CS3200 – Assignment 1 Written Report
Skylar Shyu

Question 1:

The Hubbard function was modified by adding a second argument, time. The default had hard-coded time to be 0.25, but that needed to be manipulated in this question. A line space was created over the domain of [0 1] and both Hubbard was plotted via trial and error with different values of t over the [0 1] domain. The exponential function was plotted only once over the same domain.

Once $t > 0$ for Hubbard, the points become discrete on the integers instead of remaining continuous between [0,1]. For $t = 0.75$, Hubbard has the steepest slope, while all other $0 <= t < 1$ is rather flat. The solution doesn't evolve smoothly, but rather discretely.

Question 2:

The Hubbard function was explored using five different polynomials of degree: 28, 36, 44, 52, and 60. For each degree, degree + 1 data points were generated over the space of [0 1], first using equidistant spacing, then with Chebyshev spacing. The x-values of these two sets of data points were fed into the Hubbard function (with $t = 0.25$, the hard-coded time in the version handed out) to generate two sets of y-values. The polyinterp routine was then used to create a heuristic vector over 1001 points over the space of [0 1].

To compute the errors for the polyinterp values generated, they were compared to the actual values produced by the Hubbard function over the same 1001 points in the space [0 1]; the Hubbard results were transposed to ensure that we didn't have skewed vector subtraction. L_1, L_2, and L_∞ were calculated using MatLab's $\mathbf{norm}(\boldsymbol{X}, \boldsymbol{p})$ method; X refers to the error matrix computed and p refers to the level of p-norm, which in this case was 1, 2, and ∞. Because $\mathbf{norm}(\boldsymbol{X}, \boldsymbol{p})$ returns a sum across the 1001 points, the average can be obtained by dividing the result by 1000-1.

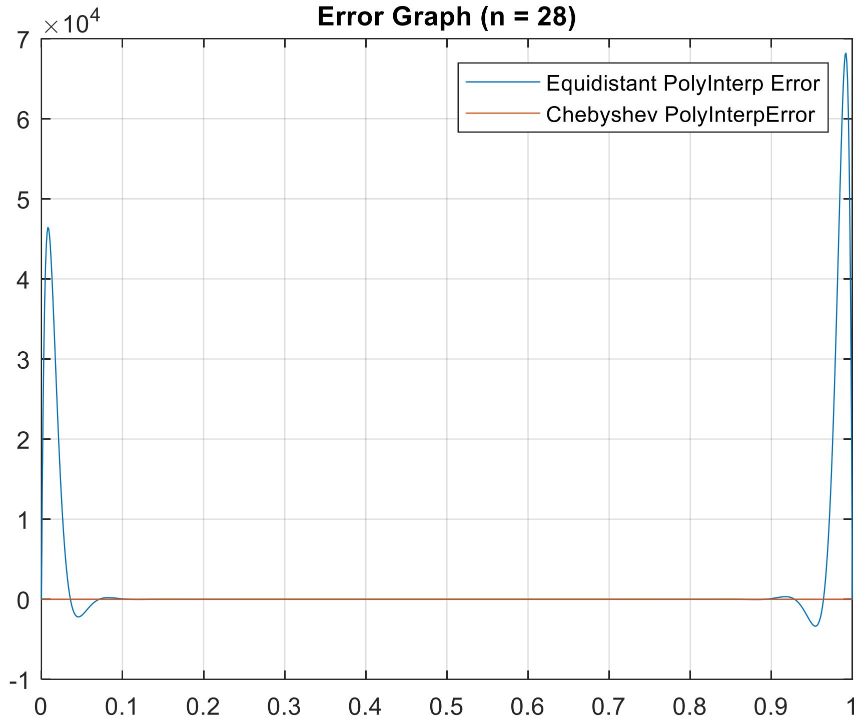
TABLE 1: p-norm Averages using Chebyshev and Equidistant Data Point Distributions with Respect to Polynomials of Degrees (n) for Hubbard Function Using Polyinterp Interpolation

	Equidistant Spacing			**Chebyshev Spacing**		
n	**L_1**	**L_2**	**L_∞**	**L_1**	**L_2**	**L_∞**
28	2.204896e+03	3.028863e+02	6.826427e+01	2.796430e-02	1.526675e-03	1.813709e-04
36	1.646038e+05	2.590970e+04	5.976216e+03	2.499096e-02	1.418292e-03	1.778002e-04
44	5.095270e+06	9.106656e+05	2.539078e+05	1.069285e-02	5.692730e-04	8.142396e-05
52	5.099891e+08	1.019903e+08	3.306230e+07	8.339938e-03	4.085341e-04	5.116829e-05
60	2.498062e+10	5.362533e+09	1.747932e+09	2.843469e-03	1.452678e-04	1.402909e-05

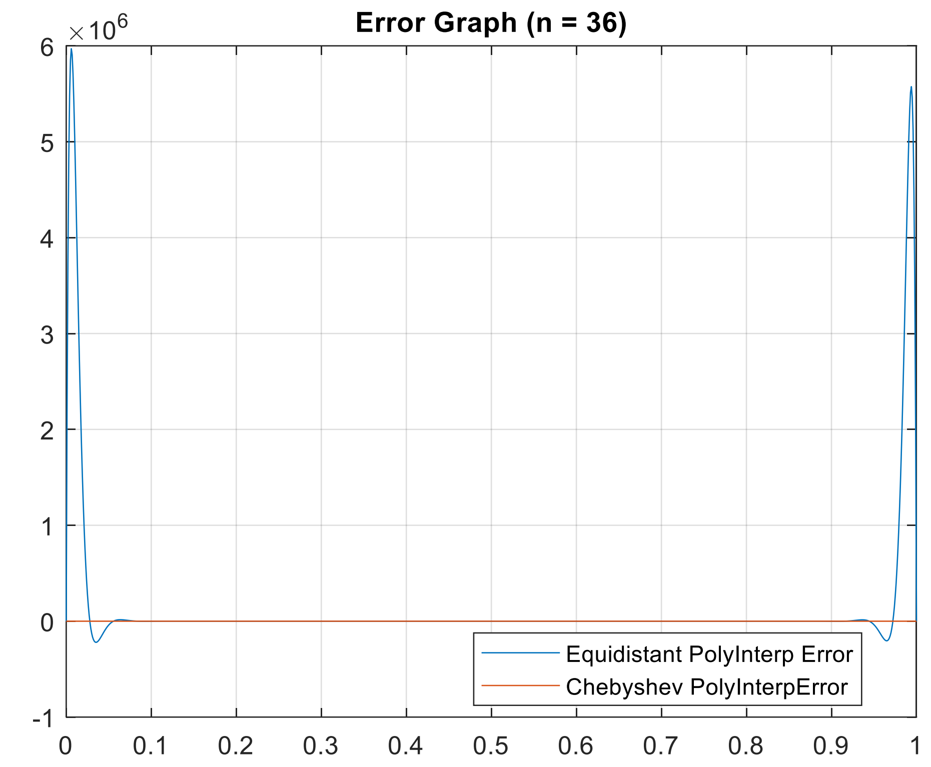
Using equidistant spaced data points, as the polynomial of degree increased, so did the error regardless of the p-norm; accuracy thus decreased as the polynomial of degree was increased. When the polynomial of degree was held constant, L_∞ was smaller than L_2, which in turn was less than L_1. Accuracy, then, increased as the *p*-norm increased.

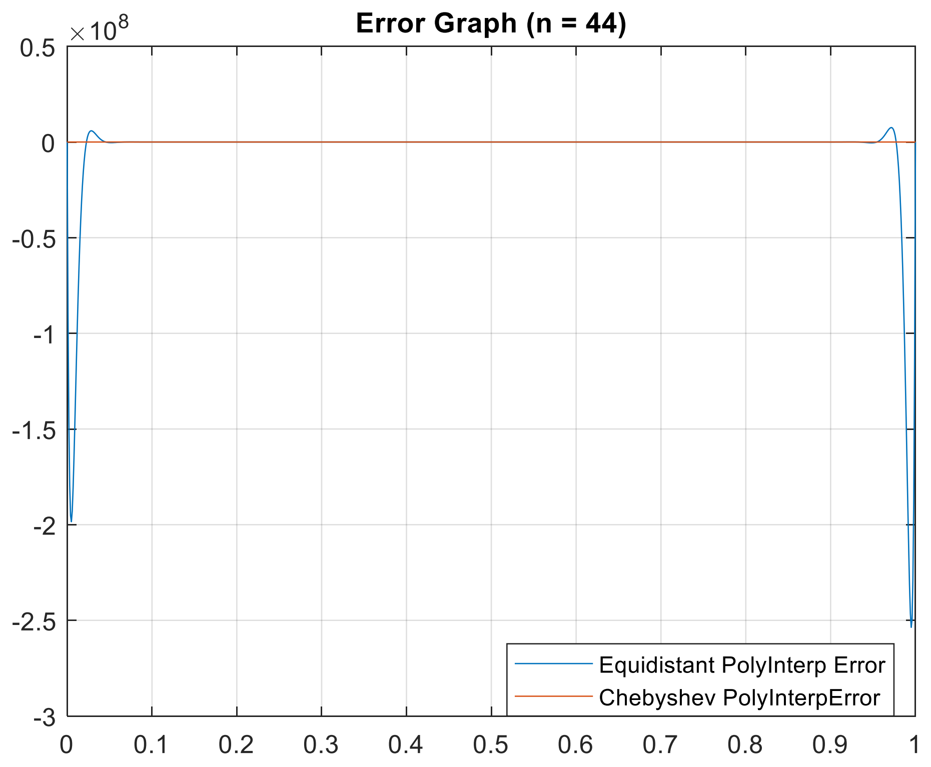
However, using Chebyshev-spaced data points, as the polynomial of degree increased, the error decreased for all *p*-norms. The trend of decreasing error was also observed in the Chebyshev points when the polynomial of degree was held constant; L_∞ was smaller than L_2, which in turn was less than L_1. Accuracy increased in tandem with *p*-norm.

Below are some error graphs to illustrate the differences in error norm calculations. Also note how the Chebyshev-spaced data points yielded more accurate data and had less drastic differences in error when the polynomial of degree was increased.

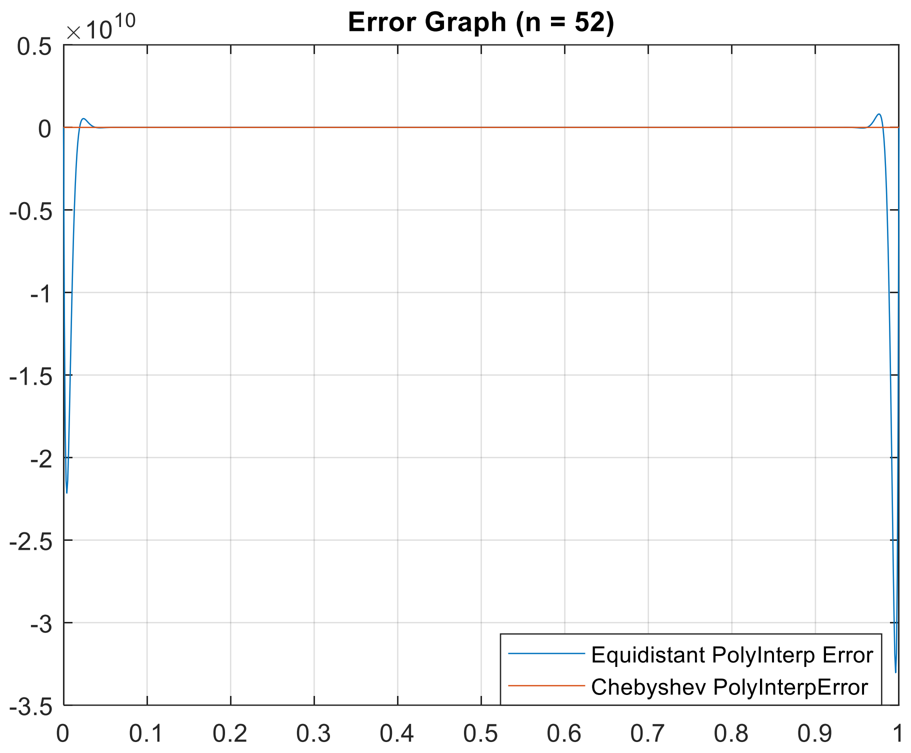


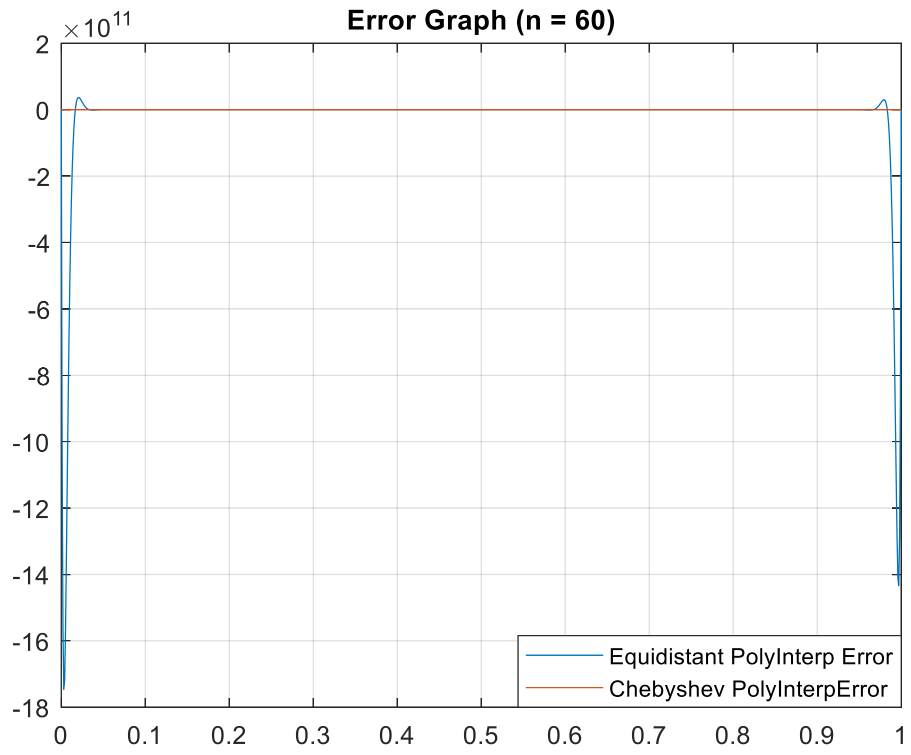
CS3200 – Assignment 1 Written Report
Skylar Shyu





CS3200 – Assignment 1 Written Report
Skylar Shyu

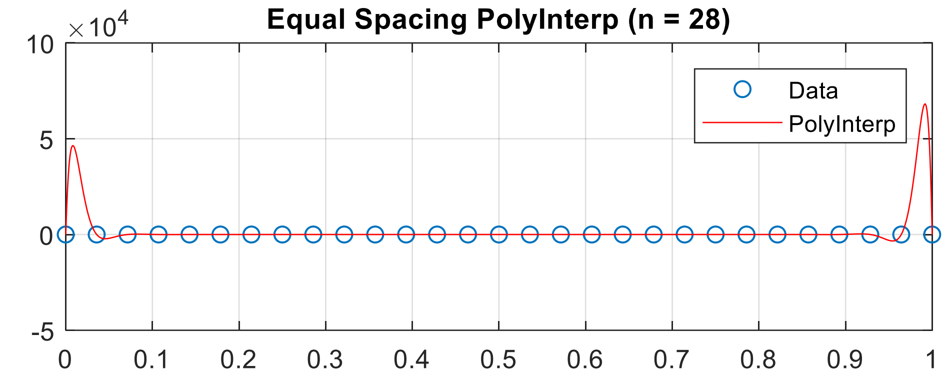


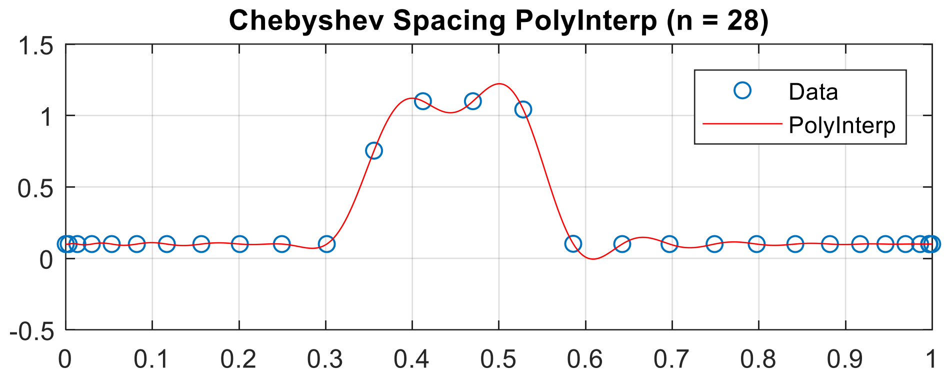


CS3200 – Assignment 1 Written Report
Skylar Shyu

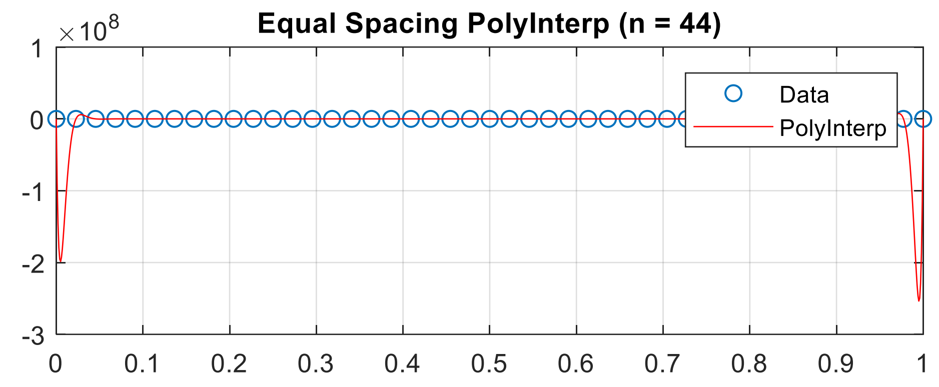
In comparison with the Runge's example in-class demo, Hubbard behaved very similarly. When using equally spaced points, as polynomial of degrees increased, so did the error in rapid fashion. When using Chebyshev spaced points, the interpolation aligned much more closely with the true solution as the polynomial of degrees increased, which was also true in the Runge's case.

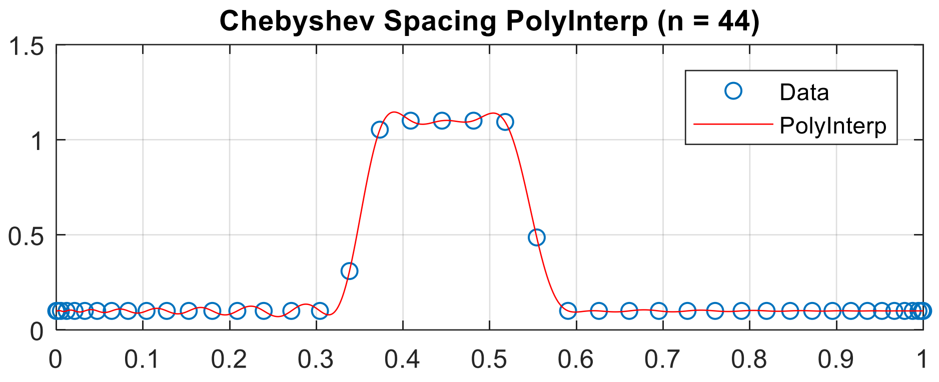
We can visually verify error trends by comparing a select few of the graphs of the polyinterpretations themselves to the true solution. Notice the large values of the axis for the equally-spaced data points to accommodate for the vector generated by the polyinterp function and compare it to the polyinterp generated by the Chebyshev points. Also, the Chebyshev polyinterp is much smoother in nature.

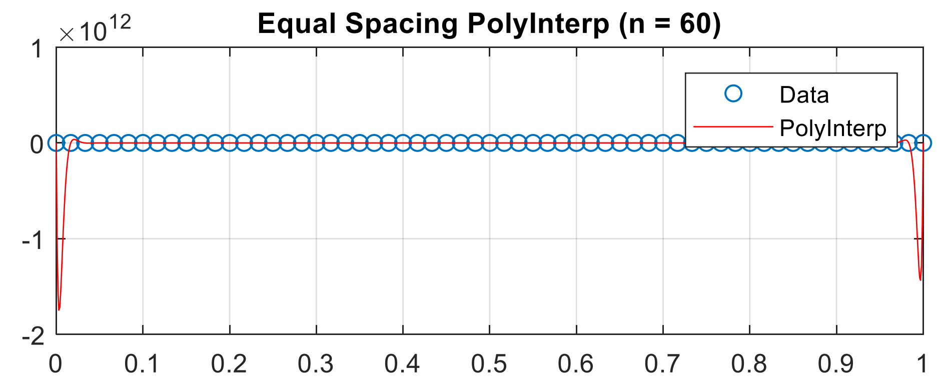


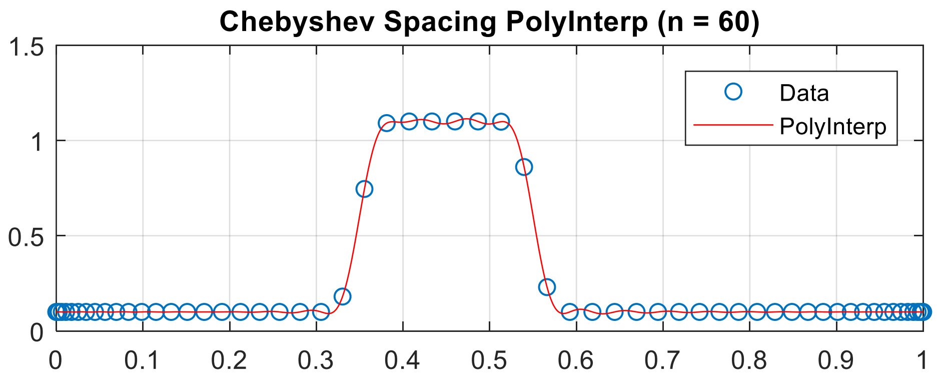


CS3200 – Assignment 1 Written Report
Skylar Shyu









CS3200 – Assignment 1 Written Report
Skylar Shyu

Question 3:

Due to the similarity in nature with Question 3 compared to Question 2, I will only be mostly emphasizing what was done differently here compared to the previous question. The methodology was largely the same, except that instead that the GelbTanner was explored in lieu of Hubbard, and the following differences came with while using the same five polynomial of degrees:

- Data points were generated over the domain of [-1 1] instead of [0 1]
- Adjustments had to be made with the Chebyshev points to shift the points left by 1 and then stretched over the x-axis to ensured that they stretched between -1 and 1.

The error and subsequent error norms were calculated in the same way, but obviously with different generated figures.

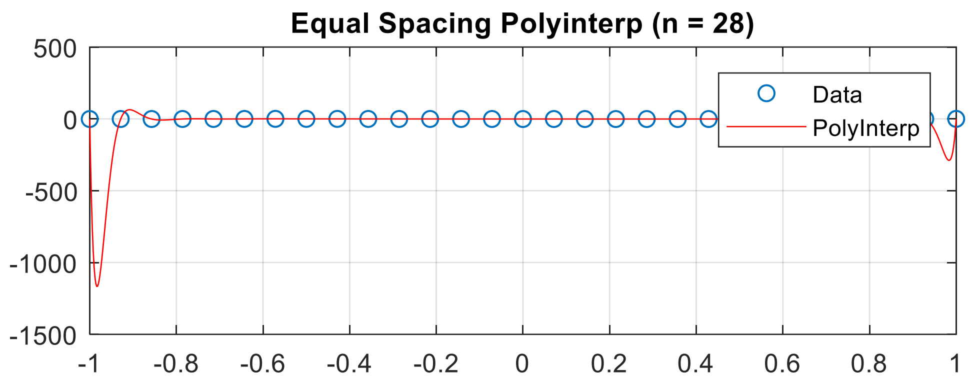
TABLE 2: *p*-norm Averages using Chebyshev and Equidistant Data Point Distributions with Respect to Polynomials of Degrees (n) for GelbTanner Function using Polyinterp Interpolation

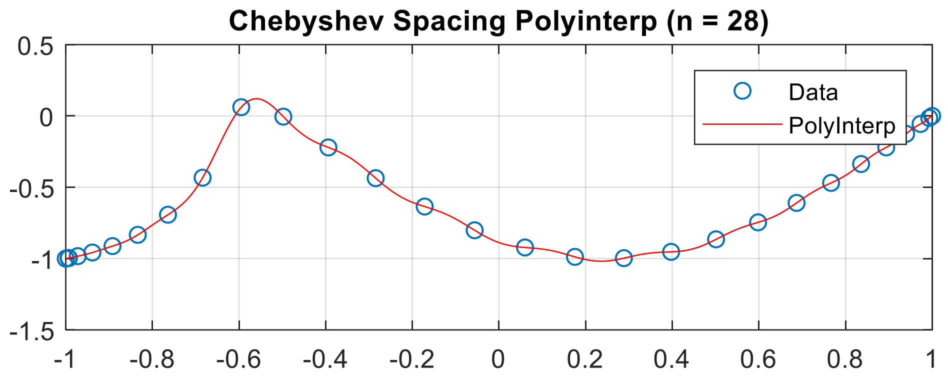
	Equidistant Spacing			**Chebyshev Spacing**		
n	**L_1**	**L_2**	**L_∞**	**L_1**	**L_2**	**L_∞**
28	2.852815e+01	4.446878e+00	1.166029e+00	2.963900e-02	3.376936e-03	9.664465e-04
36	1.879951e+03	3.382118e+02	1.022870e+02	3.276555e-02	2.309566e-03	7.674494e-04
44	1.155631e+05	2.333447e+04	7.908409e+03	3.717344e-02	2.245480e-03	5.973933e-04
52	7.234539e+06	1.608941e+06	5.999081e+05	2.484123e-02	2.701641e-03	9.515557e-04
60	4.669479e+08	1.128416e+08	4.537646e+07	2.326044e-02	1.756213e-03	6.989976e-04

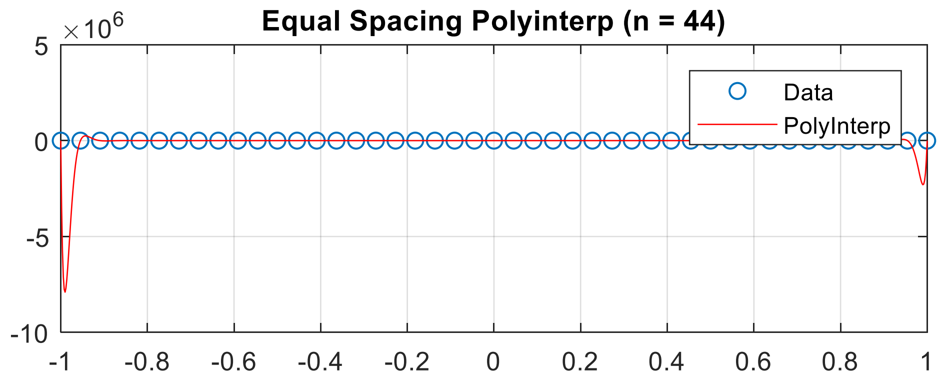
The error norms for GelbTanner behaves exactly like the error norms for Hubbard. As the *p*-norms increase, the accuracy increased for both the equidistant-spaced and Chebyshev-spaced points. However, increased to the polynomial of degrees only exacerbates the error norm for the equidistant spacings, while the same increases to polynomials of degrees yields a more accurate Chebyshev interpolation. Thus, GelbTanner behaves similarly to Hubbard when it comes to comparisons with the Runge's example in-class.

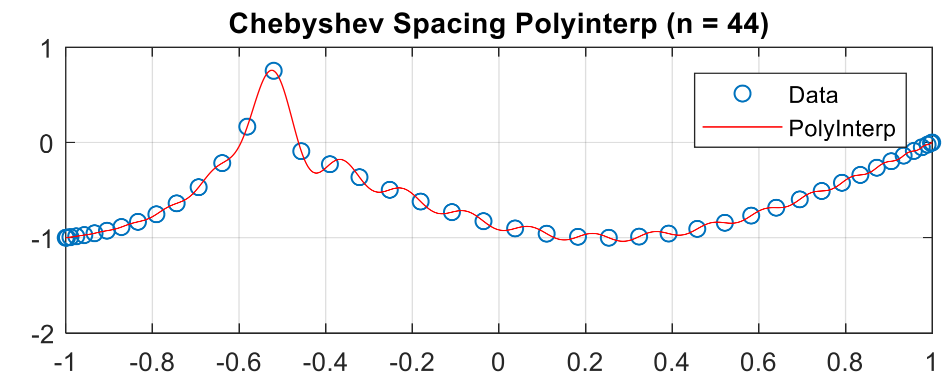
Visually, we can verify the increases and decreases in accuracy described above with the following graphs. As with the Hubbard graphs, we can check the y-axis and see how drastically the scale increases with the polynomial of degree in the equally spaced interpolation; we can similarly note the increase in accuracy and smoothness displayed by the Chebyshev interpolation.

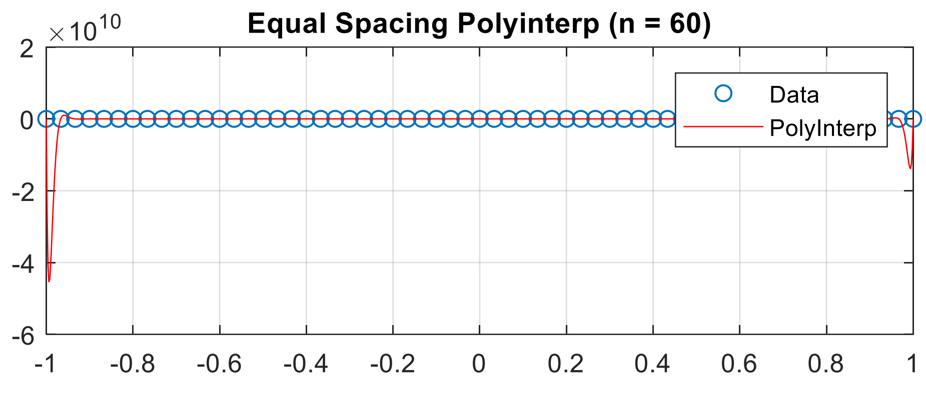
CS3200 – Assignment 1 Written Report
Skylar Shyu

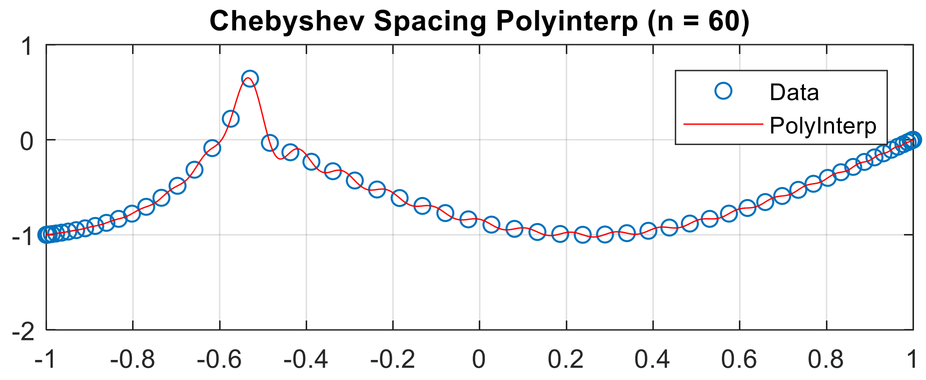












<u>Question 4:</u>

Instead of the polyinterp routine, the cubic routine, **spline(*x*,*y*,*u*)** and PCHIP routine, **pchip(*x*,*y*,*u*)**, were used to develop interpolations of the Hubbard and GelbTanner functions. The same methods and procedure from Questions 2 and 3 were utilized:

- [0 1] served as the domain for Hubbard calculations, while [-1 1] was used as the domain for GelbTanner
- Where polyinterp was used, spline and pchip were used in lieu
- Only equal-spacing was used over the same five polynomial of degrees: 28, 26, 44, 52 and 60
- *p*-norms of 1, 2, and ∞ were calculated the same way using by taking the difference of spline or pchip and the true solution across 1001 points, feeding them into the norm routine with the desire *p*-norm value and averaged over 1000-1.

CS3200 – Assignment 1 Written Report
Skylar Shyu

Table 3: *p*-norm Averages using Equidistant Data Point Distributions with Respect to Polynomials of Degrees (n) for Hubbard Function Using Spline and PCHIP Interpolation

	Spline Interpolant			**PCHIP Interpolant**		
n	**L_1**	**L_2**	**L_∞**	**L_1**	**L_2**	**L_∞**
28	7.680448e+00	5.622598e-01	9.091054e-02	5.544406e+00	5.184861e-01	7.994698e-02
36	3.240715e+00	2.433475e-01	3.771522e-02	3.039971e+00	2.871808e-01	4.641342e-02
44	1.387949e+00	1.101805e-01	1.900067e-02	1.732534e+00	1.656539e-01	2.592760e-02
52	6.865046e-01	5.342154e-02	8.885648e-03	1.060150e+00	9.962434e-02	1.574250e-02
60	2.872988e-01	2.277372e-02	3.383913e-03	7.037201e-01	6.674466e-02	1.009527e-02

For the Hubbard function, the spline interpolant overall seems to yield more accurate solution compared to the PCHIP interpolant, though both interpolations yield pretty similar results. Accuracy increased in tangent with the polynomial of degree and *p*-norm.

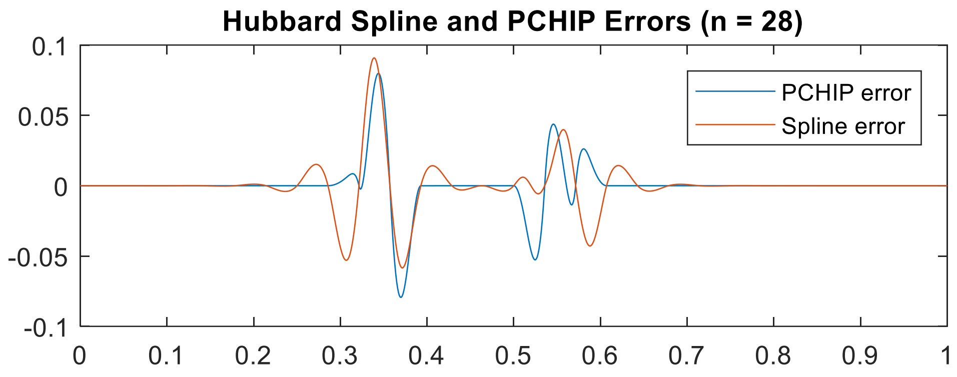
Table 4: *p*-norm Averages using Equidistant Data Point Distributions with Respect to Polynomials of Degrees (n) for GelbTanner Function Using Spline and PCHIP Interpolation

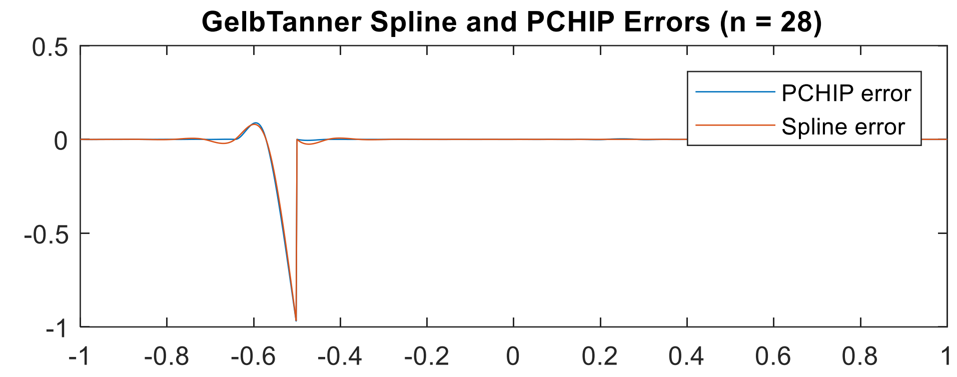
	Spline Interpolant			**PCHIP Interpolant**		
n	**L_1**	**L_2**	**L_∞**	**L_1**	**L_2**	**L_∞**
28	1.829913e+01	3.101370e+00	9.653119e-01	1.787729e+01	3.197424e+00	9.684368e-01
36	1.508452e+01	2.796522e+00	9.602655e-01	1.390984e+01	2.870490e+00	9.664743e-01
44	1.275694e+01	2.558805e+00	9.549577e-01	1.130929e+01	2.623091e+00	9.648037e-01
52	1.101301e+01	2.367694e+00	9.494491e-01	9.484777e+00	2.425772e+00	9.630169e-01
60	9.667597e+00	2.209945e+00	9.437700e-01	8.141189e+00	2.263369e+00	9.609920e-01

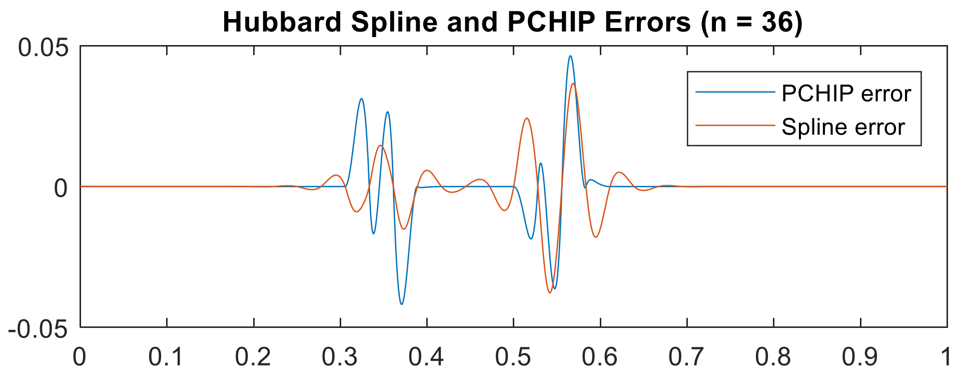
Like the Hubbard function in Table 3, the interpolations of both spline and PCHIP have fairly similar results with spline yielding marginally better results. Accuracy increased in tangent with the polynomial of degree and *p*-norm.

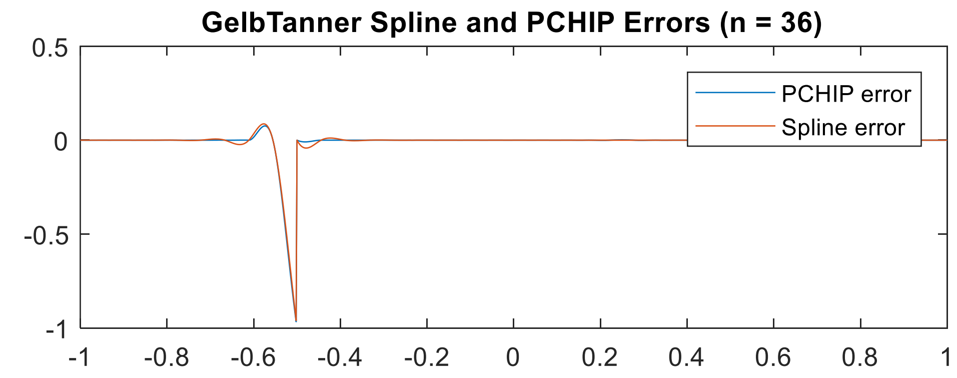
A better decision could be made by looking at the error graphs below:

CS3200 – Assignment 1 Written Report
Skylar Shyu

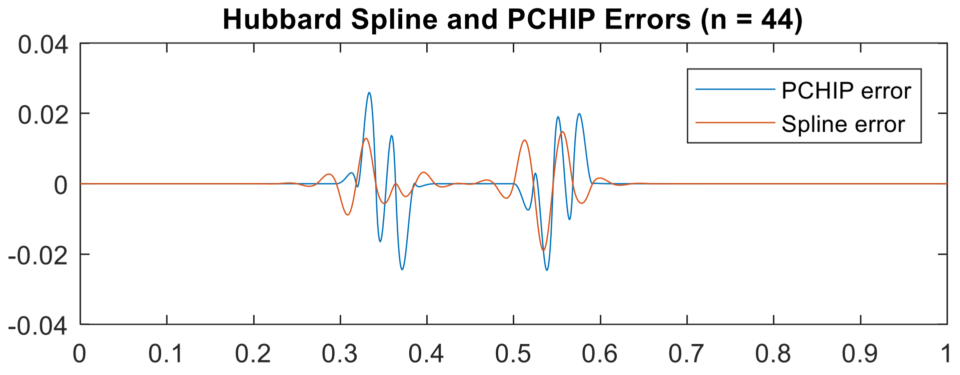


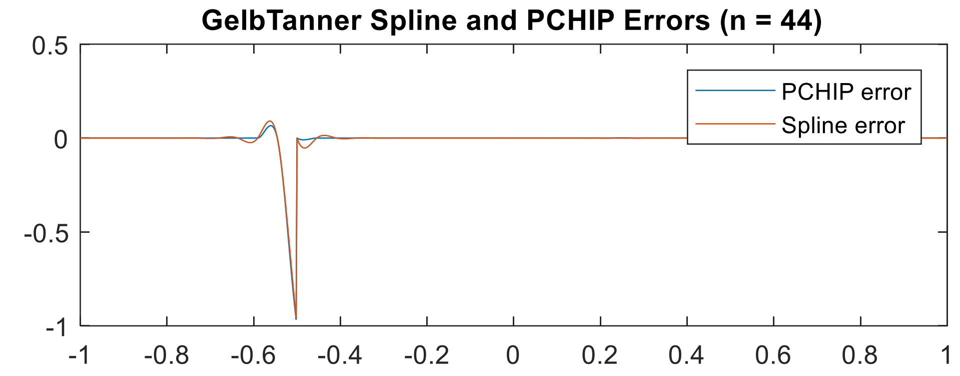


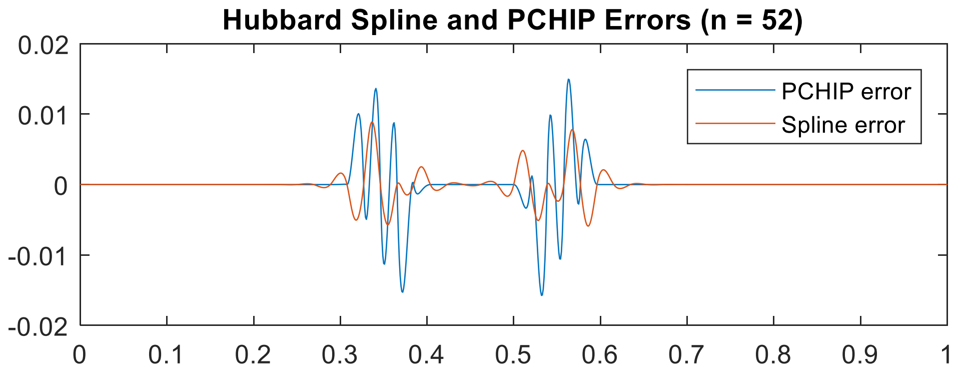


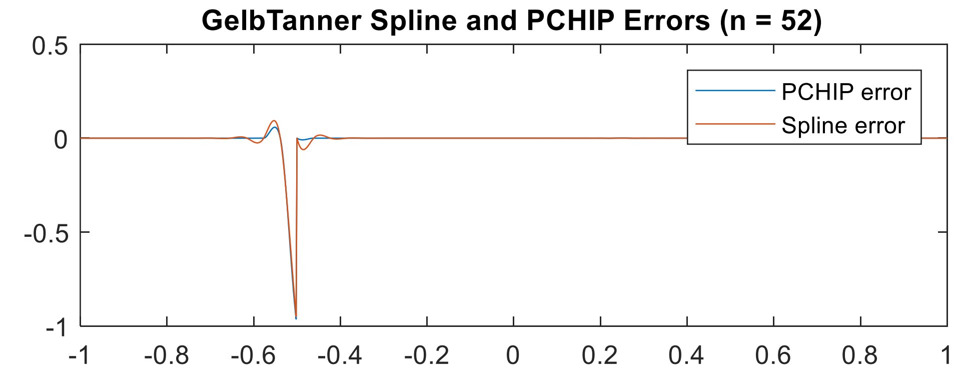


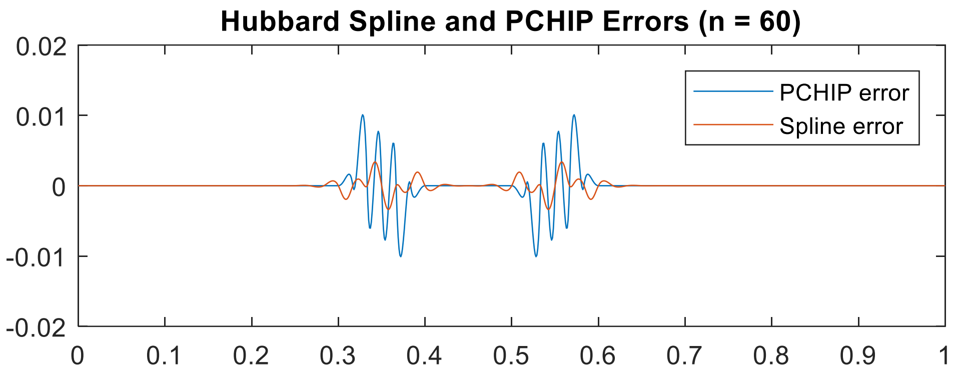
CS3200 – Assignment 1 Written Report
Skylar Shyu

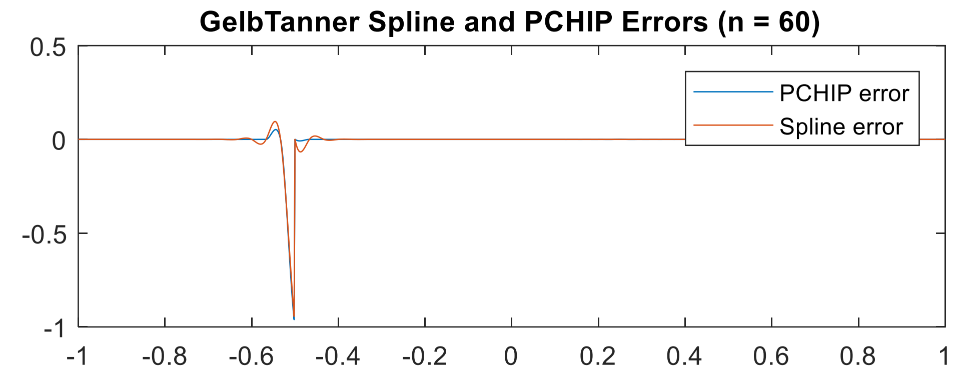








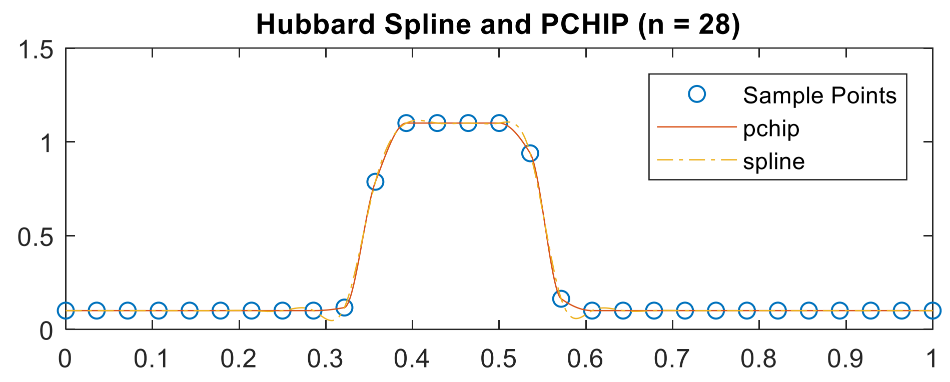


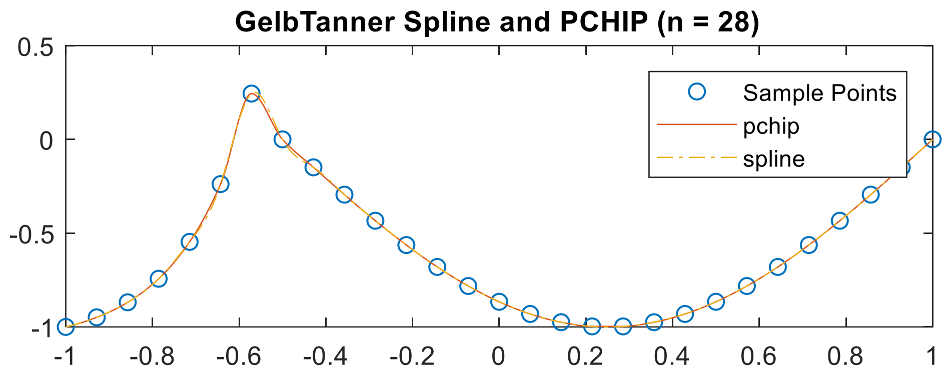


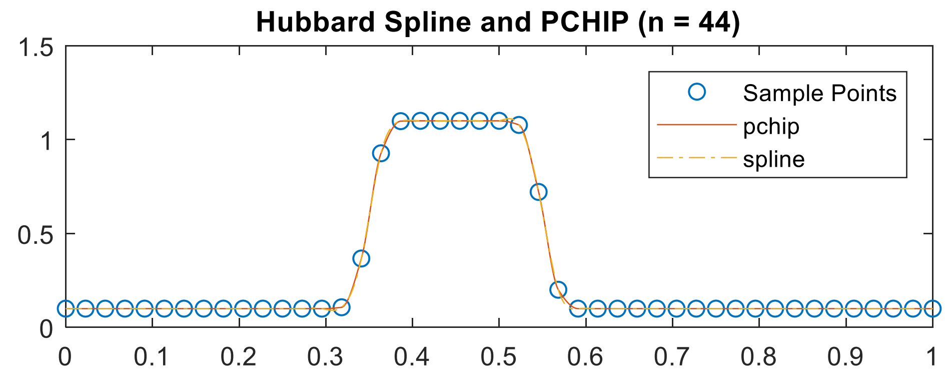
In Hubbard's case, it is visually evident that spline trumps PCHIP, and the more polynomial of degrees, the better the accuracy. This is probably owing to the fact that spline yields a smoother interpolation compared to PCHIP, which has more 'kinks' in the graph due to its eponymous piecewise nature. With GelbTanner, it is more difficult to discern visually whether spline or PCHIP is better due to the sheer amount of overlap in their error graphs.

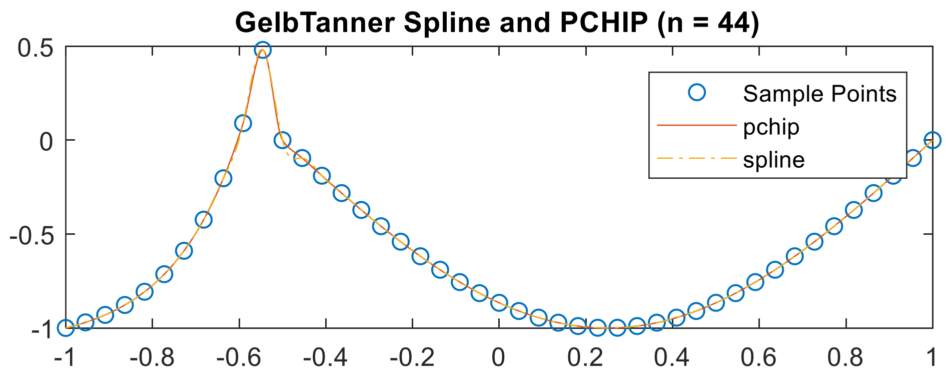
To make an informed decision on whether to use spline or PCHIP, below are a few figures of the actual interpolants. We can see that Hubbard is smooth, with gentle, rolling curves; in fact, there's a favourable plateau which is relatively easy to interpolate. With GelbTanner, however, there is a sharp increase followed by a sharp decrease over a small domain, which can be hard for PCHIP to compute. Thus, I would go with spline interpolations for both Hubbard and PCHIP. Arguably, with higher polynomial of degree, spline can have a better time adjusting to sharp changes in slope with curves, whereas the nature of PCHIP will create choppy 'blocks' as it tries to adjust to said changes.

CS3200 – Assignment 1 Written Report
Skylar Shyu

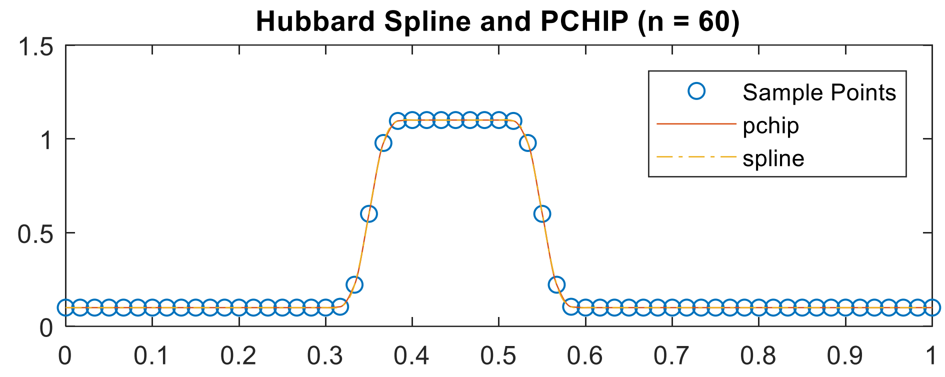


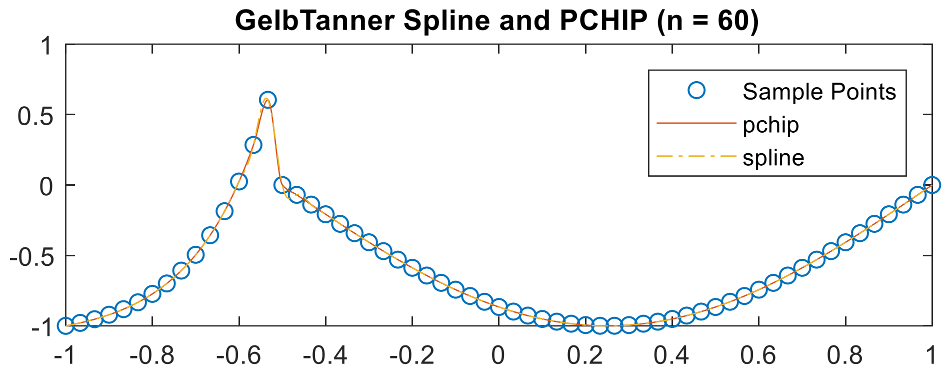






CS3200 – Assignment 1 Written Report
Skylar Shyu





Sources of Information:

Sources of information include the slides posted on Canvas (especially **ONE(a)INTERPOLATION-16.pdf**). Utilized the MatLab and Octave documentation for figure and plot help. Tajo, our course's TA, was tremendously helpful in explaining concepts (such as error calculation and vector mathematics) and introducing routines and methods in MatLab. I did not work with any classmates, but did reference Chris Widener's post on our class's Canvas Discussion forum regarding Question 2 here: https://utah.instructure.com/courses/463637/discussion_topics/2346581

Aside from gleaning information from the Canvas Discussion board for our class, I did not collaborate with any of my classmates.

Source code for **polyinterp.m** (written by Cleve Moler and copyrighted by The MathWorks, Inc.) was found at https://www.mathworks.com/matlabcentral/fileexchange/37976-numerical-computing-with-matlab?focused=6111365&tab=function

CS3200 – Assignment 1 Written Report
Skylar Shyu

Source codes **GelbTanner.m** and **Hubbard.m** were provided by Prof. Berzins. Other source codes referred to in this report, such as the Runge formula, were also provided by Prof. Berzins. They are all readily available in our course's Canvas page

Routines such as **exp(x)**, **norm(e, i)**, **pchip(x,y,u)**, and **spline(x,y,u)** all came with MatLab/Octave; I certainly did not write any of those routines.