Stat243: Problem Set 5, Due Friday October 19

October 7, 2018

This covers Unit 6 plus bits and pieces of Units 8 and 9.
It's due **as PDF submitted to bCourses** and submitted via Github at 2 pm on Oct. 19.
Some general guidelines on how to present your problem set solutions:

1. Please use Rmd/Rtex as in previous problem sets.
2. Your solution should not just be code - you should have text describing how you approached the problem and what the various steps were.
3. Your PDF submission should be the PDF produced from your Rmd/Rtex. Your Github submission should include the Rtex/Rmd file, any R code files containing chunks that you read into your Rtex/Rmd file, and the final PDF, all named according to the guidelines in *howtos/submitting-electronically.txt*.
4. Your code should have comments indicating what each function or block of code does, and for any lines of code or code constructs that may be hard to understand, a comment indicating what that code does. You do not need to show exhaustive output but in general you should show short examples of what your code does to demonstrate its functionality.
5. Please note my comments in the syllabus about when to ask for help and about working together. In particular, **please give the names of any other students that you worked with on the problem set and indicate in comments any ideas or code you borrowed from another student.**

Problems

1. Linear algebra preparation. Before class on Wednesday October 17, please read the first section of Unit 9 (Numerical Linear Algebra) and answer the following question. (Turn in your answer as part of the problem set, but please do it by Wednesday.)

   For a symmetric matrix A, we have the following matrix decomposition (the eigendecomposition): $A = \Gamma\Lambda\Gamma^\top$, where Γ is an orthogonal matrix of (column) eigenvectors, and Λ is a diagonal matrix of eigenvalues. Use the properties discussed in Section 1 to briefly show that $|A|$ is the product of the eigenvalues.

2. Numerical problems in logistic regression. Consider that in logistic regression the linear predictor, $X_i\beta$, (where X_i is the ith row of the predictor matrix, X) is transformed into a probability using the inverse logistic (expit) transformation.

   $$p_i = \text{expit}(X_i\beta)$$

   where

   $$\text{expit}(z) = \frac{\exp(z)}{1+\exp(z)}.$$

This mathematical expression for the *expit* function doesn't work numerically on a computer for large values of z. Explain why not and re-express the function such that if implemented in computer code, it is numerically stable.

3. Consider the following estimate of the variance of a set of large numbers with small variance:

```
set.seed(1)
z <- rnorm(10, 0, 1)
x <- z + 1e12
formatC(var(z), 20, format = 'f')

## [1] "0.60931443706111987346"

formatC(var(x), 20, format = 'f')

## [1] "0.60931216345893013386"
```

Explain why these two estimates agree to only 5 digits when mathematically the variance of z and the variance of x are exactly the same.

4. (This problem makes use of ideas from Unit 8, to be covered at the beginning of the week of Oct. 15, so you may need to wait until then to work on this.) Let's consider parallelization of a simple linear algebra computation. In your answer, you can assume $m = n/p$ is an integer. Assume you have p cores with which to do the computation.

   (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 8, 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 cores 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 as the jth task out of p total tasks.
      - 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 workers, resulting in p^2 different tasks that we have to iterate through using our p cores.

   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