# HOMEWORK 1
# MATH 563, SPRING 2020

DUE WEDNESDAY, JAN. 22

**Note:** You may end up using some of the interpolation code throughout the course, whenever you have need of it. This homework skews towards computation, to become accustomed to the messy process of implementation (the balance vs. theory may vary).

---

**Reading:** Read the following (book sections refer to Quarteroni unless otherwise noted).

- Section 8.2 and 8.2.1. Not all the divided diff. properties are essential, but they are examples to better understand the definition.

**Code to turn in:**

- `newt_polyval` from HW0, now with a working example that computes/evaluates the interpolant (using the given divided diff. code).
- The code from C1 and a working example applying this to C3 (e.g. interpolate $1/(1+x^2)$ with equally spaced points).
- You can submit the code for C3 (optional) if you want feedback.
- The code used to generate the plot in (c) (you can do the actual plot, annotations etc. manually; the point is to show the construction of the data).

## 1. Problems without code

**Q0 (do not submit; quick reminder on coding perils).** Identify the following:

- The number of elements in $x_i, x_{i+1}, \cdots x_j$
- The numbers in `range(n,0,-1)` in python and `n:-1:0` in `matlab`
- The `length` of `1:2:n` in `matlab`
- The result of `x.*y` where `x = [1; 2]` and `y = [2 3]` (`matlab`)
- The result of `x./y` and `x/y` where `x = [1 1]` and `y = [1 3]` (`matlab`)
- The result of `1/3` in (i) `matlab` and (ii) `python`
- The result of `1/0` in (i) `matlab` and (ii) `python`
- Let `x` $= [1, 2, 3]$. What is `y` after `y=x` and `x[1] = 1` in (i) `matlab` and (ii) `python`?

---

**Q1.** For data $f_0, f_0'$ and $f_1, f_1'$ at $x_0$ and $x_1$, respectively and $h = x_1 - x_0$, give a method for computing the CHIP at a point $x \in [x_0, x_1]$. Write this in the form you would use in writing code (done in a later problem) - you may want to break it up into short equations with temporary variables. You can use either divided differences or the basis from HW 0.

**Q2 (Inverse interpolation).** Let $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ be a given set of data from a function $y = f(x)$. Suppose we want to find a zero $x^*$ of $f(x)$ (i.e. such that $f(x^*) = 0$) and we know that exactly one exists.

a) Derive a formula for an interpolant of the **inverse** $x = f^{-1}(y)$ through the data.

b) Use (a) to estimate the value of the zero of $f(x) = \frac{1}{2}x - 3^{-x}$ in $[0, 2]$ (use the points $x = 0, 1$ and 2). Include a plot of the interpolant and the function. What does the Lagrange error formula say about the maximum error? Is it close to the actual error? *Note: you don't have to do the computation by hand.*

c) One could also interpolate $f(x)$ and estimate the root from there. What, in general, is the advantage over using inverse interpolation on $f^{-1}(y)$ vs. interpolation on $f(x)$?

**Remark:** this process is useful for getting a quick estimate of the point where a function crosses zero. An iterated version is used in a component of a popular root-finding algorithm (**Brent's method**).

---

**Q3 (Quadratic splines?)** (AC)**.** Recall the natural cubic splines defined in class.

a) Consider instead using a 'quadratic' spline. Write down the form of this spline and try to impose continuity conditions for $f$ and derivatives. How many (continuous) derivatives can it have? How many extra conditions need to be added?

b) Generalize to splines using polynomials of degree $k$.

## 2. Problems with code

**C1 (Barycentric interpolation).** The Lagrange form of the interpolant, while computationally unpleasant, can be manipulated into a nice form by a clever trick.[1]

Let $f(x)$, $\ell_i$ etc. be defined as per usual.

a) Let

$$\ell(x) = \prod_{i=0}^{n} (x - x_i).$$

Show, by factoring out $\ell(x)$ from the Lagrange form, that the interpolant can be written as

$$p(x) = \ell(x) \sum_{i=0}^{n} \frac{w_i}{x - x_i} f_i \tag{1}$$

where the $w_i$'s are numbers that can be 'computed in advance' (i.e. constants that depend only on the given points $\{x_\}$). Give a formula for $w_i$ as a product $\prod \cdots$.

[1]Reference: N. Higham, *The numerical stability of barycentric Lagrange interpolation*, IMA Journal of Numerical Analysis (2004), **24**, 547-556 or the Quarteroni, Chapter 8.

b) The formula can be improved: use (1) with $f(x) = 1$ to 'solve for' $\ell(x)$ in terms of the weights and show that

$$p(x) = \frac{\displaystyle\sum_{i=0}^{n} \frac{w_i}{x - x_i} f_i}{\displaystyle\sum_{i=0}^{n} \frac{w_i}{x - x_i}}$$

This is the **barycentric form** of the interpolant - which puts the interpolant in the form of a weighted average (with weights $f_i$).

c) [AC] It was noted by Trefethen, Higham and others that this form is actually a good choice for evaluating the interpolating polynomial if the points are chosen correctly. Why is there cause for concern in using this formula?

d) Write a routine `bary_weights(nodes)` that returns an array of weights `w` given points `nodes` $= x_0, \cdots x_n$. Then write a routine `bary_polyval(nodes,w,f,xvals)` that evaluates the Lagrange interpolant $p(x)$ in barycentric form at an array of points `xvals`.

*Caution: consider the case where $x$ is exactly one of the nodes!*

---

**C2 (Newton form).** Use the provided divided difference code plus your code from HW 0 (which you may need to adapt) to write a routine `newt_polyval`(nodes, coeffs, xvals) that evaluates the interpolant in Newton form,

$$p(x) = \sum_{i=0}^{n-1} c_i \prod_{j=0}^{i-1} (x - x_j)$$

given the coefficients `coeffs` $= (c_0, \cdots, c_{n-1})$. Check that it agrees with the alternate method in C2.

**Code given:** divided differences for the Newton form coefficients.

---

**C3 (numerical experiments).** There is more to say about high-degree interpolation that was hidden in the discussion from class.

a) To check the code: Use your barycentric (and/or Newton code) to test $f(x) = 1/(1 + ax^2)$ in $[-1, 1]$ with equally spaced nodes. Verify divergence when $a = 25$ but convergence when $a = 1$ and show a log-plot of the error (but consider also (b)).

b) [AC] There's an issue with the example in class - and possibly in (a) - that was omitted from the plots. For simplicity, instead consider the simpler function $f(x) = 1/(x + 2)$ in $[-1, 1]$.

Test what happens for the **Chebyshev nodes** $x_i = \cos(\pi(2i + 1)/2n)$ (for $i = 0, \cdots, n - 1$). as $n$ increases, using both methods of evaluating the interpolant. Show what happens (with a plot). Is this an example where the truncation error in the Lagrange interpolant blows up, or something else? Give a convincing argument for your answer.

**C4 (shapes via splines).** Suppose you are given data points for a parameterized curve $(x(t), y(t))$:

$$(x_i, y_i), \quad i = 0, \cdots, N-1, \qquad x_i = x(t_i),\ y_i = y(t_i)$$

with $t_i$'s in some range $[0, T]$.

a) Write a routine that takes in $N$ (the number of points) and this data and constructs a cubic spline approximation using CHIPs. The derivatives $x'(t)$ and $y'(t)$ should be estimated using the differences

$$\frac{x(t_{i+1}) - x(t_{i-1})}{t_{i+1} - t_{i-1}}$$

and forward/backward differences at the endpoints (code is supplied!).

b) Write a routine that takes the coefficients found in (a) and evaluates the interpolant at a set of points. Then use it on the data file provided (`points.txt`) to plot the interpolant.

**Code given:** A routine to read this data and the derivative estimation.

c) Show a plot of the result, including the interpolation points (as dots) and the spline evaluated on a finer set of points (so the curve is smooth).

**Remark:** Splines are commonly used to represent complicated shapes on a computer (e.g. in graphics or in simulations), as they are efficient to construct/evaluate.