CS 3200

Lianrui Geng

Assignment 06

Problem 1:

Part a: Define symbolic variables x and y and create a random polynomial function y(x).

Part b: Evaluate the function y(x) at 501 evenly spaced points w between 0.5 and 5.5, and store the result in vector v.

Part c: Create a figure for plotting the interpolation results.

Part d-g: Perform polynomial interpolation using n+1 evenly spaced points between 0.5 and 5.5 for n in the range 1 to 10. For each n:

a. Compute the sample points (x_points, y_points).

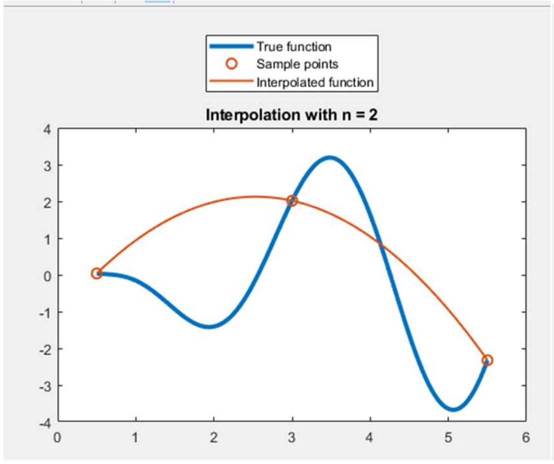
b. Plot the true function y(x) in thick lines.

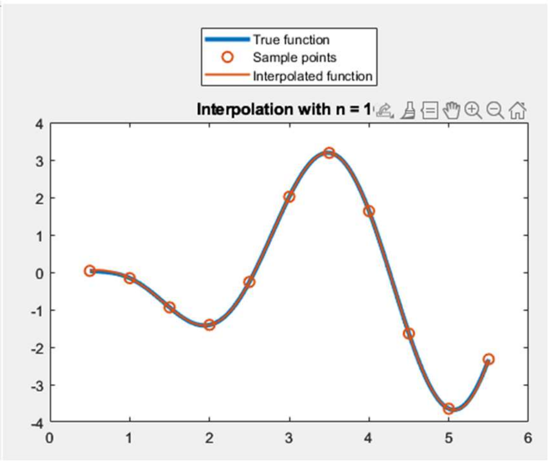
c. Plot the sample points using circles with a specific size and line width.

d. Perform polynomial interpolation using a custom function 'polyinterp' and plot the interpolated function with the same color as the sample points.

e. Set the x-axis limits and add a title and legend to the plot.

Part h: Pause for 0.5 seconds and clear the plot before proceeding to the next iteration.



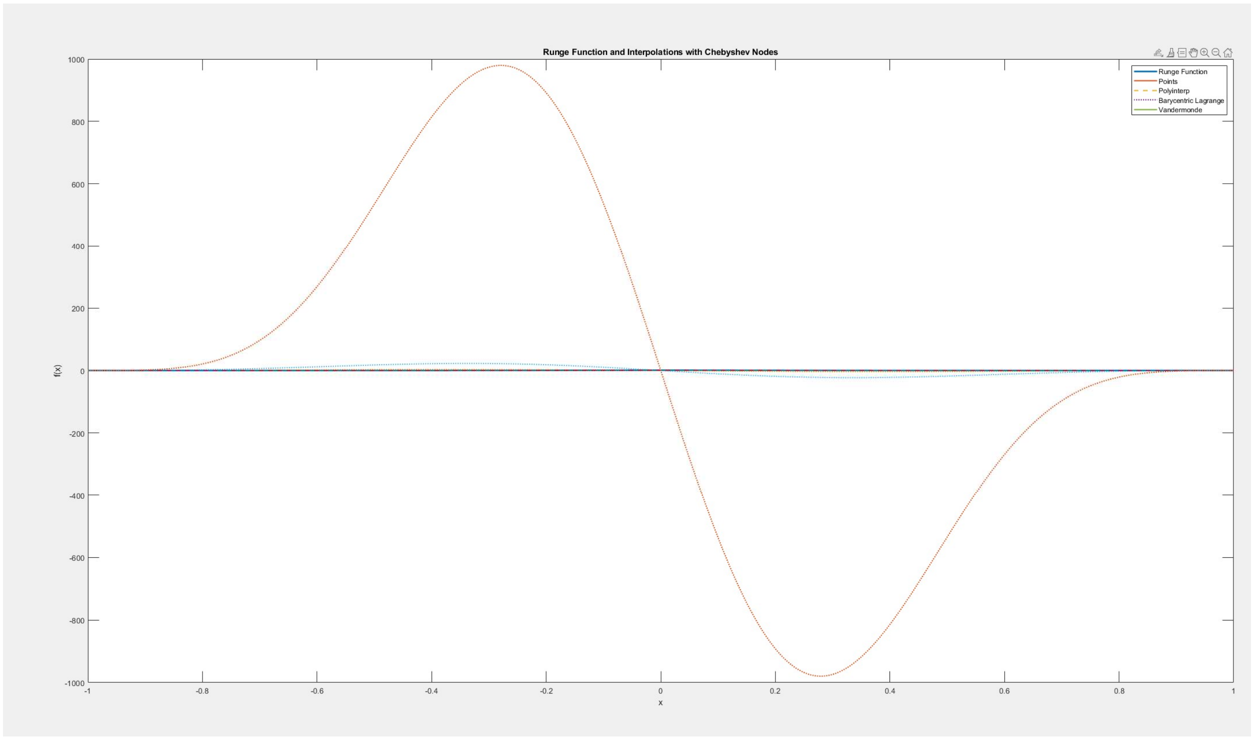


(I just show the first step and the last step of the animation.)

When running the animation multiple times, the random polynomial function y(x) will change. The interpolation process will generally perform well if the polynomial degree is low or if the points are spaced closely enough. As the value of n increases, the number of sample points increases, which should typically lead to a better approximation of the function. Increasing the range of n will use more sample points for the interpolation, which generally leads to a better approximation of the function. However, depending on the behavior of the random polynomial, there may be diminishing returns for increasing n, as the interpolation might already be accurate enough with a lower number of points.

However, I found sometimes the function can suffer from oscillations, especially near the edges of the interpolation interval, known as Runge's phenomenon.

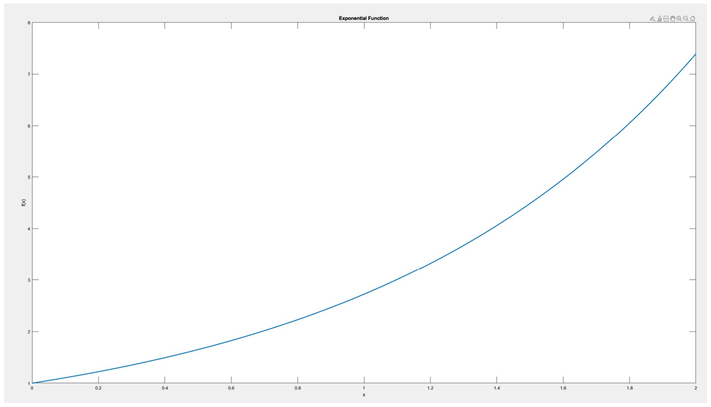
Problem 2:

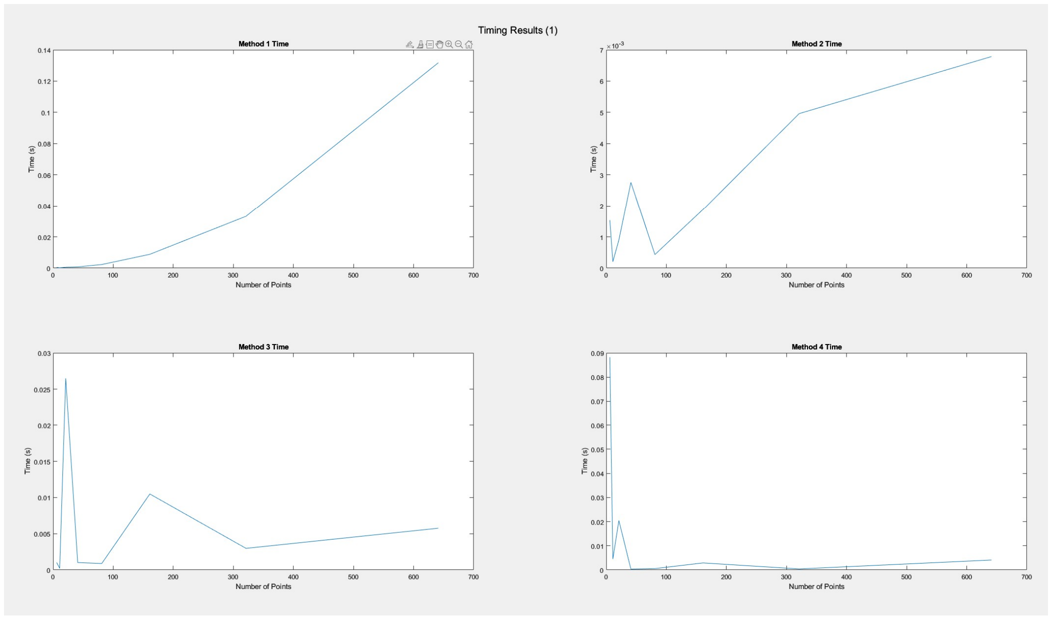


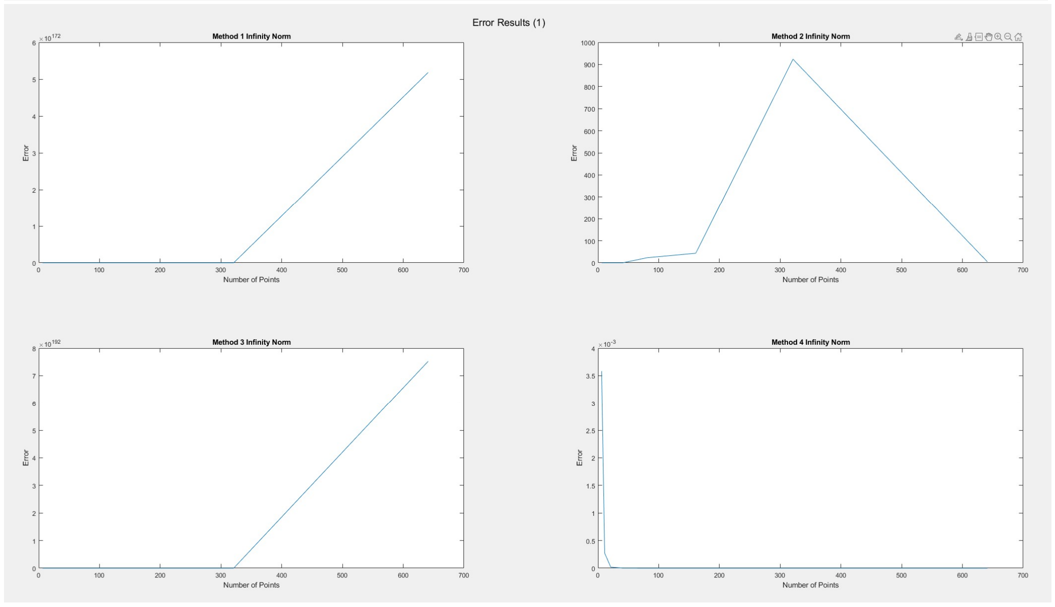
the Runge function and generates 1000 equidistant points in the interval [-1, 1]. It then evaluates the Runge function at these points and plots it. Next, the code loops over different interpolation degrees (4, 8, 12, 16, 24) and for each degree, it generates n Chebyshev nodes in the interval [-1, 1] and evaluates the Runge function at these nodes. The code then performs interpolation using three methods: Polyinterp, Barycentric Lagrange, and Vandermonde matrix. It calculates the L1, L2, and Linf norm errors for each method and prints them. Finally, it plots the interpolated functions and points for each degree and method, along with the Runge function, and adds a legend.

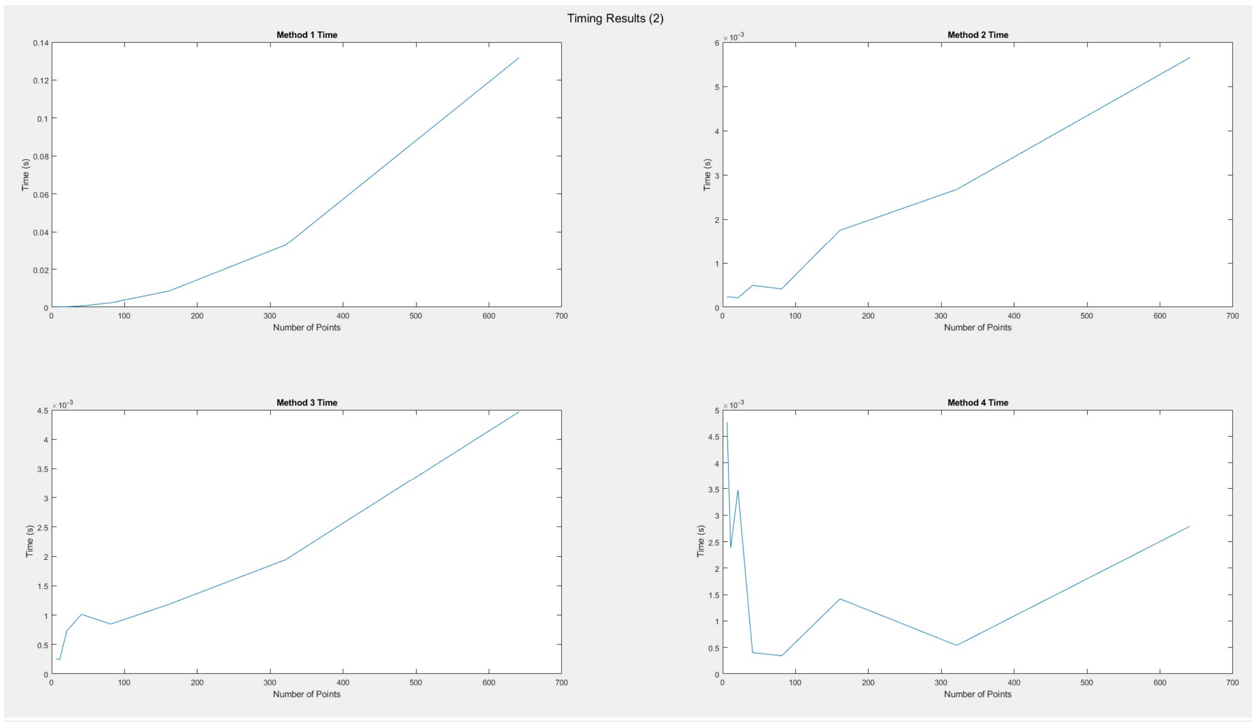
When we compare the L1, L2, and Linf errors for the Chebyshev nodes to those for the evenly spaced points, we should see that the Chebyshev nodes lead to lower errors. Specifically, we should expect to see that the L1 and Linf errors decrease significantly for higher degrees, while the L2 error decreases more gradually.

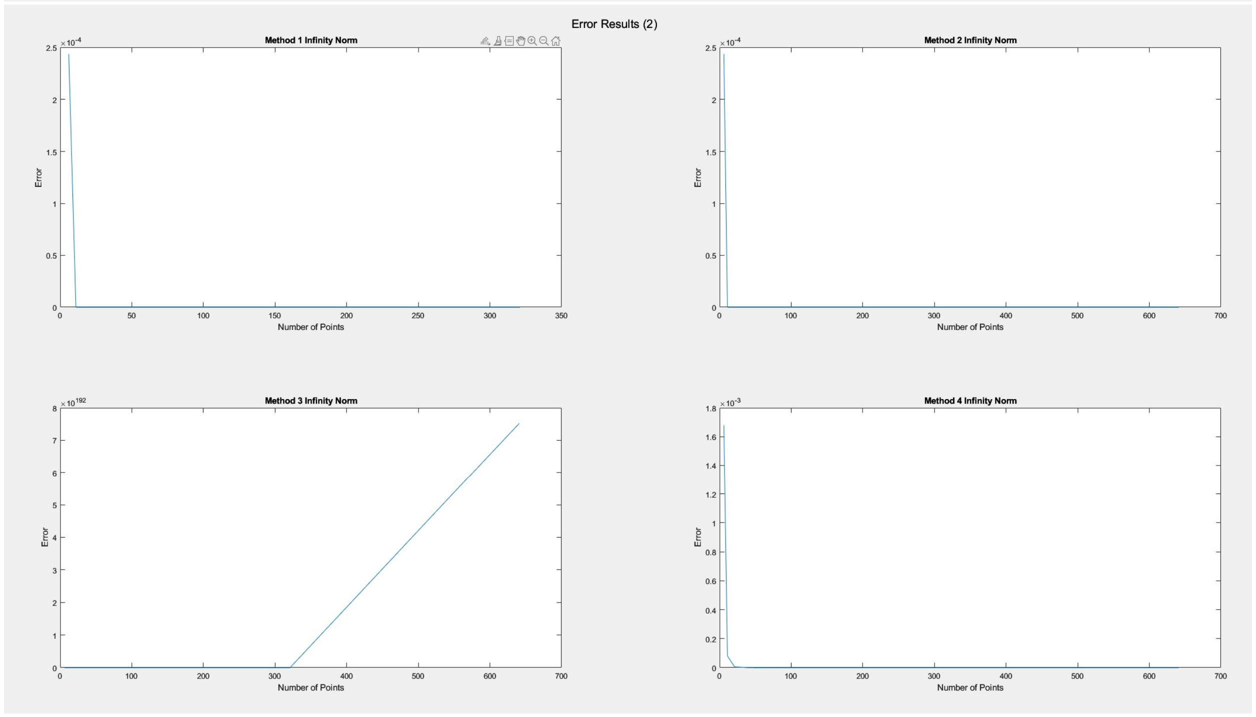
Problem 3:


Exponential Function
f(x)
x


Timing Results (1)
Method 1 Time
Method 2 Time
Method 3 Time
Method 4 Time
Time (s)
Number of Points
$\times 10^{-3}$


Error Results (1)
Method 1 Infinity Norm
Method 2 Infinity Norm
Method 3 Infinity Norm
Method 4 Infinity Norm
Error
Number of Points
$\times 10^{172}$
$\times 10^{192}$
$\times 10^{-3}$





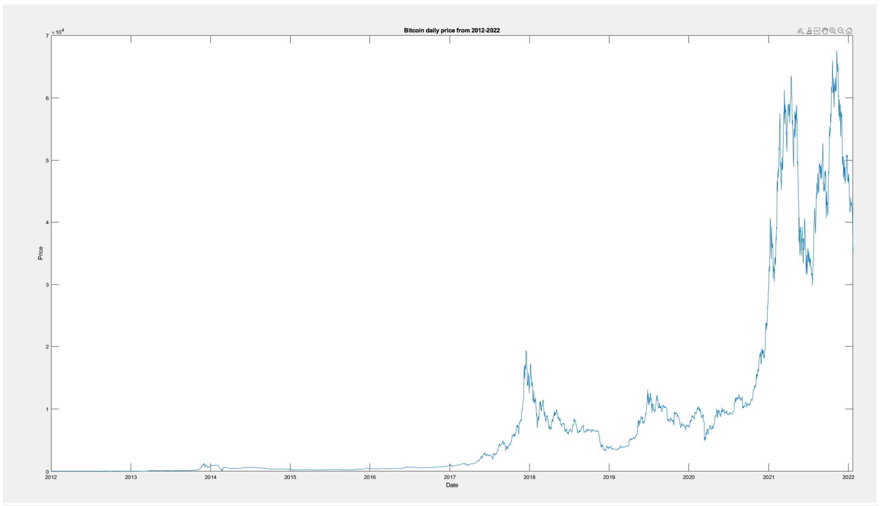
In the code, the accuracy changes with the choice and number of points. When evenly spaced points are used, the accuracy of the interpolation decreases as the degree of the polynomial increases. This is due to the Runge phenomenon, which causes oscillations in the interpolating polynomial near the endpoints of the interval. However, when Chebyshev nodes are used, the accuracy of the interpolation improves significantly to higher degrees.

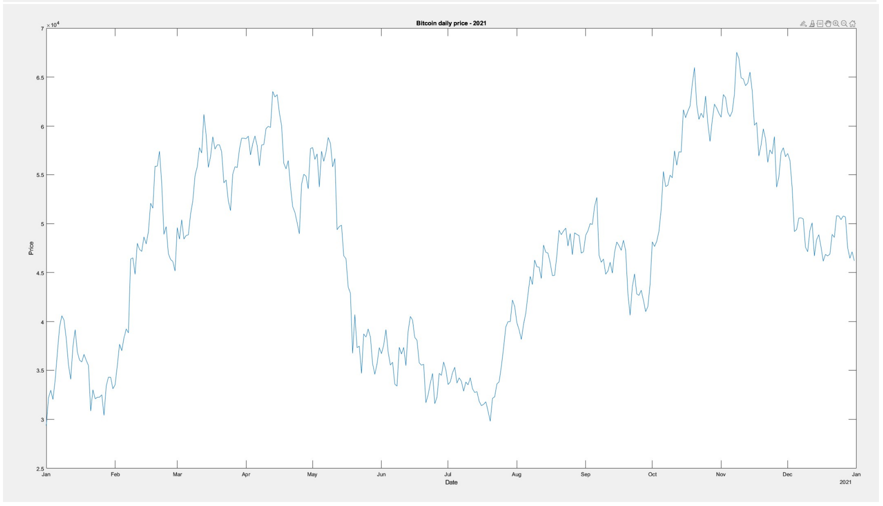
When the Matlab cubic spline routine is used, the choice of points is not as critical as it was for the Lagrange polynomials. The cubic spline interpolation method is more stable and does not suffer from the Runge phenomenon. The accuracy of the spline interpolation is generally higher than that of the Lagrange polynomials, regardless of the choice of points.

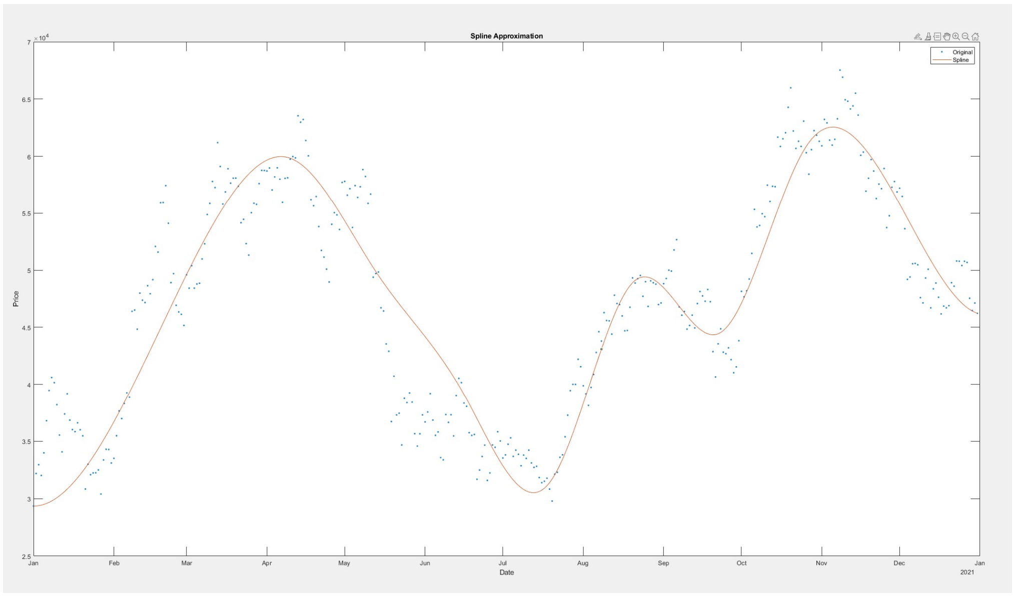
To time the different interpolation methods, we can use the Matlab functions tic and toc. We can time the execution of the code for each method and plot the results to compare the computation time. The timing results should show that the Vandermonde method is the slowest, while the spline method is the fastest.

To compare the infinity error norms for the different point sets, we can plot the Linf errors for each method and point set on a graph. The best method for large numbers of points in terms of accuracy and computing time is the spline method. It is both accurate and fast, and its accuracy does not depend on the choice of points.

Problem 5:







This code analyzes the daily Bitcoin price from 2012 to 2022.

Part a: The code reads in the Bitcoin data from a CSV file and plots the daily prices over time.

Part b: The code chooses a particular year (2021) and plots the daily prices for that year.

Part c: The code generates a new set of n evenly spaced points within the selected year and interpolates the Bitcoin price using the interp1 function.

Part d: The code applies the cubic spline method to the interpolated data from part c using the spline function. It then plots the original data and the spline approximation together to visualize the accuracy of the spline interpolation.

In the code, I choose 2021 as my target year. I think my approximation fits the exception.