CAAM 453/550: Numerical Analysis I - Fall 2021
Homework 8 - due by 5pm on Wednesday, October 27, 2021

Instructions: You may use any of the code on the Canvas page. Turn in all modified/new MATLAB/python code (scripts and functions) and all output generated by your code. All code must be commented, and it must be clear what your output is and why you are submitting it. Additionally, all plots must be labeled.

For any problems that do not require coding, either turn in handwritten work or typeset work using LaTeX or some other typsetting software. Please do not turn in math as commented MATLAB code or math that has been typed in a word processor!

MATLAB code fragments are provided in some problems. If you program in python, you may replace them corresponding numpy or scipy code.

CAAM 453 students are to complete problems 1,2, 4, 5. (110 points)
CAAM 550 students are to complete problems 1 - 5. (140 points)

CAAM 453 may complete additional problems "for fun," but you will not receive additional credit.

Problem 1 (40 points) Pledged! Complete this problem on your own. Read through the entire problem before beginning, there may be ways to do things more efficiently than just going straight through.

(a) (10 points) Write a MATLAB/Python function, which given n data points (x_j, f_j) computes the coefficients of the unique $(n-1)$st degree polynomial passing through those points using a *Newton basis*. The function should take a form similar to

```
a = NewtonInterpolate(x,f)
```

where `x` and `f` are vectors of the same length n, containing $x_1, \ldots, x_n$ and $f_1, \ldots, f_n$ respectively. Your function should return a length n vector containing the coefficients of the polynomial interpolant arranged from lowest-order to highest-order. In other words, if

$$p(x) = \sum_{j=1}^{n} a_j \prod_{k=1}^{j-1} (x - x_k)$$

the first entry of `a` should be a_1 and the last should be a_n.

(b) (7 points) We need to evaluate the polynomial from (a). An efficient way to do this is via Horner's scheme, which organizes the polynomial calculation so as to minimize the number of arithmetic operations required. If

$$p(x) = \sum_{j=1}^{n} a_j \prod_{k=1}^{j-1} (x - x_k)$$

($\prod_{k=1}^{0} = 1$), then

$$p(x) = (\cdots(((a_n(x - x_{n-1}) + a_{n-1})(x - x_{n-2}) + a_{n-2})(x - x_{n-3}) + \cdots + a_2)(x - x_1) + a_1,$$

requiring just n multiplications and $2n$ additions for each x. Write a function to evaluate polynomials using Horner's scheme, given the coefficients as `a`, the output of the function from part (a), and a vector of input points `x`. It should return a vector containing the values of the polynomial at each point in `x`. Your code should be vectorized.

(c) (7 points) Write a function that updates interpolating polynomials. It should take the coefficients of an existing $(n-1)$st degree interpolating polynomial, the points $\{x_j\}_{j=1}^n$, and a new point (x_{n+1}, f_{n+1}), and return the vector of coefficients of the nth degree polynomial passing through all the existing points as well as (x_{n+1}, f_{n+1}). If x_{n+1} is equal to one of the original interpolation points, $x_1, \ldots, x_n$, the function should return an error message.

(d) (5 points) Test out your function from (a) in the following ways: for $n = 5, 10, 20$, let $\{x_j\}_{j=1}^n$ be equally spaced points in the interval $[-2, 4]$. For each n compute the polynomial interpolant for three different functions:

$$f_j = e^{-x_j^2}, \qquad f_j = \sin(2x_j), \qquad \text{a function of your choice}$$

For each function, make a plot of the original function and the polynomial interpolants at each value of n. Use the function from part (b) to evaluate the polynomials. Include legends.

(e) (5 points) Apply your function from (a) to Runge's function, $f(x) = 1/(1+x^2)$ at 15 equally spaced points in the interval $[-3, 3]$. Plot $f(x)$ and the polynomial interpolant $p(x)$. You should see that the interpolant exhibits Runge phenomena. Choose additional points in the interval $[-3, 3]$ to try to counteract the phenomena and add them to the polynomial interpolant using the code from part (c). Create a plot of f, the original interpolant p, and the improved interpolant. Include a legend.

(f) (6 points) For $n = 1, 2, \ldots, 20$, compute the nth degree polynomial interpolants for each of the three functions in part (d) on $[-2, 4]$ using both Chebyshev points and equally spaced points. For each interpolant, calculate the approximate L^∞ error ,

$$\|f - p\|_{L^\infty([a,b])} = \max_{x \in [-2,4]} |f(x) - p(x)|,$$

by using 3001 equally spaced points in $[-2, 4]$. Make a semi-log plot showing the L^∞ error versus n for the Chebyshev points and equally spaced points.

Problem 2 (10 points)

Let $f : [a, b] \to \mathbb{R}$ be monotone and suppose that $f(a)f(b) < 0$. The intermediate value theorem guarantees the existence of $x_* \in (a, b)$ with $f(x_*) = 0$. In this problem you will use a process called *inverse interpolation* to compute x_*.

Since f is monotone, its inverse f^{-1} exists and the desired root x_* is $x_* = f^{-1}(0)$. We want to use polynomial interpolation applied to f^{-1} to compute an approximation of $f^{-1}(0)$.

(a) (8 points) Given n pairs $(x_i, f(x_i))$, $i = 1, \ldots, n$, write a program that computes the polynomials p_k of degree $k-1$ in Newton basis, that interpolate f^{-1} at $f(x_1), \ldots, f(x_k)$, and compute $p_k(0)$, $k = 1, \ldots, n$.

You may use your code from Problem 1. Your program should return a table with k and $p_k(0)$.

(b) (2 points) Consider $f(x) = \cos(x)\cosh(x) + 1$ on $[a, b] = [1.6, 2.1]$. Apply your code from Part (a) nodes $y_i = f(x_i)$, $x_i = a + (i-1)(b-a)/(n-1)$, $i = 1, \ldots, n$, to compute an approximation of the root x_* of f.

For $n = 2, 4, 6$ print the table generated by your algorithm from Part (a). What is the approximation of the root?

Problem 3 (30 points) CAAM 550 only. We want to interpolate the function $f : [a,b] \times [c,d] \to \mathbb{R}$ at points $(x_i, y_j) \in [a,b] \times [c,d]$, $i = 1,\ldots,n$, $j = 1,\ldots,m$. Specifically, let $p_1,\ldots,p_n$ be polynomials that form a basis of the space of polynomials of degree less or equal to $n-1$ and let $q_1,\ldots,q_m$ be polynomials that form a basis of the space of polynomials of degree less or equal to $m-1$. We want to determine coefficients a_{kl}, $k = 1,\ldots,n$, $l = 1,\ldots,m$ such that

$$\sum_{k=1}^{n}\sum_{l=1}^{m} a_{kl} p_k(x_i) q_l(y_j) = f(x_i, y_j), \quad i = 1,\ldots,n,\ j = 1,\ldots,m. \tag{1}$$

(a) (10 points) Let

$$p_i(x) = L_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 1,\ldots,n, \quad q_i(y) = L_i(y) = \prod_{\substack{j=1 \\ j\neq i}}^{m} \frac{y - y_j}{y_i - y_j}, \quad i = 1,\ldots,m$$

be the Lagrange polynomials. Describe how the coefficients a_{kl} in (1) can be determined.

Write a Matlab/python program that interpolates

$$f(x,y) = (1+x^2)^{-1}(1+y^2)^{-1}$$

at

$$x_i = y_i = -5 + 10\frac{i-1}{4},\ i = 1,\ldots,5$$

using Lagrange polynomials. Plot the interpolating polynomial.

(b) (20 points) Let

$$p_i(x) = N_i(x) = \prod_{j=1}^{i-1}(x - x_j), \quad i = 1,\ldots,n, \quad q_i(y) = N_i(y) = \prod_{j=1}^{i-1}(y - y_j), \quad i = 1,\ldots,m$$

be the Newton polynomials. Describe how the coefficients a_{kl} in (1) can be determined.

Write a Matlab/python program that takes inputs $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and a matrix $F \in \mathbb{R}^{n\times m}$ of function $F_{ij} = f(x_i, y_j)$ values and which returns $A \in \mathbb{R}^{n\times m}$ with the coefficients a_{ij} in (1) using the Newton polynomials.

Apply your Matlab/python function to interpolate

$$f(x,y) = (1+x^2)^{-1}(1+y^2)^{-1}$$

at

$$x_i = y_i = -5 + 10\frac{i-1}{4},\ i = 1,\ldots,5.$$

Print the matrix with $A \in \mathbb{R}^{5\times 5}$ with the coefficients a_{ij} and plot the interpolating polynomial.

Problem 4 (30 points) The goal of this problem is to introduce the Lebesgue constant. This constant is a "quality measure" for a set of interpolation points.

We need to introduce a new kind of norm.

> For functions $f \in \mathcal{C}([a,b])$, we define the L^p norm for integers $p \geq 1$ as
>
> $$\|f\|_{L^p([a,b])} = \left(\int_a^b |f(x)|^p \, \mathrm{d}x \right)^{1/p},$$
>
> and the L^∞ norm as
>
> $$\|f\|_{L^\infty([a,b])} = \max_{x \in [a,b]} |f(x)|.$$
>
> It can be shown that these quantities satisfy the properties that define a norm. They can be thought of as the continuous versions of the vector norms we discussed in class.

Let Π_n denote the linear operator that maps $f \in \mathcal{C}([a,b])$ to the polynomial p_n that interpolates f at the points

$$a \leq x_1 < x_1 < \cdots < x_{n+1} \leq b.$$

In other words, $\Pi_n f = p_n$, where p_n is the unique polynomial of degree n (or less) for which $f(x_j) = p_n(x_j)$ for each $j = 1, \ldots, n+1$. Note that Π_n not only depends on n but also on the set of interpolation points $\{x_j\}_{j=1}^{n+1}$.

(a) (5 points) Explain why Π_n is a projector. (**Hint:** What does $\Pi_n p_n$ equal if p_n is a polynomial of degree n?)

(b) (6 points) Consider the induced operator norm

$$\|\Pi_n\|_{L^\infty([a,b])} = \max_{0 \neq f \in \mathcal{C}([a,b])} \frac{\|\Pi_n f\|_{L^\infty([a,b])}}{\|f\|_{L^\infty([a,b])}} = \max_{\|f\|_{L^\infty([a,b])}=1} \|\Pi_n f\|_{L^\infty([a,b])}.$$

This norm is referred to as the Lebesgue constant and is often denoted Λ_n. Show that if $x_0 = a$ and $x_1 = b$, then $\|\Pi_0\|_{L^\infty([a,b])} = \|\Pi_1\|_{L^\infty([a,b])} = 1$.

(c) (8 points) If we write $p_n = \Pi_n f$ using the Lagrange basis polynomials $\{L_k\}_{k=0}^n$, i.e.,

$$\Pi_n f = \sum_{j=0}^n f(x_j) L_j(x),$$

show that

$$\|\Pi_n\|_{L^\infty([a,b])} = \max_{x \in [a,b]} \sum_{j=0}^n |L_j(x)|.$$

(d) (5 points) Let p_* be any polynomial of degree n. Show that

$$\|f - p_n\|_{L^\infty([a,b])} \leq (1 + \|\Pi_n\|_{L^\infty([a,b])}) \|f - p_*\|_{L^\infty([a,b])}.$$

(e) (6 points) Considering the polynomial p_* that minimizes $\|f - p\|_{L^\infty([a,b])}$ over all $p \in P_n$, explain the importance of the inequality in part (d) and in particular the role of the Lebesgue constant $\|\Pi_n\|_{L^\infty([a,b])}$.

Problem 5 (30 points) For a twice continuously differentiable function $f : [0,1] \to \mathbb{R}$, it can be shown (see problem 4) that the piecewise linear interpolant $p(x)$ on equally spaced intervals of size h satisfies the following L^2 error bound

$$\|f - p\|_{L^2([0,1])} \leq h^2 \|f''\|_{L^2([0,1])}. \tag{2}$$

The goal of this exercise is to check this bound experimentally using MATLAB/python.

(a) (8 points) Write a function which constructs a piecewise linear interpolant of a given function:

```
[c0, c1] = PiecewiseLinear(f)
```

The input `f` is a vector of the values of f at $0, h, 2h, \ldots, 1$ in order. The outputs `c0` and `c1` are vectors of length $1/h$ representing the coefficients of the linear pieces of the interpolant p. That is the ith piece of p (for $x \in [(i-1)h, ih]$ is given by

$$p(x) = \texttt{c1(i)}x + \texttt{c0(i)}.$$

Do not use any built-in MATLAB/python interpolation routines.

(b) (8 points) Write a function to evaluate piecewise linear polynomials:

```
y = EvalPiecewiseLinear(c0,c1,x)
```

Here `c0` and `c1` are the coefficient vectors returned by the function in part (a) and `x` is a vector of evaluation points (you may assume all of these values are in $[0,1]$). Your code should be *vectorized*, meaning that it should act on all of the values of `x` at once, rather than using a loop. You should not use any built-in MATLAB/python routines that can do piecewise polynomial evaluation.

(c) (4 points) Using your code from parts (a) and (b), and $f = \sin(3x)$, plot f along with its piecewise linear interpolant p for $n = 5, 10$ and 20 equally spaced points. Make sure your plot has a legend.

(d) (5 points) Now, study the convergence rate of piecewise polynomial interpolation. Let n range from at least 5 to 100 and calculate the L^2 error of the interpolant of f for n equally spaced points in $[0,1]$. Approximate the L^2 error

$$\|p - f\|_{L^2([0,1])} = \left(\int_0^1 |p(x) - f(x)|^2 \, \mathrm{d}x \right)^{1/2},$$

using a simple rule for numerical integration from Calculus (for example, Simpson's Rule), with enough points for an accurate estimate. Make a log-log plot of L^2 error versus n for the following functions (all on the same plot):

- $f(x) = \sin(3x)$
- $f(x) = \sin(30x)$
- $f(x) = e^x$
- $f(x) = x^{4/3}$
- $f(x) = |x - 0.567|$

- $f(x) = \sqrt[3]{x - \frac{1}{2}}$ (in Matlab calculate using `f = nthroot(x-0.5, 3)`).

Make sure you include a legend for your plot. Also include a reference line for the error bound. Since the bound is $O(h^2)$, you can use $y = h^2$. Do the data agree with the theoretical error bound? If not, how does the error depend on h? How do the convergence rates for $\sin(3x)$ and $\sin(30x)$ differ, and is this predicted by the error bound?

(e) (5 points) For the same functions as in part (d), approximate the L^∞ norm of the error $p(x) - f(x)$. Make a plot of L^∞ error versus n, for n ranging from at least 5 to 100. For each function, the error should be $O(h^p)$ for some value of p. Estimate p carefully for each function from your data and justify your choice of p.