Dan Ruley

CS 3200

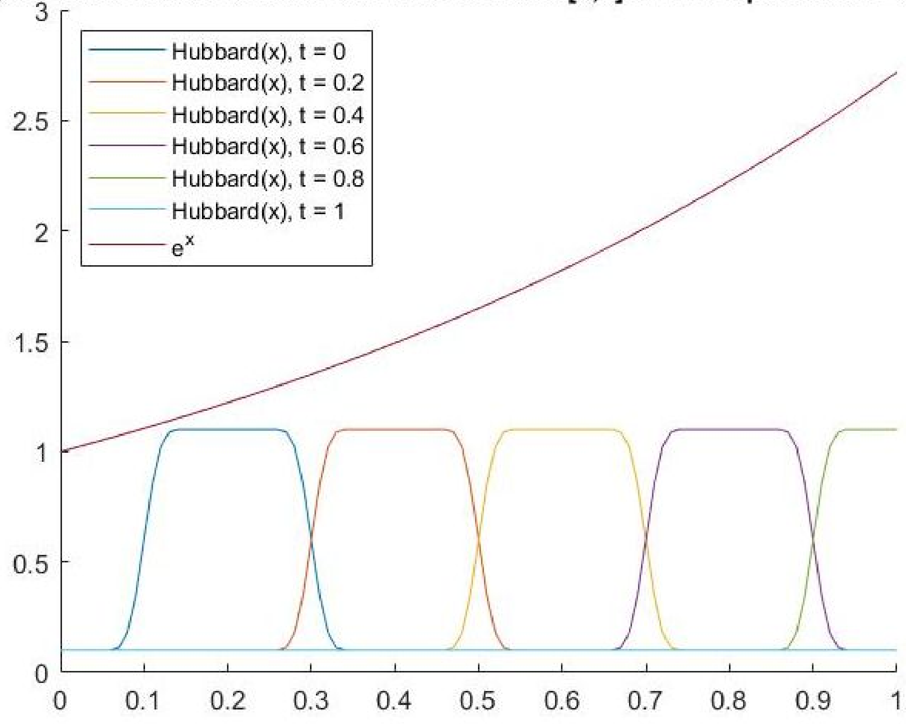
Assignment #1

February 3, 2020

1.)

I plotted the Hubbard function at six values of time between 0 and 1, along with the exponential function. The Hubbard function traces a sort of square wave as the values of time change. The top of the wave seems to be centered at 0.2 x-units to the right of the input time value and its peak is slightly over 1 on the y-axis. See Figure 1.

Figure 1: Hubbard Function for Values of t in [0,1] & the Exponential Function



2.)

I plotted the results of interpolating the Hubbard function using the Lagrangian interpolation routine supplied in the polyinterp function. I used equal-spaced and Chebyshev nodes in the interval [0,1] to define the interpolating polynomials of degrees 28, 36, 44, 52, and 60 and used 1001 test points. See Figures 2 and 3 on the next page.

Figure 2: Interpolating Hubbard Fn. Using Equidistant Points (T = 0.25)

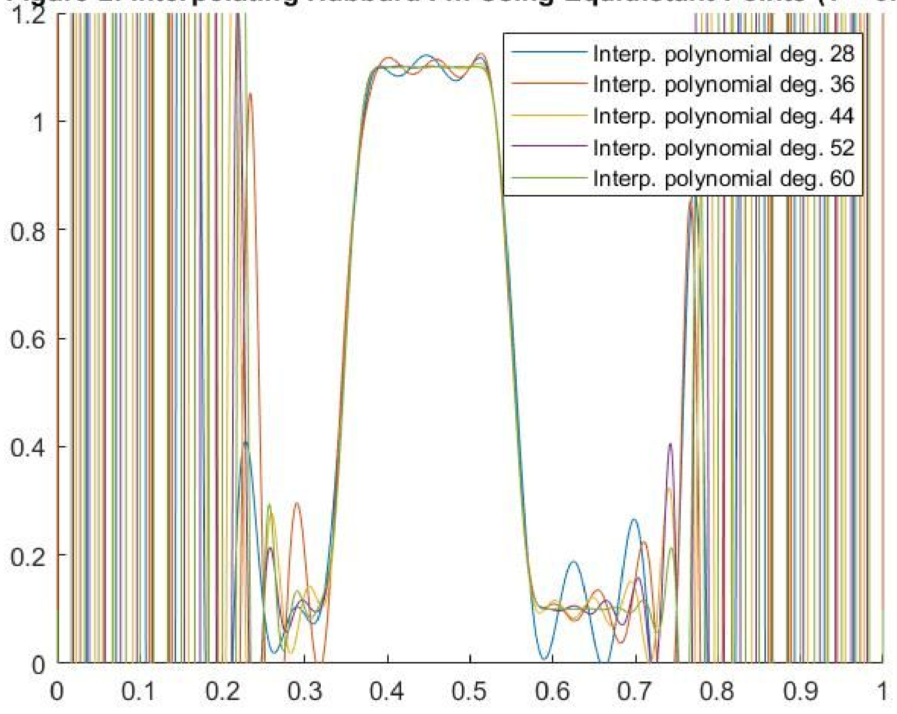
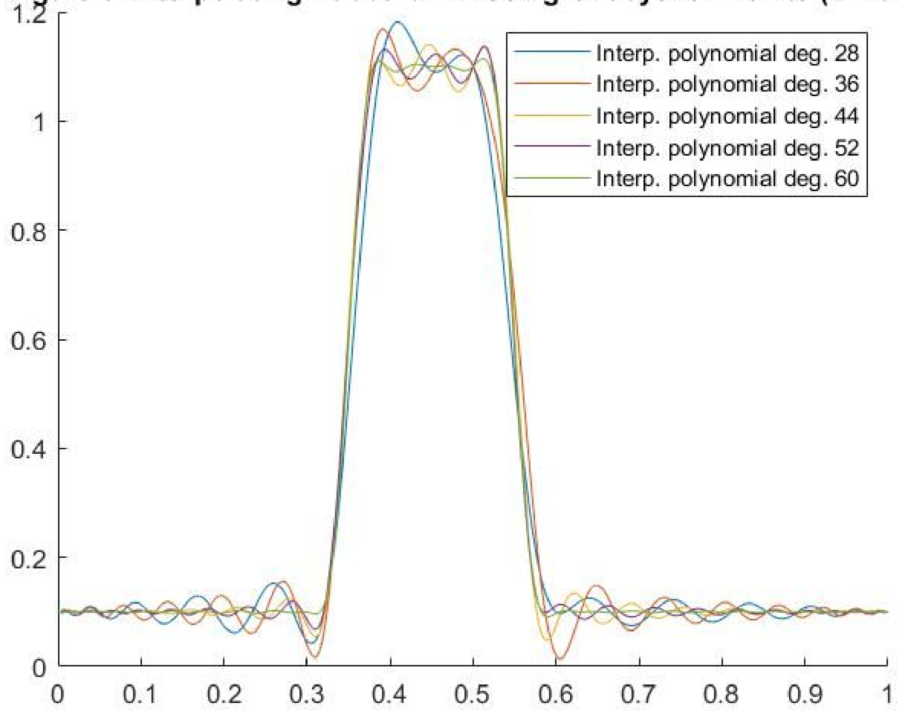


Figure 3: Interpolating Hubbard Fn. Using Chebyshev Points (T = 0.25)



Both the equal-point and Chebyshev interpolants seem to do a reasonable job at approximating the function in the “wave” section near the center; however, we can see the equal-points interpolants start to oscillate dramatically towards the left and right edges of the [0,1] interval. The results I found illustrate Runge’s phenomenon, where interpolating polynomials based on equally spaced points have large oscillations towards the edges of the interval. I also estimated the error of the interpolation per the procedure we discussed in class, using the Matlab norm function across the L1, L2 and INF norms. See Table 1 for these results.

Table 1: Hubbard Error for equal point and Chebyshev Lagrangian interpolation, approximated with L1,L2, and INF norms.

Degree	Equal L1	Equal L2	Equal INF	Chebyshev L1	Chebyshev L2	Chebyshev INF
28	78667.53	57182.89	68196	0.986	0.401	0.195
36	4567755.98	4313964.44	5970239.29	0.697	0.284	0.168
44	115685799.9	137150724	253653879	0.25	0.115	0.053
52	9797674430	14129361644	33029235854	0.161	0.08	0.045
60	$4.15 \cdot 10^{11}$	$6.92 \cdot 10^{11}$	$1.75 \cdot 10^{12}$	0.043	0.026	0.018

Clearly, the Chebyshev interpolants resulted in far less error on average, yielding higher accuracy with polynomials of larger degrees. This is clearly what I expected after a quick glance at the graphs and matches my intuition. Interestingly, the error grew exponentially for the equal-points interpolants as degree increased.

3.)

I plotted the results of interpolating the GelbTanner function using the Lagrangian interpolation routine supplied in the polyinterp function. I used equal-spaced and Chebyshev nodes in the interval [-1,1] to define the interpolating polynomials of degrees 28, 36, 44, 52, and 60 and used 1001 test points. See Figures 4 and 5 on the next page for the graphs.

Figure 4: Interpolating GelbTanner Fn. Using Equidistant Points

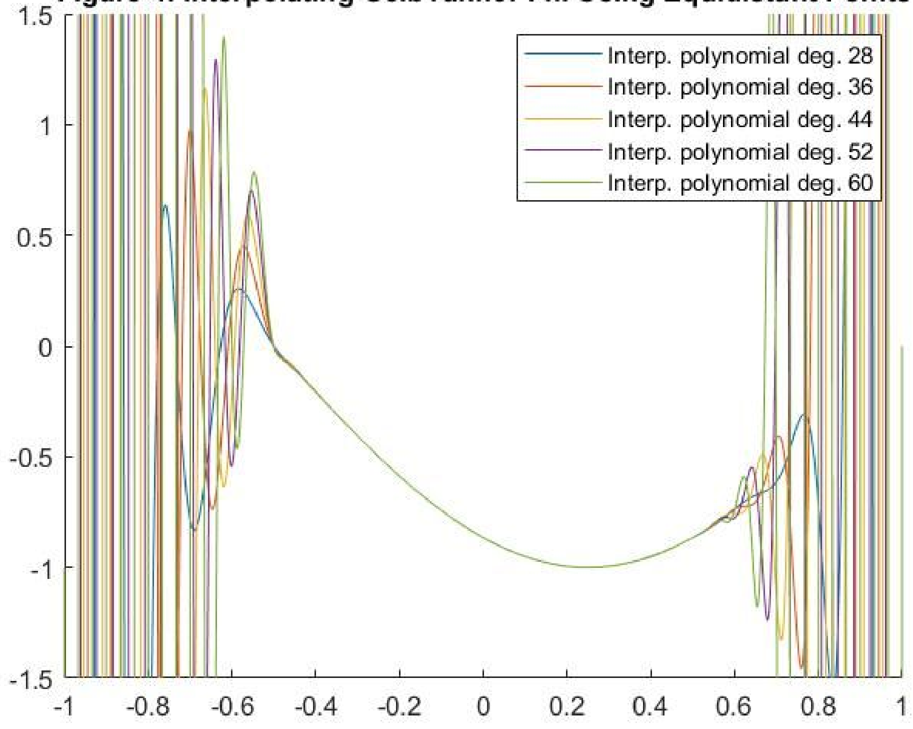
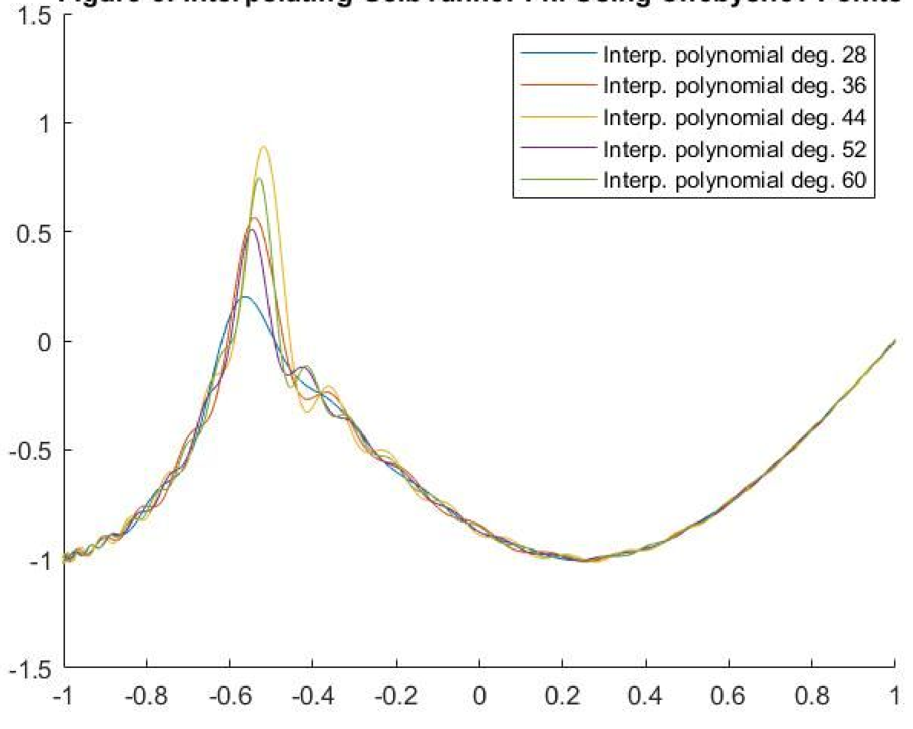


Figure 5: Interpolating GelbTanner Fn. Using Chebyshev Points



Again, the graph of the equal-points interpolating functions gives a strong example of Runge's phenomenon. There is huge oscillation at the edges of the equal-points graph and the Chebyshev interpolants fit much better across the entire interval of [-1,1]. I encountered a strange problem with the Chebyshev nodes $\in [-1,1]$ - The polyinterp yielded a result of +/- infinity when I used it. Therefore, I created a second polyinterp function that simply sets n = length(x) - 1 instead of n = length(x). For reasons that are not entirely clear to me, this fixed the problem of the interpolation result tending towards infinity.

I calculated the error norms for the equal-points and Chebyshev interpolants. Again, the Chebyshev error is much lower than the equal-points. The equal-point error also grows with the polynomial degree, while the Chebyshev error decreases. One interesting thing I noticed is that the interpolation seems to get more skewed at the point where the two pieces of GelbTanner join together. Depending on which part of the function is of interest, it may actually be more beneficial to interpolate with a lower-degree polynomial because they seem to be more accurate around the connecting point of $x \approx -0.6$. See Table 2 for these results.

Table 2: GelbTanner Error for equal point and Chebyshev Lagrangian interpolation, approximated with L2, and INF norms.

Degree	Equal L2	Equal INF	Chebyshev L2	Chebyshev INF
28	1187.29	1164.86	0.836	0.936
36	79637.58	102184.76	0.474	0.638
44	4969951.21	7900500.29	0.595	0.782
52	315223837.1	599308188	0.455	0.874
60	20581367386	45331084851	0.282	0.545

4.)

I modified my code to interpolate the Hubbard function using Matlab's cubic spline and Pchip functions. Again, these used polynomials of degree 28, 36, 44, 52, and 60 over the [0,1] interval with 1001 points. See Figures 6 and 7 on the next page for these results.

Figure 6: Interpolating Hubbard Fn. With Matlab Spline

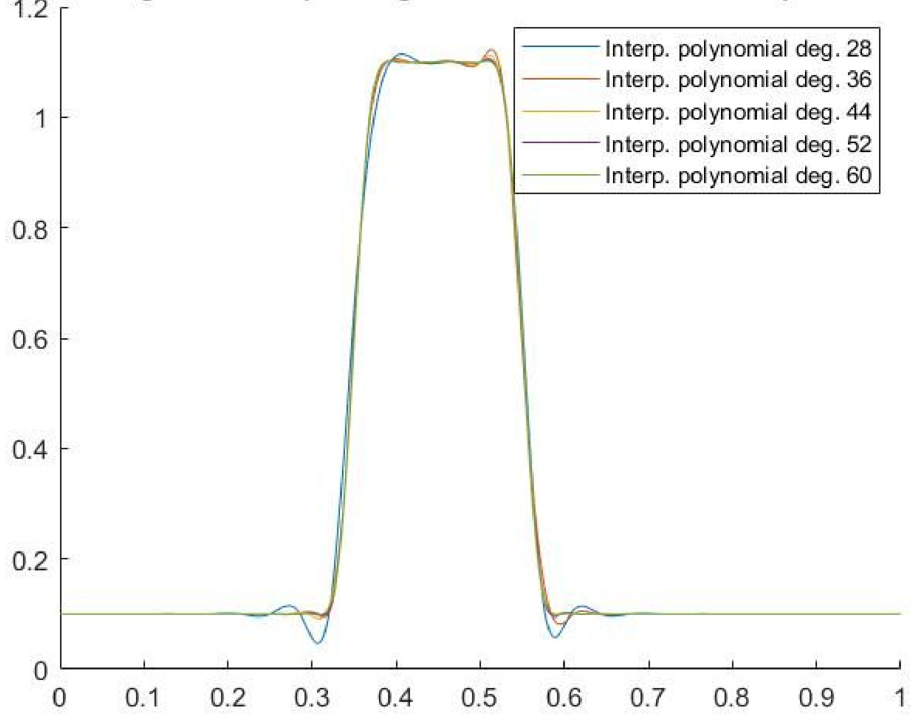
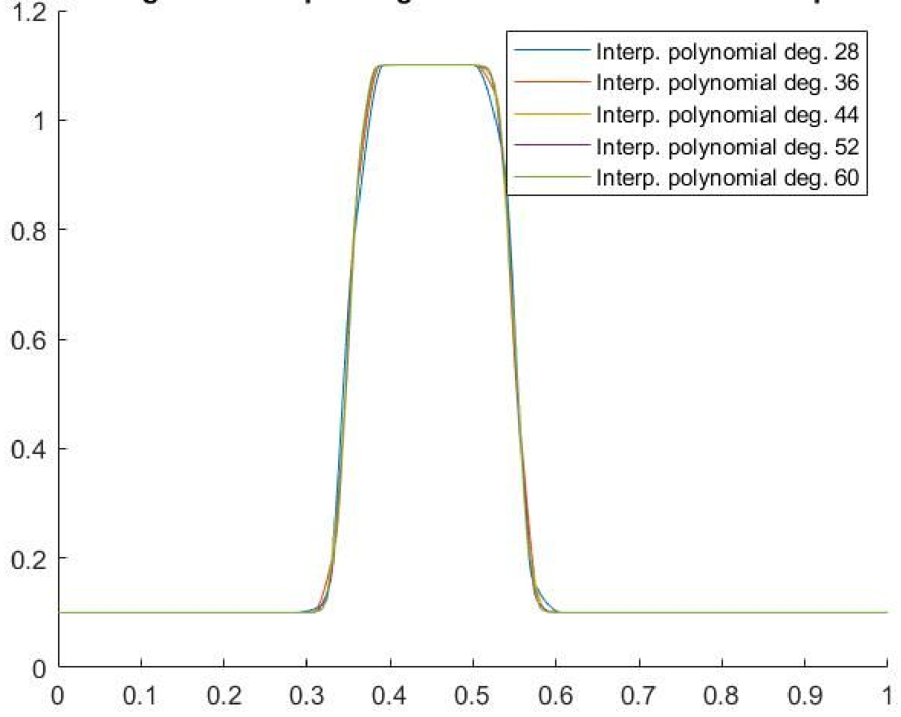


Figure 7: Interpolating Hubbard Fn. with Matlab Pchip



These results show fairly convincingly that spline and Pchip interpolation results in far greater accuracy than Lagrangian interpolation. As shown in Table 3 below, the error estimation is significantly lower than even the polynomials with Chebyshev nodes. Still, the spline seems to have some slight variation and higher errors than the Pchip function. Personally, I would opt to use the Pchip function for interpolation because it seems to be the most accurate of the four methods we have examined when interpolating the Hubbard function.

Table 3: Hubbard Error for Cubic Spline and Pchip interpolation, approximated with L2 and INF norms.

Degree	L2 Error Spline	INF Error Spline	L2 Error Pchip	INF Error Pchip
28	0.108	0.091	0.0998	0.08
36	0.041	0.038	0.0485	0.046
44	0.017	0.019	0.0253	0.026
52	0.007	0.009	0.014	0.016
60	0.003	0.003	0.0087	0.01

I also modified my code to interpolate the GelbTanner function using Matlab's cubic spline and Pchip functions. Again, these used polynomials of degree 28, 36, 44, 52, and 60 over the [-1,1] interval with 1001 points. See Figures 8 and 9 on the next page for these results.

Figure 8: Interpolating GelbTanner Fn. with Matlab Spline

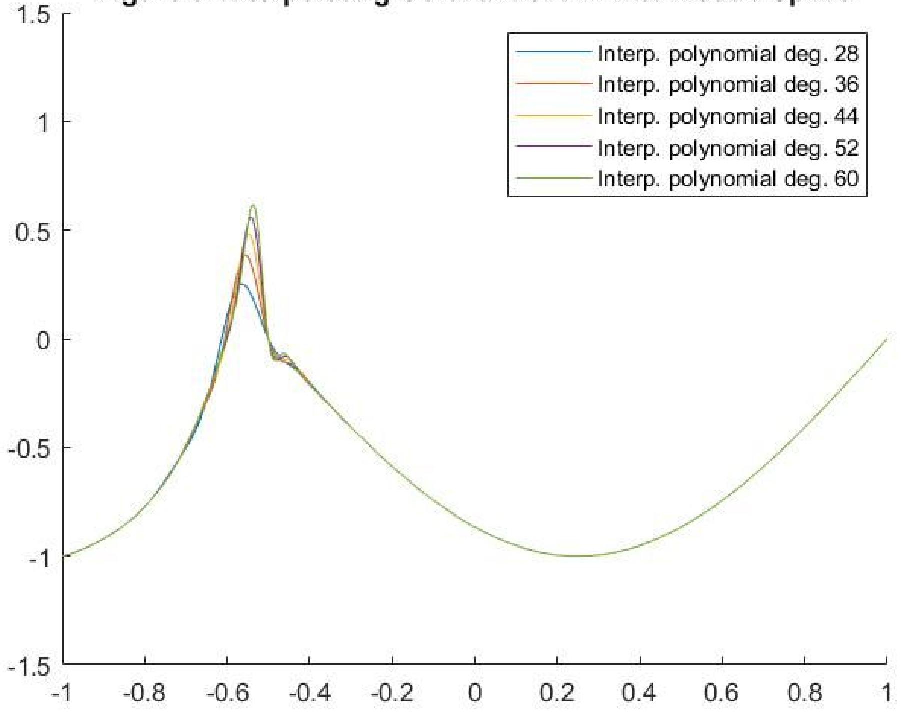
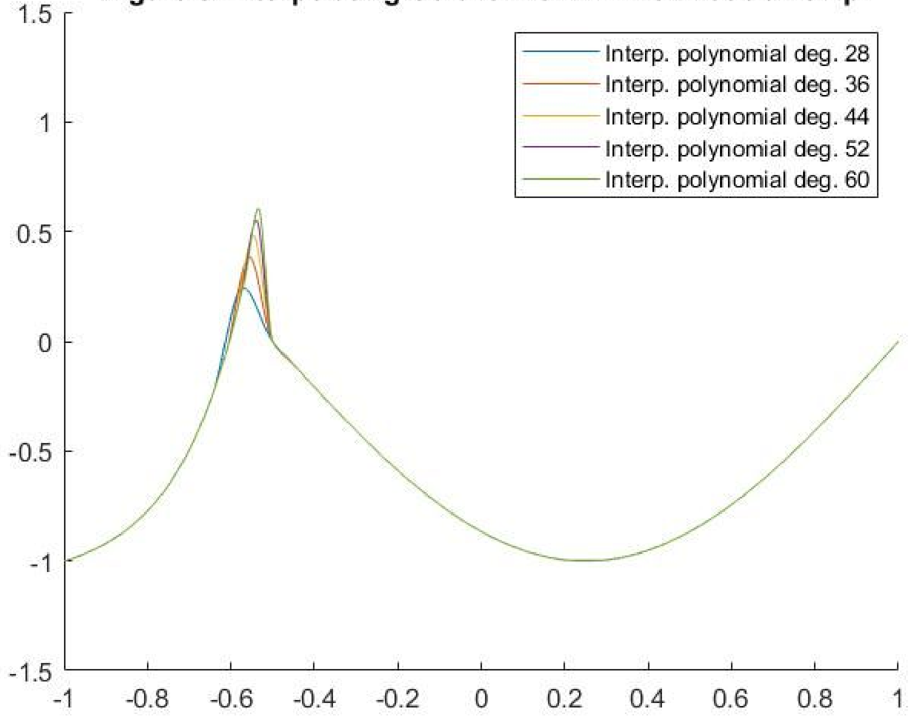


Figure 9: Interpolating GelbTanner Fn. with Matlab Pchip



Again, the Spline and Pchip interpolation seems to yield more accurate results with much less oscillation. However, they still suffer from the same problem with the connection point of the piecewise GelbTanner function. Also, the Spline interpolants seem to be more accurate than the Pchip, in contrast to the results I found using them on the Hubbard function. The errors predictably decrease as the degree of the polynomial increases. Based on these results, I would opt for a Cubic Spline interpolating function to use on a piecewise function such as GelbTanner. See Table 4 for the error results.

Table 4: GelbTanner Error for Cubic Spline and Pchip interpolation, approximated with L2 and INF norms.

Degree	L2 Error Spline	INF Error Spline	L2 Error Pchip	INF Error Pchip
28	0.597	0.965	0.615	0.968
36	0.473	0.96	0.485	0.966
44	0.39	0.955	0.4	0.965
52	0.332	0.949	0.34	0.963
60	0.288	0.944	0.295	0.961

References: Lectures and powerpoint slides as well as code examples. I also used the MatLab documentation to discover how to plot and format certain things in Matlab.

Collaborators: I discussed some of the concepts of this assignment with Gavin Gray, particularly how to solve the issue of the GelbTanner Chebyshev polyinterp yielding results of +/- infinity, since we both ran into this problem.