MATH-131 (Numerical Methods for Scientists and Engineers) — Worksheet 6

Semester: Spring 2019, Instructor: Nicholas Knight

Due Mar. 5 at 2359. *Please remember to cite your sources, including collaborators.*

Deliverable: Submit a Live script titled `worksheet6.mlx` via CatCourses (under Assignments). Divide this file into sections, one for each of the following questions, plus an extra (final) section containing all the function definitions. Document each function definition to explain the input and output arguments. Also document key portions of the algorithm to make it clear you understand how your code works.

1. The Lagrange interpolation formula,

$$y = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j},$$

evaluates $y = p(x)$, where x is given and p is the (unique) degree-n polynomial that *interpolates* $n+1$ given coordinate pairs, $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. ("Interpolates" means intersects, i.e., $y_i = p(x_i)$ for $i = 0, 1, \ldots, n$.)

   (a) Implement this as a Matlab function with the signature

   ```
   function y = linterp(X, Y, x)
   ```

   (`X` and `Y` are arrays containing $x_0, \ldots, x_n$ and $y_0, \ldots, y_n$, resp.) You may not use any of Matlab's polynomial routines (like `polyfit` or `interp1`) nor the Symbolic Math Toolbox.

   Test your code by

   1. picking your favorite polynomial p of degree five,
   2. populating `X` with six distinct numbers,
   3. setting `Y` to be p evaluated at the corresponding entries of `X`, using `polyval`,
   4. picking `x` to be a number besides those in `X`,
   5. comparing p(`x`) with `linterp(X, Y, x)`.

   For example, implement $x \mapsto x^5 - 1$ by `p = @(x) polyval([1 0 0 0 0 -1], x);`

   (b) Based on the formula, for what inputs (`X`, `Y`, `x`) will Lagrange interpolation fail (not return a real number)? Modify your `linterp` documentation to mention this (input) requirement.

   (c) Extend your code to allow `x` to be an array, so that `y` is an array of the same dimensions, each entry being the Lagrange interpolation formula evaluated at the corresponding entry of `x`. Reusing from Part (a) the polynomial function `p` and the arrays `X` and `Y`, run the following commands

   ```
   x = linspace(min(X)-1, max(X)+1, 100);
   plot(x, p(x), 'ko:', x, linterp(X, Y, x), 'rx:');
   ```

   The red ×s should coincide with the black ○s.

2. The MATLAB function `poly` inputs an array of numbers and outputs a polynomial (as a coefficient vector) whose roots are the input numbers. In other words, `poly` inputs $x_1, \ldots, x_n$ and outputs $a_0, a_1, \ldots, a_n$ where

$$\sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = (x - x_1)(x - x_2) \cdots (x - x_n) = \prod_{i=1}^{n} (x - x_i).$$

Figure out an algorithm for this transformation, and write it in MATLAB as the function

```
function p = mypoly(X)
```

You may not call `poly` within `mypoly`, and you may not use the Symbolic Math Toolbox (`syms`, etc.).

Hint: Explicit expressions for a polynomial's coefficients in terms of its roots are given by *Vieta's formulas*. However, directly evaluating Vieta's formulas performs much more arithmetic than necessary — roughly 2^n operations — and is also somewhat complicated in Matlab. Try to discover a simpler algorithm that computes all coefficients at (roughly) the same time while performing (roughly) n^2 operations. The trick is that after processing the first k roots you have found $a_0, a_1, \ldots, a_k$ satisfying

$$(x - x_1)(x - x_2) \cdots (x - x_k) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0,$$

and the $(k+1)$-th step is multiplying

$$(a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + x) \cdot (x - x_{k+1});$$

derive formulas for the $(k+2)$ coefficients of this new polynomial in terms of $a_0, \ldots, a_k$.

Challenge; ungraded Implement this algorithm recursively, splitting the root-list in half and using polynomial multiplication to combine the two (sub-)results. Then accelerate the polynomial multiplication (a convolution operation) using the Fast Fourier transform, obtaining an algorithm that performs $O(n(\log n)^2)$ operations.