AMSC/CMSC 662 Homework 5 , Fall 2013
Due: 9:30am Tuesday, November 12.

Suppose we are given some data (t_i, y_i), $i = 0, \ldots, n$, and we want to construct a polynomial $p(t)$ of degree n so that

$$p(t_i) = y_i, \quad i = 0, \ldots, n.$$

This polynomial interpolation problem is considered in elementary numerical analysis courses, and one excellent way to construct $p(t)$ is using the **Lagrange polynomial**. We express our interpolating polynomial as

$$p_n(t) = y_0 L_0(t) + y_1 L_1(t) + \ldots + y_n L_n(t), \tag{1}$$

where

$$L_\ell(t) = \prod_{\substack{j=0 \\ j \neq \ell}}^{n} \frac{(t_j - t)}{(t_j - t_\ell)}.$$

For example, if $n = 3$, then

$$L_1(t) = \frac{(t_0 - t)(t_2 - t)(t_3 - t)}{(t_0 - t_1)(t_2 - t_1)(t_3 - t_1)}.$$

1a. (1 point) Show that each $L_\ell(t)$ is a polynomial of degree n.

1b. (1) Show that $L_\ell(t)$ has been (carefully) defined so that

$$\begin{aligned} L_\ell(t_\ell) &= 1, \\ L_\ell(t_j) &= 0, \text{ for } j = 0, \ldots, n \text{ but } j \neq \ell. \end{aligned}$$

1c. (1) Therefore, show that

$$p_n(t_j) = y_j, \quad j = 0, \ldots, n.$$

1d. (2) Give the exact operations counts for evaluating $p_n(t)$ from the expressions above, calculating each $L_\ell(t)$ directly from its definition, without any reordering of operations. (Give one count for floating point multiplications/divisions and a separate count for floating point additions/subtractions).

An interpolating polynomial often needs to be evaluated at a large set of points.

In the remaining problems, you will experiment with various ways to optimize the evaluation of the Lagrange polynomial.

2a. (3) Write a C function `Lagrange.c`, to evaluate the Lagrange polynomial p_n (specified by coefficients `y` and interpolation points `tj`) at the `k` points in the vector `t`. Your arguments should be `float p[], float y[], float tj[],`

`float t[], int n, int k`, where output `p` stores the values of the polynomial at the points `t`.

2b. (2) Look at the assembly code generated from using `gcc -O0 -m32` on `Lagrange.c`. (These options tell `gcc` to use no optimization and only the 32-bit instruction set.) Suppose we are using the FDEMW pipeline discussed in the B&H textbook. Given the latencies in Figure 5.12, what is the estimated CPE (cycles-per-element) for your function?

3. (5) In this problem I am asking you to experiment with different optimization techniques.

Write a well-documented C function `Lagrange2.c` that uses the Lagrange form of the interpolating polynomial and computes the same result but runs faster than `Lagrange.c`.

To do this, choose only one or two of the following optimization techniques:

- loop interchange
- loop unrolling
- multiple accumulators
- "reassociation"

Experiment a bit to find the best one or two techniques but only submit the best. Note that I am not including "code motion", because, for full credit, you have written `Lagrange.c` well enough that this will not help.

Time your two functions for values

$$n = 2, 5, 8, 12, 15, 20,$$

$$k = 10, 50, 100, 150, 200, 250, 500, 750, 1000.$$

4. (2) Use MATLAB to plot the improvement: (time for `Lagrange.c` - time for `Lagrange2.c`) / (time for `Lagrange.c`). Show the 6 curves (one for each value of n) on a single plot, with k on the horizontal axis. Label the axes (with `xlabel`, `ylabel`) and the curves (with `legend`) and give the plot a descriptive title (with `title`).

5. (3) Discuss your results, also mentioning less effective techniques that you tried, and the effects of using different optimization levels for your compiler. Make sure you specify the hardware and the compiler and compiler options that you used. Use the same compiler options for both programs.

Data: Since multiplication times can depend on the data, let's all work with the same numbers. Use this (ugly and inefficient) code to initialize `t`, `y`, and `tj`:

```
for (i=0;i<n+1;i++){
   tj[i] = (float)i/57.0;
   y[i] = 3.2*(float)(n-i)/(float)n;
}

for (i=0;i<k;i++){
   t[i] = (float)(i+1)*(tj[n-1]-tj[0])/(float)k;
}
```

A reality check: It is actually a bad idea to fit high-degree polynomials to data, because data have errors, and polynomials are very sensitive to small changes in the data. So you might think of these polynomials as the results of a least squares fit to a much larger set of data points. Even so, it is quite unusual to use polynomials of degree higher than $n = 6$, although they are sometimes used in numerical integration.

Submission instructions: Please keep the grader in a good mood by following these instructions. For submission, Tyler wants 3 email attachments:

- A PDF called hmwk5.pdf, with all of the written parts and the graph. You can make the PDF through LaTeX, export it from Word or export it from OpenOffice, etc. But note that Tyler will not accept .doc, .docx, and other formats.
- Attach your two files `Lagrange.c` and `Lagrange2.c`.

Tyler will time your functions on his own test problem. Especially efficient (but accurate) programs will receive extra credit.

Thanks for being considerate of Tyler's time by submitting in the format he wants.

Timing operations in C: Here is information on how to time operations in C:

```
/* Put this statement before main. */

#include <time.h>

/* Put these declaration statements at the beginning,
   when you declare other local variables. */

clock_t starttime, endtime;
double elapsed;

/* This statement substitutes for Matlab's tic. */

starttime = clock();

/* Do the work.  When working with any timer, beware of
   inaccuracies:

   --  If the work takes too long (not an issue in this
   assignment), the counter can roll over to zero (like
   the odometer on an old car) and timing will be inaccurate.

   --  If the work takes too little time, the "random" errors
   in the count will be a large percentage of the measurement.

   If necessary, do the work  S  times between the two calls
   to clock, where  S  is chosen to make the difference in
   starttime and endtime about CLOCKS_PER_SEC, and divide
   elapsed  by S.
*/

/* These two statements substitute for Matlab's toc. */
endtime = clock();
elapsed = ((double) (endtime - starttime)) / CLOCKS_PER_SEC;
```

Reference: http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_19.html#SEC310