Sizhuo(Cindy) Liu
Math 128A
October 9th, 2018

Programming Assignment 2 Write-Up

Part One: Write a program that carries out calculations for polynomial interpolation using the Lagrange formulas below:

$$P(x) = \sum_{j=0}^{n} f(x_j) L_j(x), \qquad L_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k},$$

Steps taken:

1. Create a function of the form function yout = interpolate1 (f, xin, xout)
   (Details: interpolate1.m)
   - The input f is a function handle
   - The input xin is a list of points for which the method will interpolate upon
   - The input xout is another list of points for which the formula of P(x) will evaluate upon
   - The output yout is the list of output values evaluated from the interpolating polynomial
   - Set up a first layer of for loop to iterate through every evaluation point (every element of the xout list) from j = 1 to j = the length of the list xout
   - Create two empty lists L_terms and f_terms to store the individual terms calculated from the formulas
   - Set up another for loop within the first for loop in order to iterate through all the k terms from the Lagrange formula displayed above
     - Set L to be equal to 1 so that after each iteration, L will be constantly updated by multiplying the new terms to L
   - Set up a third for loop within the second loop to iterate through all the j terms displayed in the formula for P (x)
     - To avoid including the fraction that contains k and j when k = j, set up a conditional statement in which if k ≠ j, the computations would continue according to the formula; but if k = j, the fraction corresponding to that term automatically becomes 1 (so that when multiplied, it would add no effect onto the cumulated L term).
   - As the second and third loops for j and k go on, the new terms after each iteration are added into the L_terms and f_terms lists created before the loops started
   - After getting two full lists of L_terms and f_terms, multiply each of the corresponding elements of the two lists (or matrices) according to the P (x) formula displayed above.
   - Apply the sum() function to compute the final result from the polynomial interpolation formula

2. Apply the function
   - Apply the program created above by plugging in specified input values for f, xin, and xout

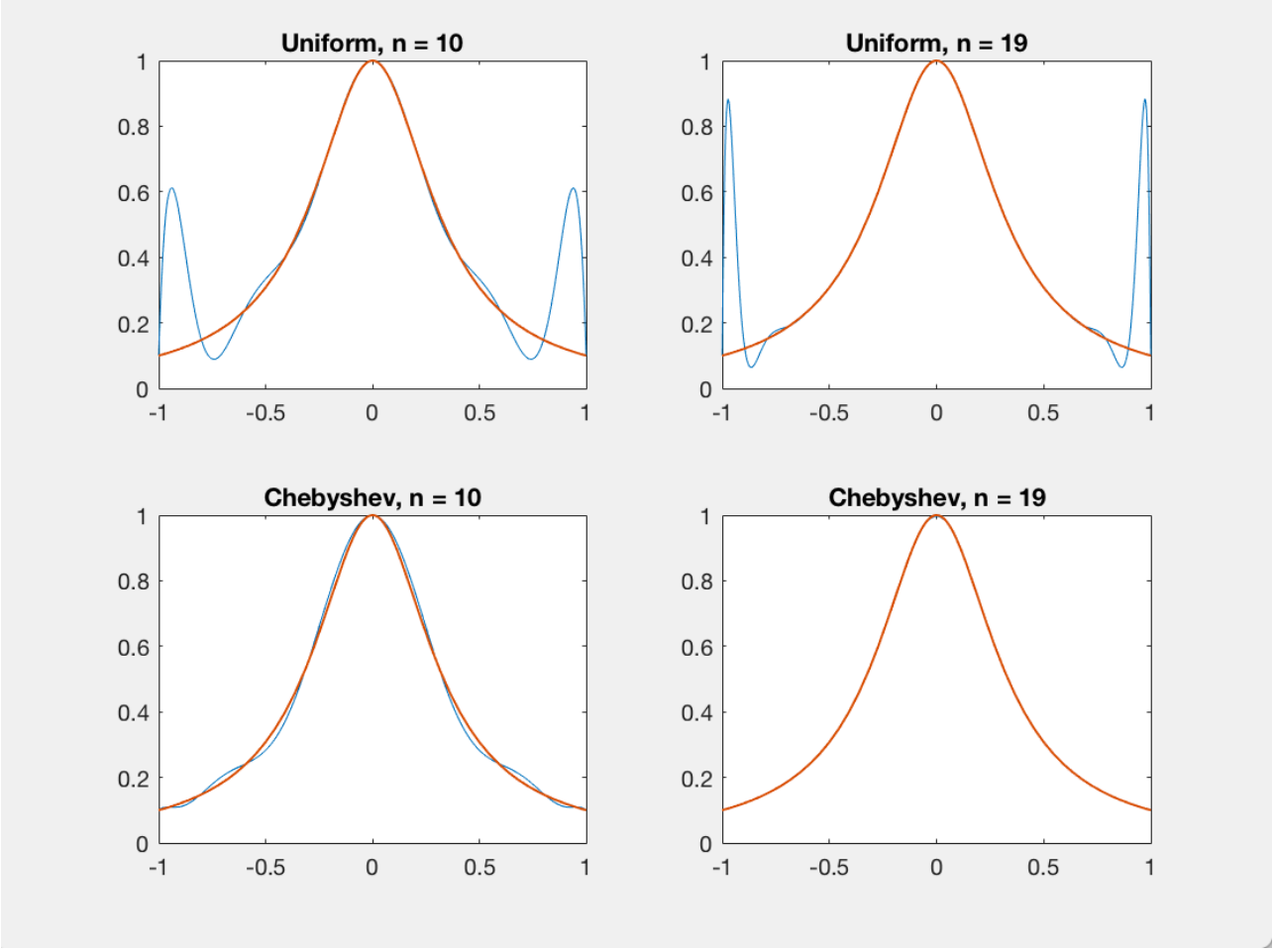
```
f = @(x) 1.0 ./ (1+9*x.^2)
xinU = linspace(-1,1,n+1)'          (uniform),
xinC = cos(linspace(-pi,0,n+1)')    (Chebyshev).
xout=linspace(-1,1,500)',
```

For values of n = 10, 19, 50, 99

3. Plot the graphs
   (Details: first_four_plots.m)
   - Create the first four plots for n ∈ {10, 19}, xin ∈ {xinU, xinC}, with f (x) and P (x) plotted on top of each other, using either of the two polynomial interpolation methods
   - Choose to use interpolate1 method (but Interpolate2 method should give the exact same output).
   - Set up the input function f, xinU, xinC, and xout in the same format as required (displayed above), with n = 10 and n = 19
   - Set up a new variable y = interpolate1 (f, xin, xout)
   - Plot the graph by implementing the MATLAB built-in function plot (xout, y)
   - Superimpose the function f (x) on top of the graph by adding in "hold on" and implement the MATLAB built-in function fplot (f, [-1, 1], 'Linewidth', 1)
   - Add a title through the built-in title() function
   - Do the same for the other three graphs to get four plots in total: xinU, n = 10; xinU, n = 19; xinC, n = 10; xinC, n = 19.
   - Implement subplot() to create four subplots in one graph
   - The final presentation of the four plots are shown on the following page

Observations:

- As shown by the four plots above, as the value of n increases, the polynomial Interpolation process becomes better as the curve for the Interpolating polynomial (blue) approaches the shape of the curve for the function (red).
- The Chebyshev grid points are clustered more closely at the endpoints of the interval
- By n = 19, The Chebyshev grid points almost completely overlap with the curve of the function
- With equal values of n, the Chebyshev grid points give better Interpolation results (the blue curve matches more closely to the red terms).



Part Two: Write a program that also carries out polynomial interpolation but using barycentric formula for interpolating polynomial as shown below:

$$P(x) = \begin{cases} f(x_j) & x = x_j \\ \sum_{j=0}^{n} \frac{\lambda_j f(x_j)}{x - x_j} \Big/ \sum_{j=0}^{n} \frac{\lambda_j}{x - x_j}, & x \notin \{x_0, \ldots, x_n\} \end{cases}, \qquad \lambda_j = \frac{1}{\prod_{k \neq j}(x_j - x_k)}.$$

Steps taken:

1. Create a function of the form function yout = interpolate2 (f, xin, xout)
   (Details: interpolate2.m)
   - The inputs and output are exactly the same as the function for interpolate1.
   - The overall structure of the loops are exactly the same as interpolate1 with a few areas of minor changes according to differences in the formula:
     - Two more empty lists numerator_terms and denominator_terms are created before the second and third loops start so that they can be used to store the individual terms of the numerator and denominator in the P (x) formula

- Add a conditional statement by the end of the second and third loops (before the first loop ends) so that the two different cases of P (x) as shown in the formula above can both be taken into account

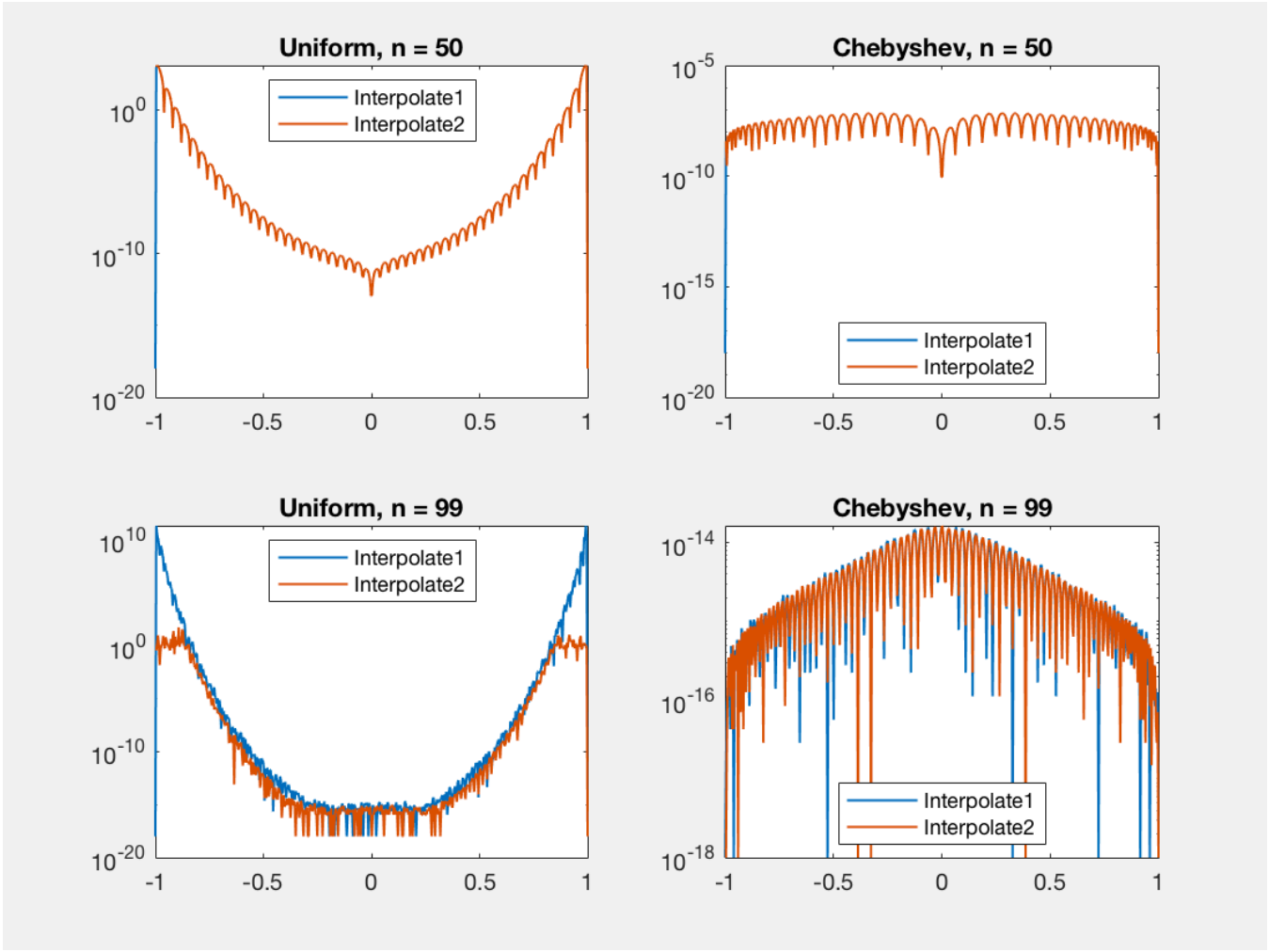
2. Plot the graphs

(Details: last_four_plots.m)

- Create the last four plots for n $\in$ {10, 19}, xin $\in$ {xinU, xinC}, plot
  semilogy (xout', 1.0e^-18 + abs([youtA1' – f(xout'), youtA2' – f(xout')]), 'linewidth', 1)
  where A is is either U or C and youtA1 and youtA2 are the set of results obtained from the two polynomial interpolation methods created earlier
- Set up the input function f, xinU, xinC, and xout in the same format as required (displayed above), with n = 50 and n = 99
- Since semilogy() itself is already a plot function, no other plot() function needs to be implemented for this section
- Do the same for the other three graphs to get four plots in total: xinU, n = 50; xinU, n = 99; xinC, n = 50; xinC, n = 99.
- Implement subplot() to create four subplots in one graph
- The final presentation of the four plots are shown on the following page

Observations:

- The below Semilog plots of error show the accuracy of the two methods of polynomial Interpolation by plotting the difference between the set of points obtained from the Interpolating polynomial and the set of points obtained from the actual function itself, or the absolute error
- The comparison between the graphs with different xin inputs and values of n indicates that:
  - As n becomes larger, the difference in accuracy between the two methods of polynomial Interpolation becomes smaller (the blue and red graphs overlap significantly more)
  - When applying the Chebyshev grid points, the absolute error tends to be the greatest at values of xout at around 0, whereas the absolute error is the lowest at around that point for the uniform grid points.
  - At n = 50, the error for the first Interpolation method is only displayed when xout is around -1.


Uniform, n = 50
Interpolate1
Interpolate2
10^{0}
10^{-10}
10^{-20}
-1
-0.5
0
0.5
1
Chebyshev, n = 50
10^{-5}
10^{-10}
10^{-15}
10^{-20}
Interpolate1
Interpolate2
Uniform, n = 99
10^{10}
10^{0}
10^{-10}
10^{-20}
Interpolate1
Interpolate2
Chebyshev, n = 99
10^{-14}
10^{-16}
10^{-18}
Interpolate1
Interpolate2