Stat243: Problem Set 7, Due Friday Nov. 14

November 6, 2014

Comments:

- This covers Units 11 and 12.
- It's due at the start of class on Nov. 14.
- As usual, simply providing the raw code is not enough; make sure to describe how you approached the problem, the steps you took, and output illustrating what your code produces.
- Please note my comments in the syllabus about when to ask for help and about working together.
- As discussed in the syllabus, please turn in (1) a copy on paper, as this makes it easier for us to handle AND (2) an electronic copy through Git following Jarrod's instructions.

Questions

1. Per the Piazza announcement, please read the Chen et al. paper in the section directory of the class repository and respond to 3 of the 7 questions we have posed. Note that this is/was due to Jarrod via Github Monday Nov. 10 at 2 p.m. following the instructions Jarrod posted on Piazza.
2. Some practice with matrix manipulations.
   (a) Show that the determinant of a square matrix that has an eigendecomposition is the product of the eigenvalues.
   (b) Show that $\|A\|_2$ is the largest eigenvalue of A for symmetric A. To do so, find the following quantity,
   $$\|A\|_2 = \sup_{z:\|z\|_2=1} \sqrt{(Az)^\top Az}.$$
   If you're not familiar with the notion of the supremum (the 'sup' here), just think of it as the maximum.
   Hints: when you get to having the quantity $\Gamma^\top z$ for orthogonal Γ, set $y = \Gamma^\top z$ and show that if $\|z\|_2 = 1$ then $\|y\|_2 = 1$. Finally, if you have the quantity $y^\top Dy$ for diagonal matrix D, express this as a sum and think intuitively about how to maximize it if $\|y\|_2 = 1$.
3. Suppose you have a dense matrix X and a diagonal matrix D. How would you compute the following in R in the most efficient way?
   (a) DX
   (b) XD

4. This problem has you work out the number of calculations involved in the LU decomposition. For any terms involving n^3 or n^2, please find the exact number of calculations, e.g., $5n^3/2 + 75n^2$, NOT the order of the calculation (i.e., not $O(n^3)$). You can ignore pivoting for the purpose of this problem. Remember not to count any steps that involve multiplying by 0 or 1. If you don't do the extra credit (part d), you can just use the result given in in 4d in the later subparts of the problem.

   (a) For an $n \times n$ invertible matrix, find the number of flops involved in the forward reduction step of the LU decomposition, which finds L and U (don't include the calculations that change b to b^*). [Remember from our derivation in class that $L = (L_{n-1} \cdots L_1)^{-1}$ can be formed directly from quantities computed in finding U, so all you're doing here is counting the number of flops in Gaussian elimination.] How does this compare to the number of calculations involved in the Cholesky decomposition based on the count we did in class? Now also consider the additional computation involved in finding b^*, but don't count (i.e., double-count) any calculations that you've already counted.

   (b) How many additional flops are involved in the backward elimination step that finds x based on $Ux = b^*$? (You do not need to re-derive this - we did it in class.)

   (c) How many flops are involved in (a) and (b) if b is actually a matrix (let's call it B) with p columns?

   (d) (Extra credit) We could use $solve()$ in R (i.e., the *dgesv* function in LAPACK) to explicitly find the inverse, A^{-1} and then multiply A^{-1} by the matrix B. This seems like it might be appealing - if p is large, maybe it's more efficient to find the inverse and then just use matrix multiplication to find the result all at once rather than doing a bunch of backward eliminations, one per column of B. Count the number of steps involved in using the LU to find $V = A^{-1}$ such that $LUV = I$. You should find that finding the inverse takes $n^3 + O(n^2)$ flops, including the flops needed to do the initial LU decomposition. For this part you do NOT need to find the constant in front of the n^2 term.
   Hints: $A^{-1} = U^{-1}L^{-1}$ and $L^{-1} = L_{n-1} \cdots L_2 L_1$ and the terms in the L_j matrices are known from doing the initial LU decomposition.

   (e) Now suppose we have $A^{-1} = V$ from part (d). Count the flops involved in calculating VB.

   (f) Finally compare the total flops involved in finding $A^{-1}B$ based on (c) and based on (d-e). Does the comparison of which is better depend on how big p is?

   (g) Empirically test your results in R for calculating $A^{-1}B$ using an arbitrary matrix $A = Z^\top Z$ with $n \in \{100, 3000\}$ and $p \in \{1, 100, 3000\}$. Compare the use of (1) the LU via *solve()* without explicitly finding the inverse, (2) *solve()* to explicitly find the inverse followed by matrix multiplication, and (3) using the Cholesky decomposition. Use only one thread in your timing.

   (h) Now suppose you are going to have to do a calculation with lots of new matrices, B_j, $j = 1, 2, \ldots$ in the future. Is there any advantage to precomputing A^{-1} and just doing the matrix multiplication, $A^{-1}B_j$ for many B_j as opposed to having L and U in hand and computing $(LU)^{-1}B_j$?

5. The following calculation arises in solving a least squares regression problem where the coefficients are subject to an equality constraint, in particular, we want to minimize $(Y - X\beta)^\top (Y - X\beta)$ subject to the m constraints $A\beta = b$ for an m by p matrix A. Solving this problem is a form of optimization called quadratic programming. Some derivation using the Lagrange multiplier approach gives the following solution:

$$\hat{\beta} = C^{-1}m + C^{-1}A^\top (AC^{-1}A^\top)^{-1}(-AC^{-1}m + b)$$

where $C = X^\top X$ and $m = X^\top Y$. Write an R function to efficiently compute $\hat{\beta}$, taking account of the principles discussed in class in terms of matrix inverses and factorizations. Note: you can use any of R's matrix manipulation functions that you want - I'm not expecting you to code up any algorithms from scratch. In reality an efficient solution is only important when the number of regression coefficients, p, is large.

6. (Extra credit) For this problem, your task is to empirically explore the condition number of the eigendecomposition. First create a set of eigenvectors. An easy way to do this is to find the eigenvectors of $A = Z^\top Z$ for arbitrary Z, throwing away the eigenvalues. Now explore creating positive eigenvalues of different magnitudes and creating $\Gamma\Lambda\Gamma^\top$ from the Γ you got for A and your chosen eigenvalues. Use *eigen()* and see how close the computed eigenvalues are to the actual eigenvalues. I would use $n = 100$ and let the eigenvalues vary between all being equal and having a range of values from very large to very small. At what condition number do you empirically see that your matrix is not numerically positive definite? How does the error in the estimated eigenvalues relative to the known true values vary with the condition number and the magnitude of the eigenvalues?