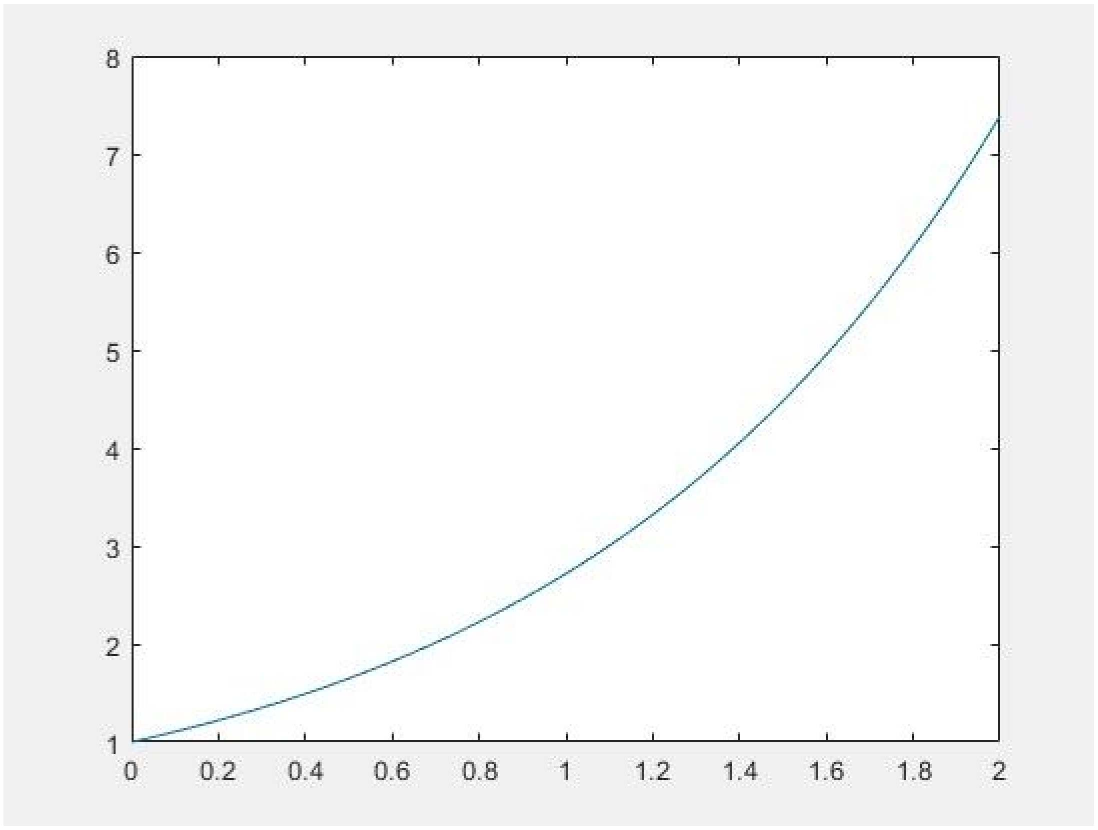
Brandon Walters

Prof. Berzins

CS 3200

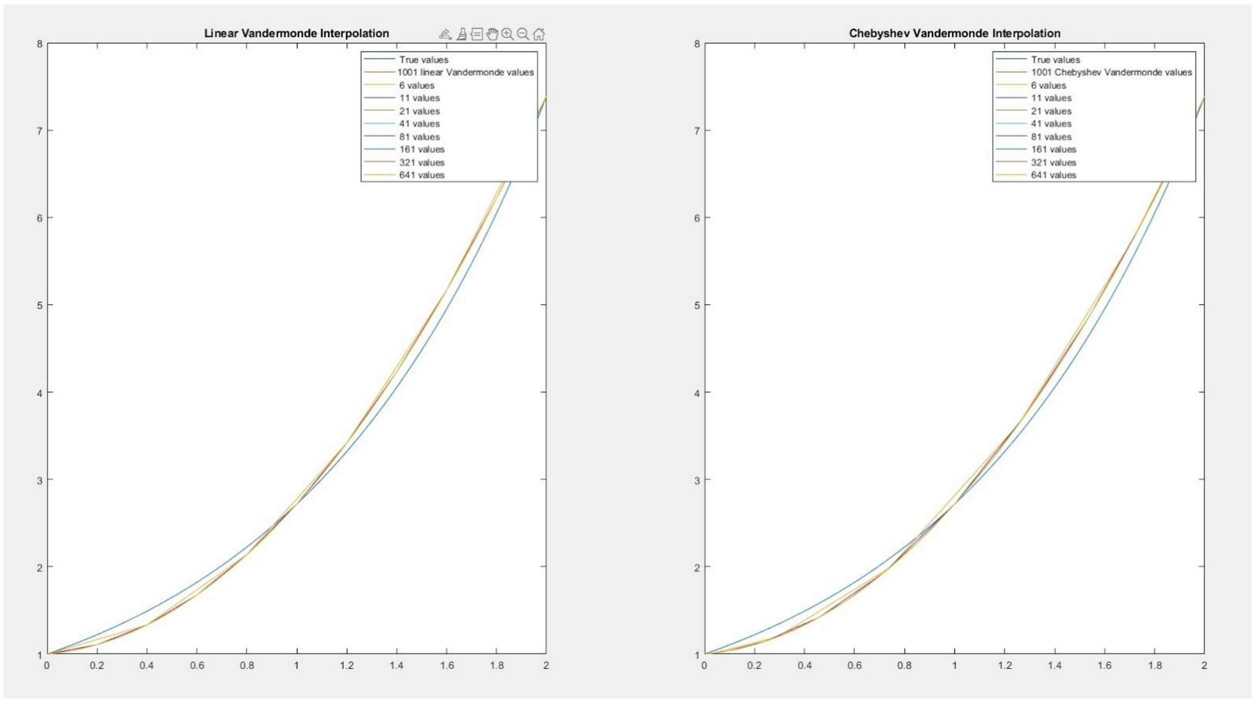
Homework 2 Report

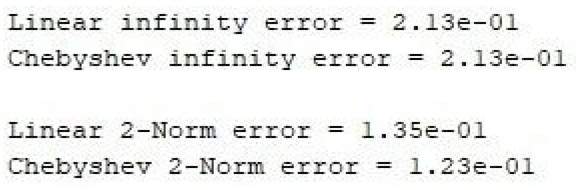
1. Our simple program to plot the value of exp(x) on an interval [0 2] of 1001 evenly spaced points gives us the following plot:



2. When we use the Vandermonde matrix to linearly interpolate our different point sets on both linearly spaces and Chebyshev point sets, we immediately notice that both interpolations have a good fit to the actual values we are looking for. It seems like the number of points has a small but noticeable effect on the accuracy of our fit: the line bends down closer to the true value line when it has more point data to interpolate from. We also find from our error printouts that

both methods of point generation give us similarly low errors, as expected: the 2-Norm variant shows a slight improvement for Chebyshev over the linear version, which fits with our knowledge of how Chebyshev points help tamp down on oscillations towards the very ends of the graph, thus slightly decreasing error.

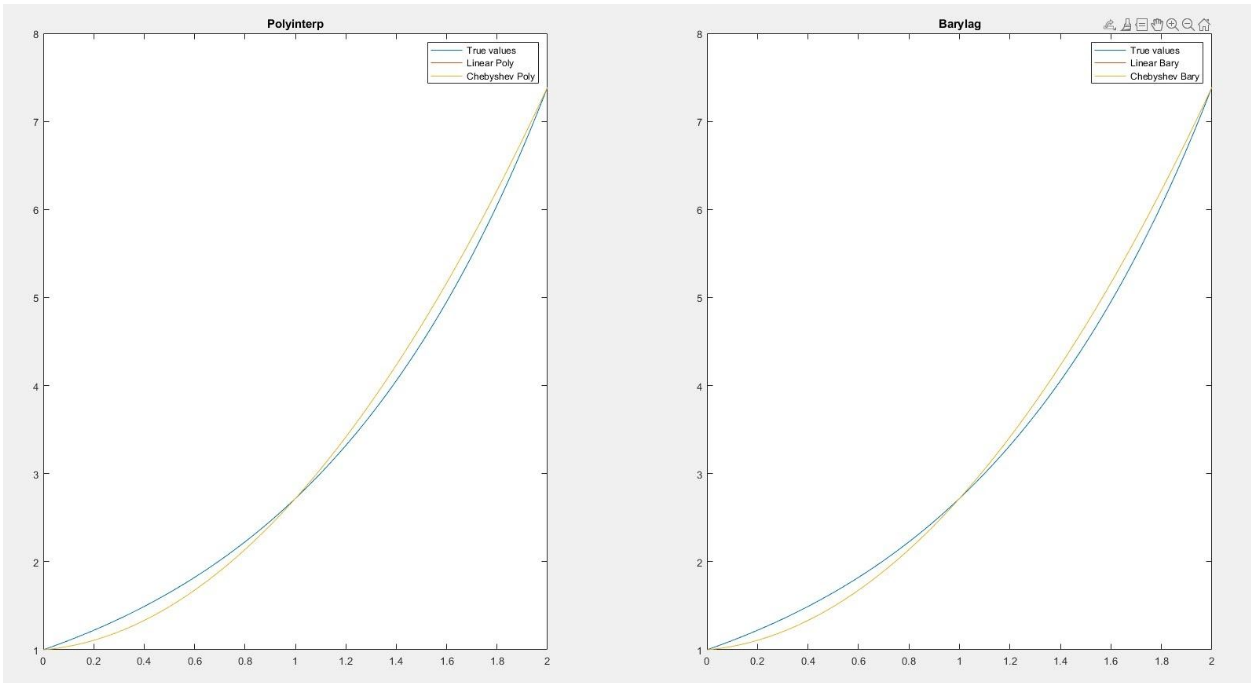



```
Linear infinity error = 2.13e-01
Chebyshev infinity error = 2.13e-01

Linear 2-Norm error = 1.35e-01
Chebyshev 2-Norm error = 1.23e-01
```

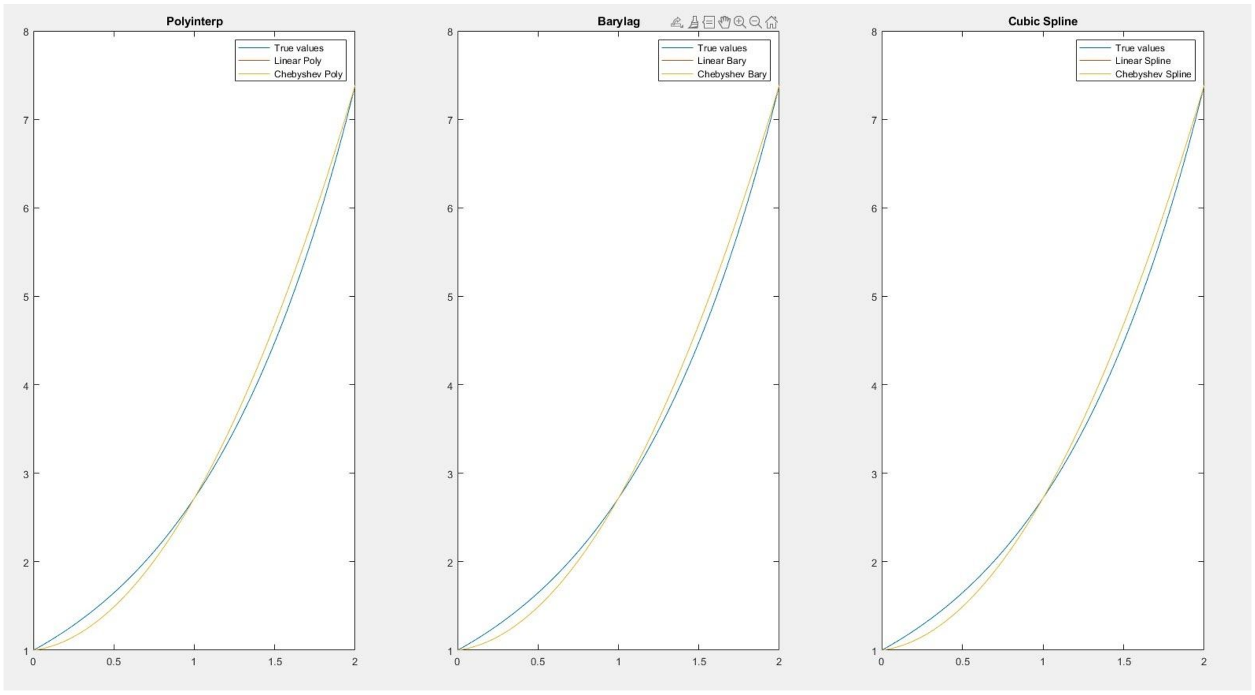
3. Using polyinterp and barylag immediately shows us a much more consistent set of lines with a strong fit to the actual data. We also notice polyinterp’s issues with stability (as noted before in this class), which led to having to use the 6-point linear estimation and the 81-point Chebyshev version, although it seems like the number of points doesn’t matter too much beyond the initial stability concerns. Barylag was much more stable, which allowed us to use higher point counts in

our work, although we still had some issues with ballooning at higher point values that limited our point usages.



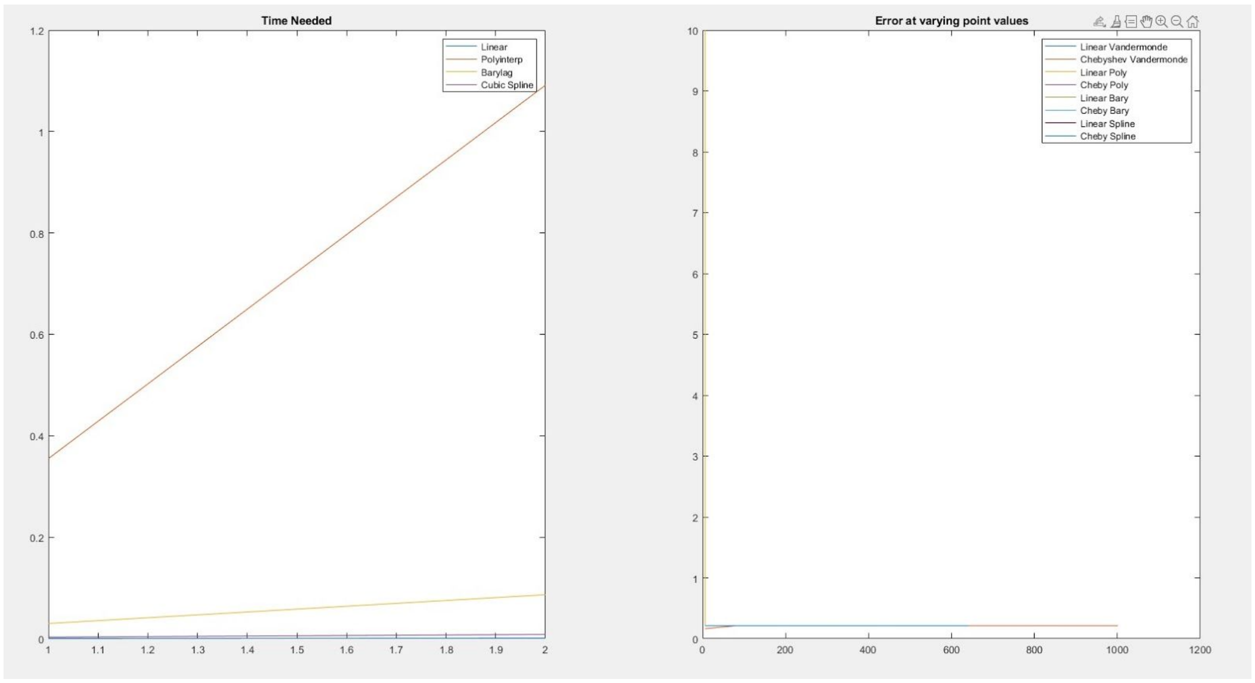
4. After adding the cubic spline function provided by Matlab, we can see a very similar result in terms of the line fit. However, unlike polyinterp and barylag, we get a much more stable result in the case of the spline, meaning that the number and location of the points we select for the calculation is much less important

than it was for the previous two methods.



5. Graphing our times and for our different interpolation methods along with the varying errors for a variety of point arrays immediately shows us one serious culprit for some issues: Polyinterp, which had some serious issues at both ends of the spectrum when it came to error, as well as the slowest time of the four major methods. This was to be expected, however, as it is the most complicated and time-consuming method of them all, and we already knew it had major stability issues with large point values. Barylag was the second-worst on time, with the spline method being just a fraction slower than the linear generation. In terms of error, aside from polyinterp all of the methods were very low on error issues, with Chebyshev points providing marginal increases across multiple methods. This was what we expected, as Chebyshev points help lower error right at the ends of the point set, and linear interpolation and the spline method are purposefully built to generate quickly, while barylag and polyinterp are more

data-heavy and use more calculations to get a better estimation.



6. In summary, we know that classic and simple linear interpolation will provide the fastest answer, which makes sense as the method itself is the most simple and requires very little cubic or interpolation-related math. The most accurate and robust method of calculation is the cubic spline method, which is very close to the fastest method, has low error values, and creates a very accurate approximation.
7. Barylag is a dense method, but the basics of its actual work are quite simple: Starting with a two-column vector that contains the x and y coordinates of the given points and the x coordinate list of all of the points we want to interpolate, we start by computing the barycentric weights of each given point's x-coordinate, which is contained in the first column of the data variable. We then create a matrix W that contains computed weights for each point we want to interpolate. Next, we calculate the distance from any interpolated point in our matrix x to one of the given points. We use these values to find any points that were already

given to us in the function input. From there, we finally end by calculating each point that was not already given to us and constructing our final array to return, which contains our interpolated points.