Homework 5

Henry Dyer

April 6, 2024

* All code is in the 'HW5/4600hw5.py' file pushed to GitHub

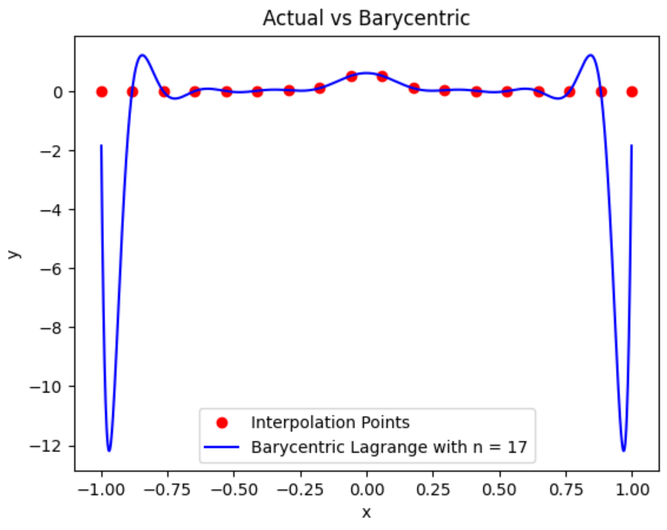
1. In this problem we find that the polynomial $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$ that interpolates the data $(x_j, y_j) = (x_j, f(x_j)), j = 0, \ldots, n$.

   (a) Write down a pseudocode for a python function that uses barycentric Lagrange to evaluate the interpolant at target points, that is, $p(z)$. Briefly explain which of these formulas should be used, and why.

```
Algorithm 1: Barycentric Lagrange Interpolation
  Input: Data points $(x_j, y_j) = (x_j, f(x_j)), j = 0, \ldots, n,$ and point to eval $z$
  Compute weights:
  for $i = 0$ to $n$ do
      $w_i = \frac{1}{\prod_{j \neq i}(x_i - x_j)}$;
  Evaluate p(z):
  $p(z) = \frac{\sum_{i=0}^{n} y_i \frac{w_i}{x - x_i}}{\sum_{i=0}^{n} \frac{w_i}{x - x_i}}$
```

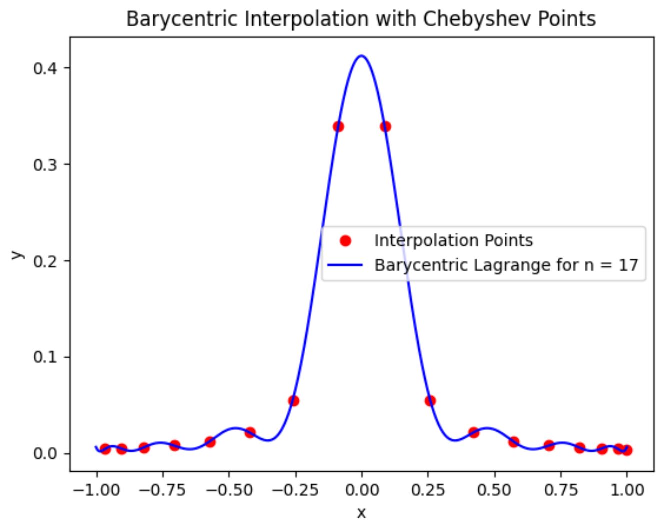
Here we use the formulas for the second barycentric Lagrange interpolation because it does not require computing $\psi(x)$, furthermore the weights can be scaled if necessary to improve stability.

(b) Plot data points as circles and, in the same plot, plot the polynomial and $f(x)$ on a finer grid (still on $x \in [-1, 1]$), say with 1001 points. Observe what happens when you increase n. Try $n = 1, 2, 3, \ldots$ and continue until the maximum value of $p(x)$ is about 100. As you can see the polynomial behaves badly near the endpoints of the interval due to Runge's phenomena.



As we increase n the Runge phenomena becomes worse near the ends points.

(c) Change the input interpolation points in you code to Chebyshev nodes. Perform the same tests as above. Briefly note the differences between the errors for these two kinds of interpolation points.



Obviously the Runge phenomena is not present when Chebyshev nodes are used. This image has $n = 17$ which was the same as in the original graph, infact with the Chebyshev nodes we can continue increasing n and do not encounter the Runge phenomena.

(d) How does this help explain how Chebyshev nodes avoid the Runge phenomenon?

Chebyshev nodes avoid the Runge phenomena by clustering interpolation points towards the ends of the interval as opposed to spacing them out equally. Since these points are clustered towards the boundary where the polynomials will oscillate more, it in a way adds weight to approximations that are suitable near the boundary points which in turn helps mitigate the Runge phenomena.

2. Recall how we build the Lagrange basis for the standard and Hermite interpolation problems. Say we now have the following "mixed" polynomial interpolation problem: We want to find a quadratic polynomial $p(x)$ defined on $[-1, 1]$ such that $p(-1) = y_0, p(1), = y_1, p'(1) = z_1$.

   (a) Find a Lagrange basis for this problem. That is, find quadratics $L_0(x), L_1(x), L_2(x)$ such that they each satisfy one condition equal to 1 and the rest equal to 0.

   We begin by defining the following Lagrange polynomials:

$$L_0(-1) = 1, L_0(1) = 0, L_0'(1) = 0$$
$$L_1(-1) = 0, L_1(1) = 1, L_1'(1) = 0$$
$$L_2(-1) = 0, L_2(1) = 0, L_2'(1) = 1$$

   since these are all quadratics, they must be of the form $a_ix^2 + b_ix + c_i$ so we can expand each of these above equations as follows

$$\begin{aligned}
L_0(-1) &= a_0(-1)^2 + b_0(-1) + c_0 = a_0 - b_0 + c_0 = 1 \\
L_0(1) &= a_0(1)^2 + b_0(1) + c_0 = a_0 + b_0 + c_0 = 0 \\
L_0'(1) &= 2a_0(1) + b_0 = 2a_0 + b_0 = 0
\end{aligned}$$

$$\begin{aligned}
L_1(-1) &= a_1(-1)^2 + b_1(-1) + c_1 = a_1 - b_1 + c_1 = 0 \\
L_1(1) &= a_1(1)^2 + b_1(1) + c_1 = a_1 + b_1 + c_1 = 1 \\
L_1'(1) &= 2a_1(1) + b_1 = 2a_1 + b_1 = 0
\end{aligned}$$

$$\begin{aligned}
L_2(-1) &= a_2(-1)^2 + b_2(-1) + c_2 = a_2 - b_2 + c_2 = 0 \\
L_2(1) &= a_2(1)^2 + b_2(1) + c_2 = a_2 + b_2 + c_2 = 0 \\
L_2'(1) &= 2a_2(1) + b_2 = 2a_2 + b_2 = 1
\end{aligned}$$

   all of which are linear systems which can be easily solved in NumPy and yield the following quadratic polynomials for $L_0(x), L_1(x), L_2(x)$:

$$\begin{aligned}
L_0(x) &= \frac{1}{4}x^2 - \frac{1}{2}x + \frac{1}{4} \\
L_1(x) &= -\frac{1}{4}x^2 + \frac{1}{2}x + \frac{3}{4} \\
L_2(x) &= \frac{1}{2}x^2 - \frac{1}{2}
\end{aligned}$$

   (b) Using this Lagrange basis, write down a formula for the interpolant p(x) given data y_0, y_1, z_1.

$$p(x) = y_0(\frac{1}{4}x^2 - \frac{1}{2}x + \frac{1}{4}) + y_1(-\frac{1}{4}x^2 + \frac{1}{2}x + \frac{3}{4}) + z_1(\frac{1}{2}x^2 - \frac{1}{2})$$

(c) Using standard and Hermite interpolation as inspiration, indicate what would be a Newton basis for this problem.

Starting from the form of a standard Newton basis

$$\begin{aligned} p(x) &= a_0 + a_1(x+1) + a_2(x+1)(x-1) \\ &= a_0 + a_1x + a_1 + a_2x^2 - a_2 \\ &= a_2x^2 + a_1x + (a_0 + a_1 - a_2) \end{aligned}$$

we can differentiate this to determine $p'(x)$

$$p'(x) = 2a_2x + a_1$$

which gives us three equations and three unknowns

$$\begin{aligned} y_0 &= p(-1) = a_2(-1)^2 + a_1(-1) + (a_0 + a_1 - a_2) = a_0 \\ y_1 &= p(1) = a_2(1)^2 + a_1(1) + (a_0 + a_1 - a_2) = a_0 + 2a_1 \\ z_1 &= p'(1) = 2a_2(1) + a_1 = a_1 + 2a_2 \end{aligned}$$

solving these for the constants a_i gives us the following equivlances

$$\begin{aligned} a_0 &= y_0 \\ a_1 &= \frac{y_1 - y_0}{2} \\ a_2 &= \frac{2z_1 + y_0 - y_1}{4} \end{aligned}$$

substituting this back into our initial polynomial, $p(x)$, we get

$$\begin{aligned} p(x) &= y_0 + \left(\frac{y_1 - y_0}{2}\right)(x+1) + \left(\frac{2z_1 + y_0 - y_1}{4}\right)(x^2 - 1) \\ &= \left(\frac{2z_1 + y_0 - y_1}{4}\right)x^2 + \left(\frac{y_1 - y_0}{2}\right)x + \left(\frac{y_0 + 3y_1 - 2z_1}{4}\right) \end{aligned}$$

3. We now consider the problem of interpolating the data $(x_j, y_j) = (x_j, f(x_j)), j = 0, \dots, n$ using a cubic spline.

   (a) Explain the difference between a natural and a periodic cubic spline.

   For a natural cubic spline, it is assumed that the second derivative is zero at the end points. For a periodic cubic spline, it is assumed that the first and second derivative, are equal at the two end points.

   (b) Consider the derivation in class of an algorithm to obtain the coefficients for a natural cubic spline. Indicate what step would need to be modified if, instead, you wanted to obtain a representation for a periodic cubic spline.

   (c) Use code provided in modules or a SciPy routine to interpolate the periodic function $f(x) = \sin(9x)$ using periodic cubic splines with $n = 5, 10, 20, 40$ equispaced points in $[0, 1]$. Plot the logarithm of the interpolation error for each one, and briefly indicate what you observe.

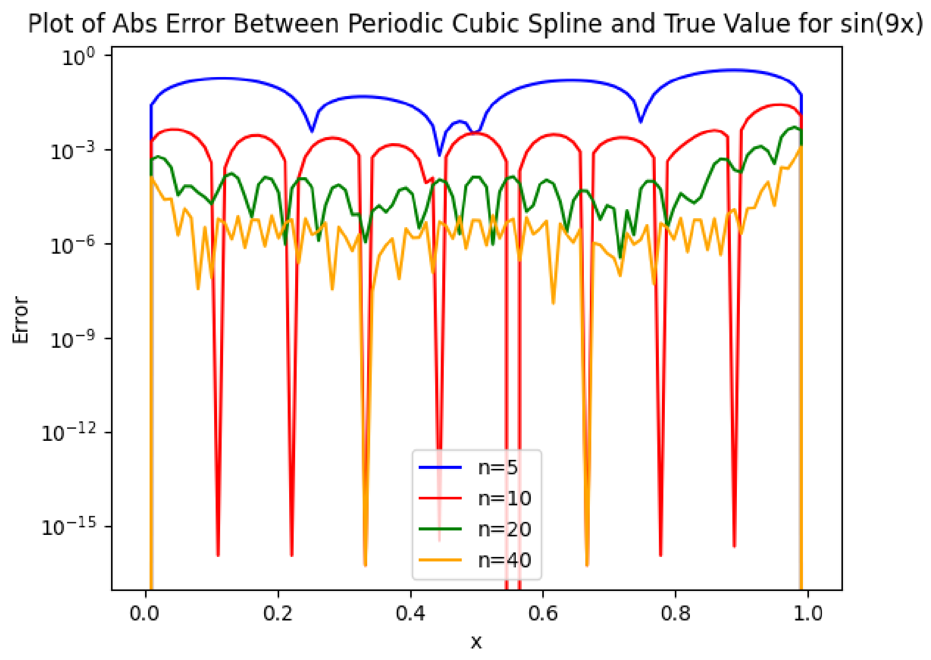


Figure 1: Enter Caption

As we can see, the largest errors seen for each approximation gets better as the number of interpolating nodes grows. The error at the best approximations also tends to get larger as the number of interpolating nodes grows as well.

4. Consider the following problems using discrete least squares:

   (a) Say we have four data points $(0,1),(1,4),(2,2),(3,6)$. We want to find the linear polynomial $p(x) = a_1 x + a_0$ that best fits this data in the least squares sense. Write down the least squares optimization problem, and find the system of normal equations $\mathbf{G}a = \mathbf{b}$. Solve this system to find the coefficients a_0 and a_1.

   First we solve $\min_x \|Ax - b\|_2^2$ for the euclidian norm:

$$\begin{aligned}\min_x \|Ax-b\|_2^2 &= \langle Ax-b, Ax-b\rangle \\ &= (Ax-b)^T(Ax-b) \\ &= x^T A^T Ax - b^T Ax - x^T A^T b + b^T b\end{aligned}$$

   We want to minimize this and when $A^T A$ is semi-positive definite, this is strictly convex so we take the gradient and set the result equal to zero.

$$\begin{aligned}0 &= \nabla(x^T A^T Ax - b^T Ax - x^T A^T b + b^T b) \\ &= 2A^T Ax - 2A^T b\end{aligned}$$

   so we derive the normal equations $A^T Ax = A^T b$. With the given points, we want to solve the following system $\mathbf{A}a = \mathbf{b}$:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 6 \end{bmatrix}$$

   assuming there is no solution to this system, we turn to the least squares approach where we solve the normal equations $\mathbf{G}a = b$ where $\mathbf{G}^{-1} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$:

$$\begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \left(\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 2 \\ 6 \end{bmatrix}$$

   which produces the solutions to the least squares optimization $p(x) = 1.3x + 1.3$.

(b) We now want to consider a weighted least squares approach. Let $\mathbf{D}$ be a 4×4 diagonal matrix with entries $\mathbf{D}(i,i) = \sqrt{w_i}$. Show that q_w can be written in the form $\left\|\mathbf{D}(Ma - y)\right\|^2$. What are the normal equations for this problem?

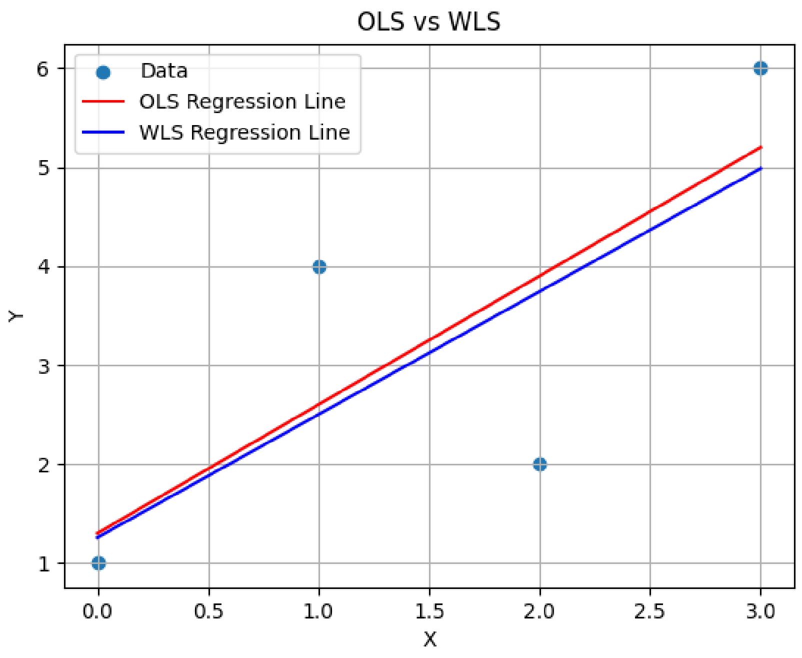
In the same manor as the previous work, we can expand the equations of $\min_x \left\|\mathbf{D}(Ax - b)\right\|^2$ to acquire the normal equations $A^T DAx = A^T Db$ since $Ax = b$ leads to $DAx = Db$

(c) Using the information above, find the coefficients for the weighted least squares problem. Compare plots for both lines against the given data, and explain intuitively what the effect of the weights is on the linear fit.

Here the normal equations $\mathbf{G}a = \mathbf{b}$ have the form $a = \mathbf{G}^{-1}\mathbf{b} = (\mathbf{A}^T\mathbf{D}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{D}\mathbf{b}$:

$$\begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \left(\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & \sqrt{6} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & \sqrt{6} \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 2 \\ 6 \end{bmatrix}$$

which produces the solution to the weighted least squares optimization problem $p(x) = 1.24x + 1.26$.


OLS vs WLS
Data
OLS Regression Line
WLS Regression Line
Y
X