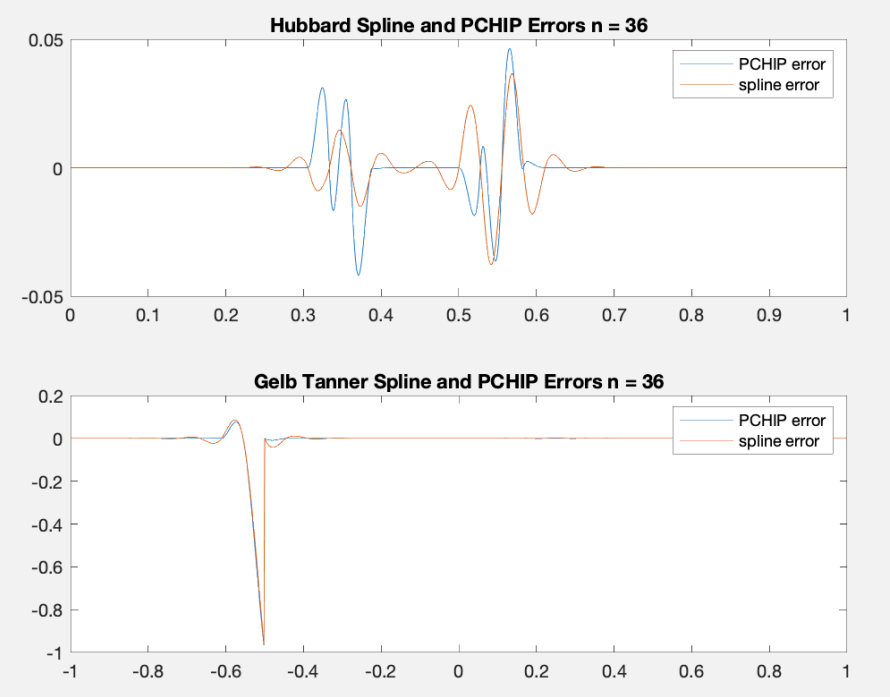
N = 28 ***Interpolants***



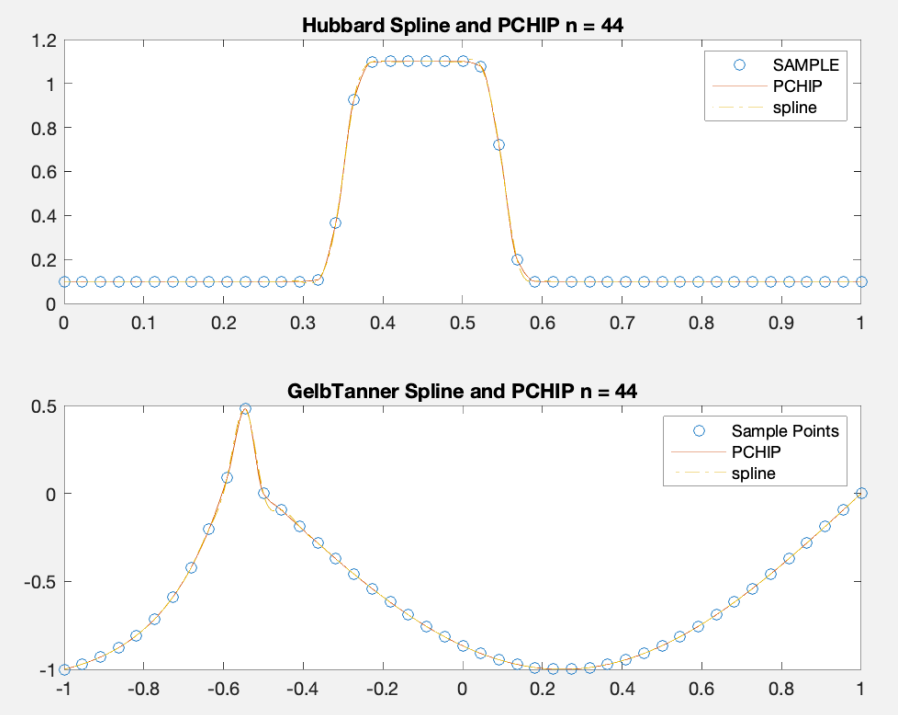
N = 28 ***Error***



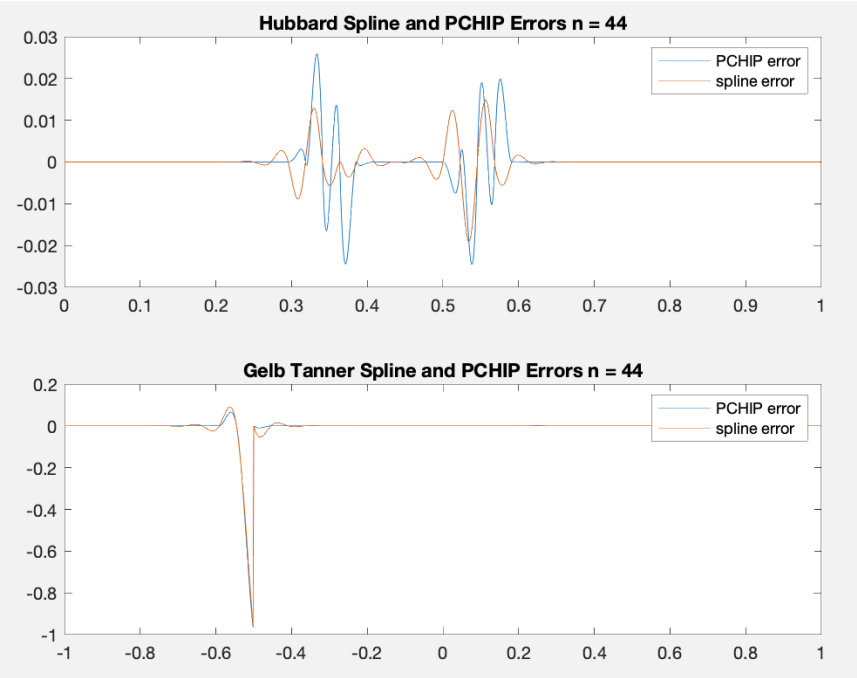
N = 36 ***Interpolants***



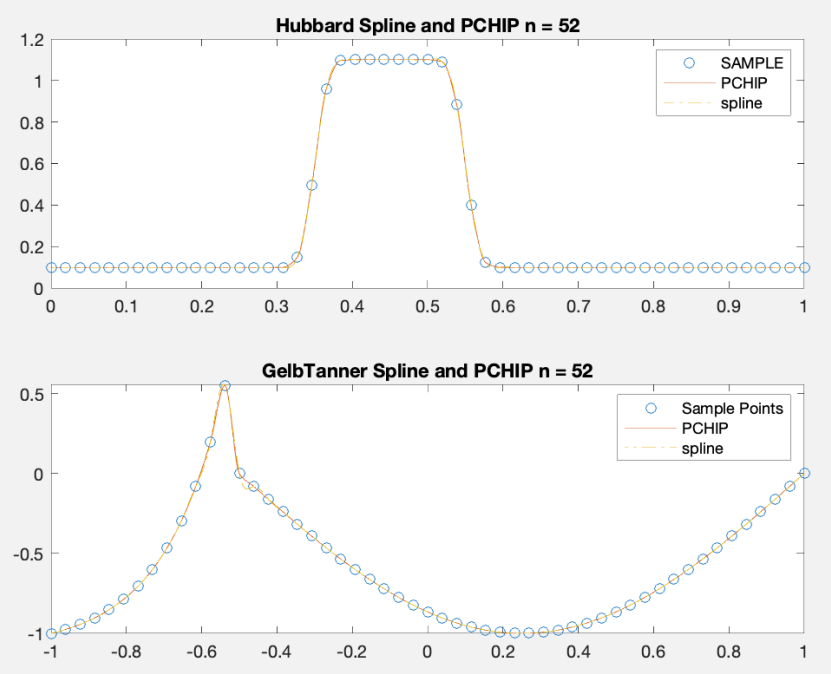
N = 36 ***Error***



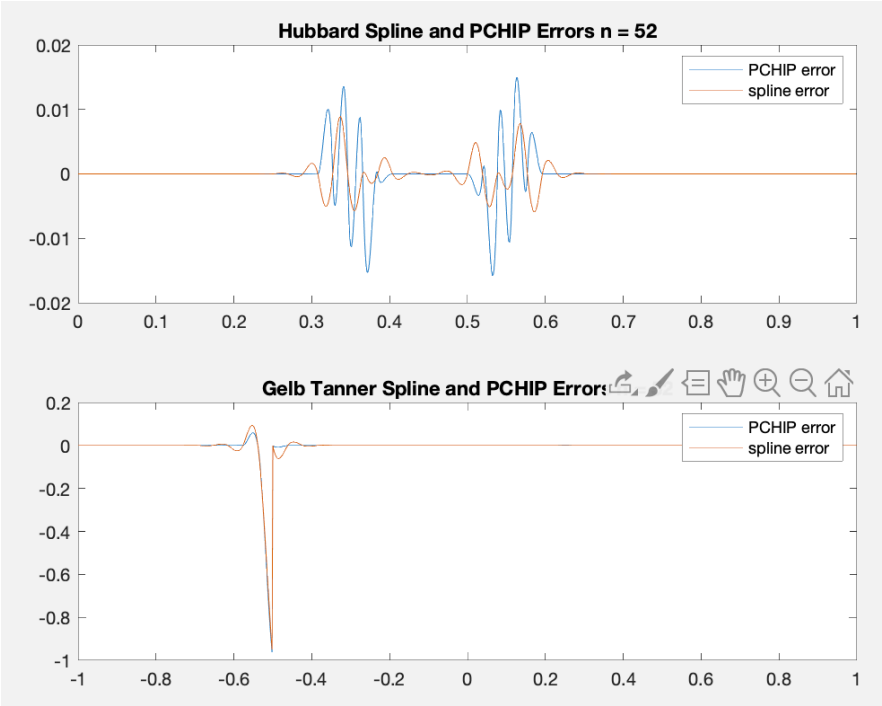
N = 44 ***Interpolants***



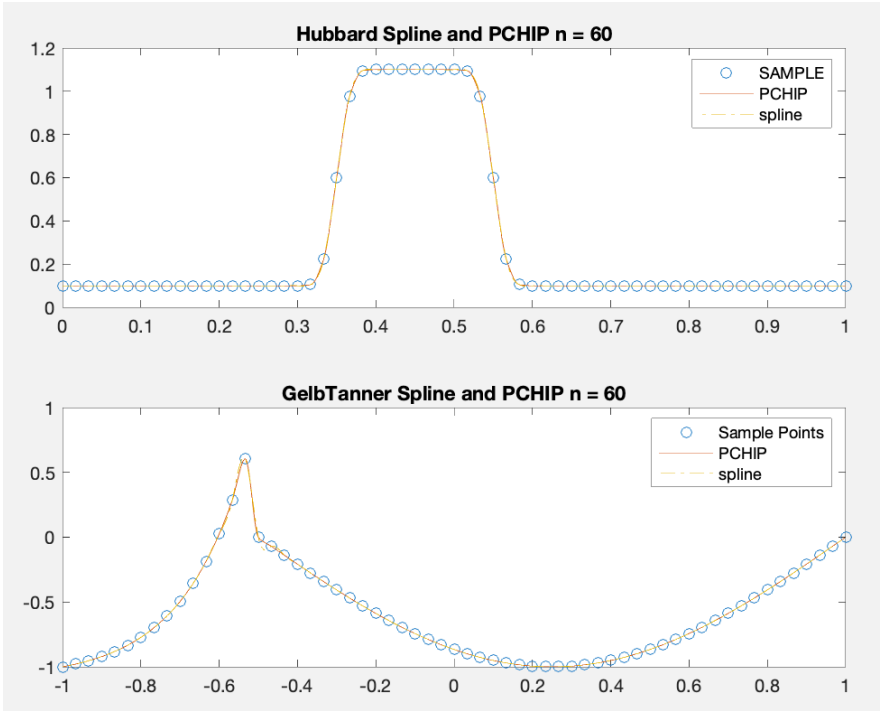
N = 44 ***Error***



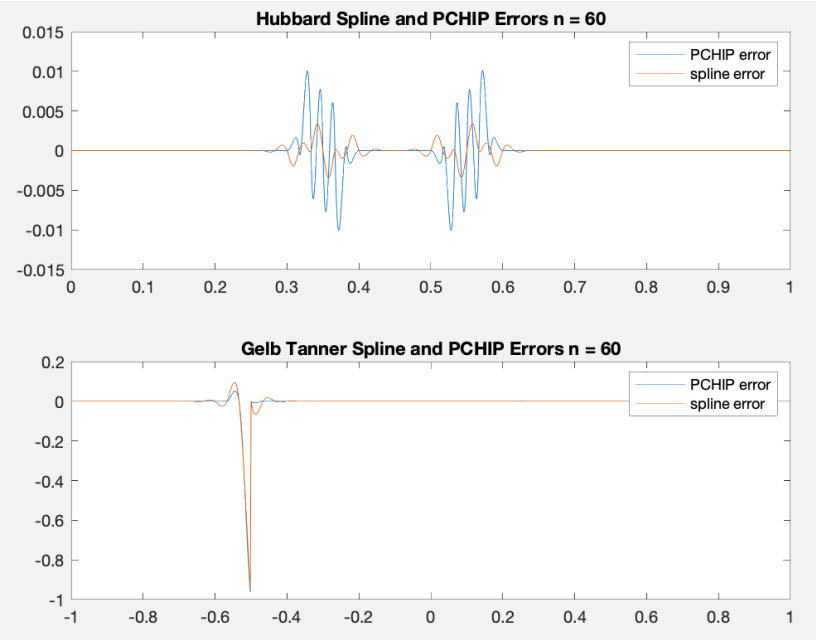
N = 52 ***Interpolants***



N = 52 ***Error***



N = 60 ***Interpolants***



N = 60 ***Error***

So, in all cases, both PCHIP and spline are very close together. Even the errors mimic each other to a large extent and it's a tossup which method will be better in which scenario. Notably, the larger the polynomial degree is, the larger the visible difference in the errors between PCHIP and spline are. However, PCHIP does have a computational advantage over spline, so it's best to use that when possible. Visually, the errors for each of the graphs appear similar. But there's a winner in each case. It's best, then, to experiment with different methods of interpolation to decide the best one. Except equidistant points. Don't use that one.

In general, however, Matlab's builtin PCHIP and spline methods outperform equally spaced points and Chebyshev, so I'd use those to do interpolation over the other methods discussed within the scope of this assignment.