Assignment 3

DSE 512

Assigned 2022-3-29 — Due 2022-4-9 11:59pm

Please read everything carefully and ask me if you have any questions. Your solution should be a document submitted to canvas that contains all of your code and any additional information requested in the problem.

Let matrix A be an $n \times n$ matrix where the ij'th element a_{ij} is given by

$$a_{ij} = \exp\left(-\frac{i+j}{ij}\right)$$

where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

Part 1: Background

1. Create two (base) R functions that take n as an input and return A as an output. Each should consist of a double `for` loop, iterating over the rows and columns and setting the values of A element by element. The first function should iterate over the rows first, the second over the columns first. Name the functions `rf()` for "rows first" and `cf()` for "columns first". (Hint: start with a matrix of 0's)
2. Create two analogous Python functions to those above using Numpy but *without* Numba.
3. Time all of your functions evaluated at $n = 10,000$. Summarize your results in a table or boxplot. Within each language, which implementation is fastest (rows first or columns first)? Is it the same across languages (rows/cols first), or do they differ?

Part 2: Performance Optimization

1. Choose your best Python method above and implement a new version that uses Numba. Evaluate it at $n = 10,000$ and compare to the other Python implementations. (Hint: make sure your Numba functions don't include the function compilation in the timer!)
2. In R, implement a vectorized version of its best method from Part 1. Say for example that iterating over columns first was fastest. In that case, you would loop over the columns, and then vectorize over the rows. Evaluate this at $n = 10,000$ and compare to the implementations in Part 1.

Part 3: Doing Only What's Necessary

1. For any value n, the matrix A is *symmetric*. So we need only fill one triangle and copy it over. In the language of your choosing, implement a function that employs this strategy. Start with the fastest function you have implemented in that language so far across Parts 1 and 2.
2. Suppose what I *really* wanted was to be able to compute Av for some n-length vector v, but for $n = 500,000$. For double precision, storing the matrix alone would use roughly 2TB of RAM, but storing the vector v uses only 4MB. Describe a strategy to compute Av in this case. You do not have to implement it in software, although you may if you prefer. (Hint: you don't need to go out of core!)