Problem Set 7

Due Mon. November 18, 10 am

Comments

- This covers material in Unit 10.
- It's due at 10 am (Pacific) on November 18, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting and attribution requirements.
- Note that is is fine to hand-write solutions to the the non-coding questions, but make sure your writing is neat and insert any hand-written parts **in order** into your final submission.

Problems

1. We've seen how to use Gaussian elimination (i.e., the LU decomposition) to solve $Ax = b$ and that we can do the solution in $n^3/3$ operations (plus lower-order terms). Suppose I want to know how inefficient it is to explicitly invert the matrix A and then multiply, thereby finding $x = A^{-1}b$ via matrix-vector multiplication. If we look into the C++ code behind `numpy.linalg` at the `inv` function, we see it solves the system $AZ = I$ to find $Z = A^{-1}$, using a Lapack routine DGESV, which uses the LU decomposition. Count the number of computations for

   a. transforming $AZ = I$ to $UZ = I^*$ (where I^* is no longer a diagonal matrix),
   b. for solving for Z given $UZ = I^*$, and
   c. for calculating $x = Zb$.

   Then compare the total cost to the $n^3/3$ cost of what we saw in class.

   Notes:

   - In counting the computations you should be able to make use of various results we derived in class concerning the Gaussian elimination computations and computations involved in a backsolve, so your answer should be able to simply combine together results we've already discussed without any detailed new derivation.
   - Given that numpy's call to `dgesv` doesn't take account of the special (diagonal) structure of I on the right-hand side, you do not need to take account of the fact that because I has zeroes and ones, one can actually save some computation. You can simply count the calculations as if I were filled with arbitrary values. (Note: if we did actually try to be careful about making use of the structure of I, it turns out we could save $n^3/3$ calculations.)

2. Suppose I need to compute the generalized least squares estimator, $\hat{\beta} = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} Y$, where X is $n \times p$ and Σ is a positive definite $n \times n$ matrix. Assume that $n > p$ and n could be of order several thousand and p of order in the hundreds.

   a. First write out in pseudo-code how you would do this in an efficient way - i.e., the particular linear algebra steps and the order of operations. Then write efficient Python code in the form of a function, `gls()`, to do this - you can rely on the various high-level functions for matrix decompositions and solving systems of equations, but you should not use any code that already exists for doing generalized least squares. Warning: the Cholesky functions in numpy and scipy don't return the result in the same form.

   b. Now compare your approach to directly using the inverse in Python (but make sure to only calculate the inverse once and to use an efficicient order of operations). Is the timing consistent with the results of Question 1 and the computational efficiency results from Unit 10? For simplicity, you can construct a positive definite matrix Σ as $\Sigma = W^\top W$, with the elements of the $n \times n$ matrix W generated randomly. (In a real problem, Σ would be set up based on the context of course.) You can also generate X and Y randomly

   c. Are the results for the solution, $\hat{\beta}$, the same numerically for the two approaches (up to machine precision)? Comment on how many digits in the elements of $\hat{\beta}$ agree, and relate this to the condition number of the calculation.

3. Two-stage least squares (2SLS) is a way of implementing a causal inference method called instrumental variables that is commonly used in economics. Consider the following set of regression equations:

$$\begin{aligned} \hat{X} = &\quad Z(Z^\top Z)^{-1} Z^\top X \\ \hat{\beta} = &\quad (\hat{X}^\top \hat{X})^{-1} \hat{X}^\top Y \end{aligned}$$

   which can be interpreted as regressing Y on X after filtering such that we only retain variation in X that is correlated with the instrumental variable Z. An economics graduate student asked me how he could compute $\hat{\beta}$ if Z is 60 million by 630, X is 60 million by 600, and Y is 60 million by 1, but both Z and X are sparse matrices.

   a. Describe briefly why I can't do this calculation in two steps as given in the equations, even if I use the techniques for OLS discussed in class for each stage.

   b. Figure out how to rewrite the equations such that you can actually calculate $\hat{\beta}$ on a computer without using a huge amount of memory. You can assume that any matrix multiplications involving sparse matrices can be done on the computer (e.g., using `scipy.sparse`). Describe the specific steps of how you would do this and/or write out in pseudo-code.

   Notes:

   - The product of two sparse matrices is not (in general) sparse and would not be sparse in this case.
   - As discussed in Section 6 of Unit 10, there are approaches in Python (and software packages more generally) for efficiently storing (to save memory) and efficiently doing matrix manipulations (to save computation time) with sparse matrices.

4. (Extra credit) In class we saw that the condition number when solving a system of equations, $Ax = b$, is the ratio of the absolute values of the largest and smallest magnitude eigenvalues of

A. Show that $\|A\|_2$ (i.e., the matrix norm induced by the usual L2 vector norm; see Section 1 of Unit 10) is the largest of the absolute values of the eigenvalues 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. It accounts for situations such as trying to find the maximum of the numbers in the open interval (0,1). The max is undefined in this case since there is always a number closer to 1 than any number you choose, but the *sup* in this case is 1.

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 $y^\top Dy$, think about how this can be rewritten given the form of D and think intuitively about how to maximize it if $\|y\|_2 = 1$.