

# Problem Set 7

Due Wednesday Nov. 15, 10 am

## Comments

- **No late submissions.** Given the timing of the second quiz, I will hand out solutions in class November 15.
- This covers material in Unit 10.
- It's due at 10 am (Pacific) on November 15, 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 it is fine to hand-write solutions to 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. Details of the Cholesky decomposition presented in Unit 10. Work out the operation count (total number of multiplications plus divisions) for the Cholesky decomposition, including the constant  $c$ , not just the order, for terms involving  $n^3$  or  $n^2$  (e.g.,  $5n^3/2 + 8n^2$ , not  $O(n^3)$ ). You can ignore the square root and any additions/subtractions. You can ignore pivoting for the purpose of this problem. Remember not to count any steps that involve multiplying by 0 or 1. Compare your result to that given in the notes.
2. Compare the speed of  $x = A^{-1}b$  using: (i) `np.linalg.inv(A) @ b`, (ii) `np.linalg.solve(A, b)`, and (iii) Cholesky decomposition followed by solving triangular systems. To ensure that  $A$  is positive definite (needed of course for the Cholesky), you can construct a matrix  $A$  as  $A = W^T W$ , with the elements of the  $n \times n$  matrix  $W$  generated randomly. Note that if your Python/numpy installation is not using a fast BLAS package, all three of these approaches will likely take a lot longer than if you are using a fast BLAS (e.g., on the SCF). See Section 6.1 of Unit 10 and/or Section 5 of Unit 6.
  - a. Using a single thread, how do the timing and relative ordering amongst methods compare to the order of computations we discussed in class and the notes using  $n = 5000$ ? Note that if one works out the complexity of the full inversion using the LU decomposition, it is  $4n^3/3$ .
  - b. Show how the timing scales with  $n$  for a few values up through  $n = 5000$  for all three of the approaches.

- c. Are the results for the solution  $\mathbf{x}$  the same numerically for methods (ii) and (iii) (up to machine precision)? Comment on how many digits in the elements of  $\mathbf{x}$  agree, and relate this to the condition number of the calculation. You can do this for one of the smaller values of  $n$  to reduce the time to compute the condition number.
3. 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  $(Y - X\beta)^\top(Y - X\beta)$  with respect to  $\beta$  subject to the  $m$  constraints  $A\beta = b$  for an  $m \times p$  matrix  $A$ . (Each row of  $A$  represents a constraint that a linear combination of  $\beta$  equals the corresponding element of  $b$ .) Solving this problem is a form of optimization called *quadratic programming*. Some derivation using the Lagrange multiplier approach (we'll see this in Unit 11) gives the following solution:

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

where  $C = X^\top X$  and  $d = X^\top Y$ .  $X$  is  $n \times p$ .

- a. Describe how you would implement this in pseudo-code, taking account of the principles discussed in class in terms of matrix inverses and factorizations
- b. Write a Python function to efficiently compute  $\hat{\beta}$ , using numpy or scipy's matrix manipulation/factorization functions. Note: in reality a very efficient solution is only important when the number of regression coefficients,  $p$ , is large.