CAAM 453: Numerical Analysis I

Problem Set 3
Due: October 18th, 2017

Note: Turn in all MATLAB *code that you have written and turn in all output generated by your* MATLAB *functions/scripts.* MATLAB *functions/scripts must be commented, output must be formatted nicely, and plots must be labeled.*

Problem 1: Piecewise Linear Error Bounds (30 points)

In this problem, you will prove an L^2 error estimate for piecewise linear polynomial interpolation. Let $f(x)$ be a C^2 (twice differentiable) function on $[0,1]$, and let the domain $[0,1]$ be split uniformly into intervals of length $h = 1/n$. Let p be the piecewise linear polynomial interpolating f on each interval. A key question is how the interval size h affects how well p approximates f. Here you will prove the following L^2 error estimate:

$$\sqrt{\int_0^1 |f(x) - p(x)|^2 \, dx} \leq h^2 \sqrt{\int_0^1 |f''(x)|^2 \, dx}. \qquad (*)$$

The integrals above are measuring the "size" of $f(x) - p(x)$ and $f''(x)$ respectively. So, what this estimate says is that (for fixed f) the L^2 error is $O(h^2)$.

Here are some relevant theorems that will help you prove equation $(*)$:

Taylor's Theorem (2nd Order): $f(x) = f(x_0) + (x - x_0)f'(x_0) + \displaystyle\int_{x_0}^{x} f(s)(x - s)\, ds.$

Mean Value Theorem: Given any $[a, b]$, $\dfrac{f(a) - f(b)}{a - b} = f'(z)$ for at least one $z \in [a, b]$.

Cauchy Inequality: For any functions u, v continuous on $[a, b]$,

$$\int_a^b u(x)v(x)\, dx \leq \sqrt{\int_a^b |u(x)|^2 \, dx} \sqrt{\int_a^b |v(x)|^2 \, dx}.$$

Fundamental Theorem of Calculus: For any C^1 function g, $g(x) = g(0) + \int_0^x g'(x)$.

Hint on proof:

1. *Start by looking only at the first interval $[0, h]$. On that interval, first prove $p(x) = f(0) + xf'(\alpha)$ for some $\alpha \in [0, h]$.*

2. *Estimate $|f(x) - p(x)|$ when $x \in [0, h]$. Use Taylor's theorem on f and then use the Cauchy inequality to show that*

$$|f(x) - p(x)| \leq C\sqrt{h}x \sqrt{\int_0^h |f''(s)|^2 \, ds}.$$

for some constant C.

3. Prove the estimate on $[0, h]$*:*

$$\sqrt{\int_0^h |f(x) - p(x)|^2 \, dx} \leq h^2 \sqrt{\int_0^h |f''(x)|^2 \, dx}.$$

4. Finally, prove the original estimate $(*)$ *on all of* $[0, 1]$.

Problem 2: Piecewise Polynomial Interpolation (30 points) In problem 1, you showed a quadratic error estimate for piecewise linear polynomial interpolation,

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

where $p(x)$ is a piecewise linear polynomial interpolation of $f \in C^2([0,1])$ on equally spaced intervals of size h. Here $\|g\|_{L^2([0,1])} = \sqrt{\int_0^1 |g(x)|^2 \, dx}$.

In this problem, we will check the theoretical error bound experimentally, using MATLAB. *In your writeup, include your code for (a–b) and your plots from (c–e).*

(a) Write a function which constructs a piecewise linear interpolation of a given function f:

```
[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`, `c1` are vectors of length $1/h$ representing the coefficients of the linear pieces of the interpolant p. Namely, the i^{th} piece of p (for x in the interval $[(i-1)h, ih]$) is given by

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

You should not use any built-in MATLAB interpolation routines, of course.

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

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

Here `c0` and `c1` are the coefficient vectors returned by your function from part (a). `x` is a vector of input points (you may assume all inputs are in $[0,1]$). The output `y` should be a vector of values of the piecewise linear polynomial described by `c0`, `c1`. Your code should be *vectorized*, meaning it should act on all the input values in `x` at once, rather than using a loop.

As in (b), you should not use any built-in MATLAB routines that can do piecewise polynomial evaluation.

(c) Using your code from (a–b), let $f = \sin 3x$, and plot f along with its piecewise linear interpolant p for $n = 5, 10, 20$ equally-spaced points.

(d) Now, study the convergence rate of piecewise polynomial interpolation. Let n range at least from 5 to 100 and calculate the L^2 error of the piecewise linear interpolant of f for n equally spaced points in $[0,1]$. Approximate the L^2 error $\|p - f\|_{L^2([0,1])} = \sqrt{\int_0^1 |p(x) - f(x)|^2 \, dx}$, using, e.g., 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 (include all functions on the same plot):

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

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

Make sure to include a legend! Also include on the plot a reference $O(h^2)$ line (the line $y = h^2$). 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) For the same functions as in (d), approximate the L^∞ norm of the error $p(x) - f(x)$. Make a plot of L^∞ error versus n, for n ranging at least from 5 to 100. For each function, the error should follow an $O(h^p)$ trend for some value of p. Estimate p carefully for each function from your data and justify your choice of p. Compare your choices of p to the error bound from hint 2 in problem 1.

Problem 3: Newton Interpolation (40 points, pledged)

Recommendation: read through all parts of this problem before starting on it. You may be able to save yourself some time!

In your writeup, include your code for (a–c) and your plots from (d–f).

(a) Write a MATLAB program which, given $n+1$ data points (x_j, f_j), computes the coefficients of the unique degree-n polynomial passing through those points, using *Newton interpolation*. The top-level function should take the form

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

Here `x` and `f` will be vectors of the same length $n+1$, containing $x_0, \ldots, x_n$ and $f_0, \ldots, f_n$ respectively. Your function will return `c`, a length-$(n+1)$ vector containing the polynomial's coefficients arranged from highest-order to lowest-order (i.e. the x^n coefficient first, and the x^0 coefficient last).

(b) 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 $p(x) = c_n x^n + \cdots + c_1 x + c_0$, Horner's scheme computes $p(x)$ as

$$p(x) = (\cdots(((c_n x) + c_{n-1})x + c_{n-2})x + \cdots c_1 x) + c_0,$$

requiring just n multiplications and n additions for each x.

Write a function to evaluate polynomials using Horner's scheme, given the coefficient `c` as output by your program 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, meaning that it works with all the points of `x` at once instead of looping over them individually.

(c) Write a function which updates interpolating polynomials. It should take the coefficients of an existing degree-n interpolating polynomial, the points $x_0, \ldots, x_n$, and a new point (x_{n+1}, f_{n+1}), and return the vector of coefficients of the degree-$(n+1)$ 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 $x_0, \ldots, x_n$, an error should be thrown.

(d) Test out your program from (a) in the following ways: For $n = 5$, 10, 20, let $x_0, \ldots, x_n$ be equally spaced points in the interval $[-2, 4]$. For each n, compute the polynomial interpolant for three different functions:

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

For each function, make a plot of the original function and the polynomial interpolant (using part (b) to evaluate the polynomial) for each n value (include legends!).

(e) Next, apply your program 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 $p(x)$ matches $f(x)$ in some areas within $[-3, 3]$ better than others (the

behavior in the bad parts is *Runge's phenomenon*). To improve the fit, choose appropriate additional x values in the range $[-3, 3]$ and add them to the polynomial interpolation using your code from part (c).

Make a plot of f, the original interpolant p, and your improved interpolant (including a legend).

(f) In this problem we will see how different choices of interpolation points affects the polynomial fit.

In (d–e), we just used equally spaced interpolation points x_j. Recall that another set of interpolation points are the *Chebyshev nodes*. For the interval $[-1, 1]$, the $(n+1)$-Chebyshev nodes are defined as follows:

$$x_j = \cos\left(\frac{i\pi}{2(n+1)}\right), \qquad i = 1, 3, 5, \ldots, 2n+1.$$

(The name is appropriate as these are the roots of the $(n+1)^{\text{st}}$ Chebyshev polynomial.) For a general interval $[a, b]$, the x_j are mapped linearly from $[-1, 1]$ to the interval $[a, b]$ of interest.

For $n = 1, 2, \ldots, 20$, compute the degree-n polynomial interpolants for each of the three functions in (d) on $[-2, 4]$, using both Chebyshev nodes and using equally spaced points. For each interpolant $p(x)$, approximate the L^∞ error $\max_{x\in[-2,4]} |f(x) - p(x)|$ by computing $|f(x) - p(x)|$ at 3001 equally spaced points in $[-2, 4]$.

Make a semi-log plot showing L^∞ error versus n for the Chebyshev nodes and equally spaced points.