PHYS 581: Assignment #2

Scott Salmon UCID: 30093320

February 3rd, 2025

Question 1 (10 points)

The definition of a Lagrange Basis polynomial of degree $\leq$ k is:

$$\ell_j(x) = \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m}$$

$$\ell_j(x) = \frac{(x - x_0)}{(x_j - x_0)} \ldots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \ldots \frac{(x - x_k)}{(x_j - x_k)}$$

I wrote a function in Mathematica that generates a Lagrange Basis with a given set of x-points using these equations. This function can be found in "Assignment 2 - Mathematica Workbook.nb" file submission.

Let's do an example input to verify that the function is working as expected:

$$x_{points} = \{-1, 0, 1\}$$

$$\ell_1(x) = \frac{(x - x_2)}{(x_1 - x_2)} \frac{(x - x_3)}{(x_1 - x_3)}$$

$$\ell_1(x) = \frac{(x - 0)}{(-1 - 0)} \frac{(x - 1)}{(-1 - 1)}$$

$$\ell_1(x) = \frac{x\,(x - 1)}{2} = \frac{1}{2}\,(1 - x)\,x$$

$$\ell_2(x) = \frac{(x - x_1)}{(x_2 - x_1)} \frac{(x - x_3)}{(x_2 - x_3)}$$

$$\ell_2(x) = \frac{(x + 1)}{(0 + 1)} \frac{(x - 1)}{(0 - 1)}$$

$$\ell_2(x) = \frac{\left(x^2 - 1\right)}{-1} = 1 - x^2$$

$$\ell_3(x) = \frac{(x - x_1)}{(x_3 - x_1)} \frac{(x - x_2)}{(x_3 - x_2)}$$

$$\ell_3(x) = \frac{(x + 1)}{(1 + 1)} \frac{(x - 0)}{(1 - 0)}$$

$$\ell_3(x) = \frac{x\,(x + 1)}{2} = \frac{1}{2}\,(1 + x)\,x$$

The output for the function I created was as follows:

Out[38]= {-1, 0, 1}

Out[40]= $\left\{\frac{1}{2}\ (-1 + x)\ x,\ 1 - x^2,\ \frac{1}{2}\ x\ (1 + x)\right\}$

which matches our expected output, meaning we know the function is working properly.

Question 2 (50 points)

Question 2a (10 points)

Using Mathematica, I was able to obtain explicit expressions for the Lagrange basis polynomials and their first/second derivatives for the uneven grid positions given by Chevyshev nodes as shown below:

$$x_k = \cos\left[\frac{\pi}{m}\left(k - \frac{1}{2}\right)\right], \quad k = 1, ..., m$$

These results are found in the Question 2a cell in "Assignment 2 - Mathematica Workbook.nb".

Question 2b (10 points)

Using the explicit expressions from 2a, I derived mth order finite difference approximations for the first and second derivatives. I then evaluated these expressions at 0 and at 1, and calculated their stencil coefficients. All of these results are found in the Question 2b cell in "Assignment 2 - Mathematica Workbook.nb".

Using the following table of the stencil coefficients generated in Mathematica:

Out[28]=

k	L_k'(0)	L_k'(1)	L_k''(0)	L_k''(1)
1	0.0160359	23.8574	-1.55137	317.469
2	-0.0571854	-37.8314	4.96689	-680.353
3	0.141421	23.3179	-9.6	637.466
4	-0.432302	-16.0196	17.7217	-479.832
5	4.03604	11.5697	-11.5372	358.95
6	-4.03604	-8.46695	-11.5372	-267.203
7	0.432302	6.08586	17.7217	193.853
8	-0.141421	-4.11787	-9.6	-131.866
9	0.0571854	2.38809	4.96689	76.7003
10	-0.0160359	-0.783058	-1.55137	-25.1837

We can see that the stencil coefficients do indeed change when we change x. Near x=0, the coefficients are much smaller, meaning that they are more balanced and supporting of a stable and symmetric stencil. However, near x=1, the coefficients are much larger, meaning the stencil becomes biased and less stable near the boundary.

These results do indeed make sense, as Chebyshev nodes are *more densely packed* near the center ($x \approx 0$) and more widely spaced near the boundaries ($|x| \approx 1$). As finite-difference approximations rely on neighboring points, its intuitive that results will be better when the difference between points is smaller (as was extensively proven in Assignment # 1...), meaning it makes sense that the stencil coefficients are much lower as $x \to 0$.

While not explicitly shown here, I did also do some testing using -1 for both L'_k and L''_k, and the results were identical as when I used 1, except in reverse (i.e. the result for k=2 for $L'_k(1)$ is the same as k=9 for $L_k(-1)$). This is expected behaviour, as the grid we're using is inherently periodic.

Question 2c (15 points)

Again, using the explicit Lagrange Basis Polynomial expressions derived in 2a, I obtained interpolated approximations to the Chebyshev polynomials and their derivatives for all $1 \leq n \leq m$. I then compared the interpolated results to the "real" results through the use of plots. All of these results are found in the Question 2c cell in "Assignment 2 - Mathematica Workbook.nb".

As is defined by Equation (1.21) in the "Course Lecture Notes", the reason we're able to create these interpolation approximations is due to the very useful delta-function property of basis polynomials:

$$L(x_k) = \sum_j a_j \ell_j(x_k) = \sum_j a_j \delta_{jk} = a_k$$

Using the Chebyshev nodes, x_k, from Question 2a, we sample Chebyshev polynomial $T_n(x)$ at these points. We then use these samples to construct Lagrange interpolations, defined as $L_n(x)$, which approximate $T_n(x)$, $T'_n(x)$, and $T''_n(x)$. This follows from the delta-function property ensuring that the interpolation exactly matches $T_n(x)$ at the chosen nodes.

$$x_k = cos\left[\frac{\pi}{m}\left(k - \frac{1}{2}\right)\right], \quad k = 1,\ ...,\ m$$

$$T_n(x) \approx L_n(x) = \sum_{k=1}^{m} T_n(x_k)\ell_k(x)$$

$$T'_n(x) \approx L'_n(x) = \sum_{k=1}^{m} T_n(x_k)\ell'_k(x)$$

$$T''_n(x) \approx L''_n(x) = \sum_{k=1}^{m} T_n(x_k)\ell''_k(x)$$

From the plots in the Mathematica workbook, we can see that when we apply the uneven grid points provided by Chebyshev nodes to this interpolation function, the final results are very close to the true polynomials. The $T''_1(x)$ plot has a lot of visual noise, however the noise has a magnitude of around 1.5×10^{-13}, which is low enough to be considered equal to zero.

Interestingly, the interpolated results when $n = m$ are all also approximately equal to zero. When I was initially writing my code, I was sure that it was an error on my part, but after further inspection, I realized that this is mathematically expected. Lagrange interpolate functions are all of degree $m-1$, so when $n = m$ occurs on an irregular grid, all of the interpolates will end up being equal to 0. This explains why for the last plot in the workbook, the interpolated results are zero instead of matching the true polynomial function.

Question 2d (15 points)

This question is a complete repeat of Question 2c, except instead of using the uneven grid of Chebyshev Nodes like we did previously, we instead had to use a evenly-spaced grid. All of these results are found in the Question 2d cell in "Assignment 2 - Mathematica Workbook.nb".

For my new set of points, I just made an evenly spaced grid of 10 points from -1 to 1. I then had to make a new set of sample points, $T_n(x_k)$, with the new grid of points and then I found the interpolations and compared them to the true results again.

The results for $1 \leq n \leq m$ were exactly the same as with the even grid. The result at $n = m$ however were different. Unlike with the even grid, there was actually results, but the results were bad with large amounts of error.

This is expected behaviour, as for evenly spaced grids can produce large oscillations near the endpoints. This is known as Runge's phenomenon. The delta-function property of Lagrange interpolations still holds at the actual interpolation nodes, but global accuracy suffers - dramatically.

Question 3 (40 points)

Question 3a (10 points)

For this question, I had to write a quick program that could calculate the barycentric weights from an arbitrary set of grid points. This is an application of Equation (1.31) from the "Course Lecture Notes":

$$w_j \equiv \frac{1}{\prod_{i=1}^{N}(x - x_j)}$$

My commented C code for this can be found in the submitted file named "question3a.c".

Question 3b (10 points)

This question uses the function that I initially made in Question 3a to obtain the interpolated approximations for the Chebyshev polynomials again, similar to what we did in Question 2. However, this time instead of using Equation (1.21), we will instead use Equation (1.36), which incorporates the weights for the grid points:

$$f(x) = \frac{\Sigma_j \frac{w_j a_j}{x - x_j}}{\Sigma_j \frac{w_j}{x - x_j}}$$

To do this, I used the same Chebyshev nodes we used in Question 2, x_k:

$$x_k = cos\left[\frac{\pi}{m}\left(k - \frac{1}{2}\right)\right], \quad k = 1,\ ...,\ m$$

And I applied these points to the original closed form definition of the Chebyshev polynomials to create our sample points a_j (because unlike before in Mathematica, I couldn't use the built in ChebyshevT function):

$$T_n(x) = \cos[n \cos^{-1}(x)]$$

I then used print statements to compare the results. The results are nearly the exact the same for all points for all n except when $n = m$. At this point, all of the interpolated results were equal to 0, exactly as what happened in Question 2. As was explained beforehand, this is expected behaviour of the interpolations when we're using an uneven grid.

My commented C code for this can be found in the submitted file named "question3b.c".

Question 3c (20 points)

To solve the problem outlined in this question, we first need to find the derivative of Equation (1.36). The easiest way to do this is to define the numerator as N and the denominator as D. For the weight-interpolation equation (Eq 1.36), this means that:

$$N = \sum_j^m \frac{w_j a_j}{x - x_j}$$

$$D = \sum_j^m \frac{w_j}{x - x_j}$$

Now, using the quotient rule, we know that:

$$f'(x) = \frac{N' \cdot D - N \cdot D'}{D^2}$$

Using Mathematica, I was able to easily find the derivatives N' and D'.

$$N' = \sum_j^m -\frac{w_j a_j}{(x - x_j)^2}$$

$$D' = \sum_j^m -\frac{w_j}{(x - x_j)^2}$$

Now, you could totally substitute these into the quotient rule equation and find a nice expression for the derivative (and I actually did do that on Mathematica)... but for my implementation in C, I was lazy and instead of combining everything, I instead individually found N, N', D, and D' for each iteration, and then combined them in the quotient rule to find the derivative for each calculation. I did this because I didn't want to try and implement the bigger equation, and this still works.

With that out of the way, I essentially copy-pasted my code from question3b.c, added a function that finds the derivative interpolation using the method that was described above, and then also added a function to find the true derivative points of the Chevyshev Polynomial. To do this, I simply took the derivative of the closed form expression:

$$T'_n(x) = \frac{n \sin[n \cos^{-1}(x)]}{\sqrt{1 - x^2}}$$

I then used print statements to compare the results. The results are nearly the exact same for all **non-boundary** points for all n except when $n = m$, for the same reasons as mentioned before.

The reason that the boundaries are not the same is because for the true results, the derivative at the boundaries is actually undefined (as ± 1 makes denominator in the derivative function equal to 0, as can be seen in the $T'_n(x)$ equation above). However, this 1/0 issue does not happen for the interpolation function, so it will still spit out an approximation for 1 and -1, even though there is no true value at those points.

Outside of the boundary points, and when $n \neq m$, the interpolation function produced nearly identical results to the real function.

My commented C code for this can be found in the submitted file named "question3c.c". Additionally, the required derivative calculation is found in the Question 3c cell in "Assignment 2 - Mathematica Workbook.nb".