Stat243: Problem Set 5, Due Wed. October 18

October 11, 2017

This covers Units 6 and 7.
It's due **on paper** and submitted via Github at the start of class
Some general guidelines on how to present your problem set solutions:

1. Please use your Rtex/Rnw/Rmd solution from PS1, problem 4 as your template for how to format your solutions (only non-Statistics students are allowed to use R Markdown).
2. As usual, your solution should mix textual description of your solution, code, and example output. And your code should be commented.
3. Your paper submission should be the printout of the PDF produced from your Rtex/Rnw/Rmd file. Your Github submission should include the Rtex/Rnw/Rmd file, any R or bash code files containing chunks that you read into your Rtex/Rnw/Rmd file, and the final PDF.
4. Use functions as much as possible, in particular for any repeated tasks. We will grade in part based on the modularity of your code and your use of functions.
5. Please note my comments in the syllabus about when to ask for help and about working together.
6. Please give the names of any other students that you worked with on the problem set.

Problems

1. Please read pages 1-7 (Sections 1.1-1.10) of Unit 9. With regard to Section 1.10 specifically, we'll discuss it in detail in class so you don't need to absorb everything from it, but please do read through it in advance.
2. In class I mentioned that integers as large as 2^{53} can be stored exactly in the double precision floating point representation. Demonstrate how the integers 1, 2, 3, ...,$2^{53} - 2$, $2^{53} - 1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format where d is represented as 52 bits. I'm not expecting anything particularly formal - just write out for a few numbers and show the pattern. Then show that 2^{53} and $2^{53} + 2$ can be represented exactly but $2^{53} + 1$ cannot, so the spacing of numbers of this magnitude is 2. Finally show that for numbers starting with 2^{54} that the spacing between integers that can be represented exactly is 4. (Note you don't need to write out e in base 2; you can use base 10).
3. GPUs tend to use single precision floating point calculations (4 bytes per real number). One advantage of this is that one is moving around half as much data during the process of computations so that should speed up computation, at the expense of precision. Let's explore a similar question related to integers.
   (a) Is it faster to copy a large vector of integers than a numeric vector of the same length in R? Do a bit of experimentation and see what you find.

(b) Is it faster to take a subset of size $k \approx n/2$ from an integer vector of size n than from a numeric vector of size n?

4. Let's consider parallelization of a simple linear algebra computation. In your answer, you can assume $m = n/p$ is an integer.

   (a) Consider trying to parallelize matrix multiplication, in particular the computation XY where both X and Y are $n \times n$. There are lots of ways we can break up the computations, but let's keep it simple and consider parallelizing over the columns of Y. Given the considerations we discussed in Unit 7, when you parallelize the matrix multiplication, why might it be better to break up Y into p blocks of $m = n/p$ columns rather than into n individual column-wise computations? Note: I'm not expecting a detailed answer here – a sentence or two is fine.

   (b) Let's consider two ways of parallelizing the computation and count (1) the amount of memory used at any single moment in time, when all p workers are doing their calculations, including memory use in storing the result and (2) the communication cost – count the total number of numbers that need to be passed to the workers as well as the numbers passed from the workers back to the master when returning the result. Which approach is better for minimizing memory use and which for minimizing communication?

      - Approach A: divide Y into p blocks of equal numbers of columns, where the first block has columns $1, \ldots, m$ where $m = \frac{n}{p}$ and so forth. Pass X and the jth submatrix of Y to the jth task.
      - Approach B: divide X into p blocks of rows, where the first block has rows $1, \ldots, m$ and Y into p blocks of columns as above. Pass pairs of a submatrix of X and submatrix of Y to the different tasks.

   Note: in reality parallelized matrix multiplication is done in much more complicated/sophisticated ways – if you're interested, there is some discussion in these notes from Jim Demmel's CS class: https://people.eecs.berkeley.edu/∼demmel/cs267_Spr12/Lectures/lecture11_densela_1_jwd12.ppt

5. Extra credit: Explain the mystery of why 0.2+0.3==0.5 and 0.01+0.49==0.5 are both TRUE in R but 0.3==0.2+0.1 is FALSE. (Caution: I don't know the answer to this myself and don't know how hard it will be to figure out.)