E0 226: Linear Algebra and Probability

Assignment 1

Submitted by: Prachatos Mitra *24.09.2018*

1 Problem 1

1.1 Problem 1 - Bonus Question

$$A = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1k} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nk} \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

$$Y = \begin{bmatrix} \sum_{i=1}^{k} c_{1i} \times x_i \\ \sum_{i=1}^{k} c_{2i} \times x_i \\ \vdots \\ \sum_{i=1}^{k} c_{ni} \times x_i \end{bmatrix}$$

The question given here is to find the maximum amount of potion that can be prepared given the set of constraints. Modelling this as a matrix equation, we have $AX = Y$, where A is the $n \times k$ matrix of percentages of each ingredient per person, X is the $k \times 1$ matrix which contains the quantity of each ingredient every person can use and Y is the $n \times 1$ matrix containing the total amount of potion that each person made. As shown above, every entry in Y is a sum of products of entries of A and X. Therefore, clearly we have to maximize Y, which means we need to maximize X as A is fixed.

The maximum quantity of the potion is thus bounded by the constraints specified in the problem, i.e. the vector M of maximum amount of each ingredient that can be used. Therefore, $A \times M$ gives the required maximum amount of potion that can be prepared.

2 Problem 2

2.1 Problem 2 - Part 2

The times taken by Numpy LinAlg versus the algorithm used here are listed below:

Data size	Time in Numpy (s)	Time in used algorithm (s)
500 x 500	0.039	48.471
100 x 100	0.005	0.263
50 x 50	0.002	0.035

Our algorithm converts A to A^{-1} by squaring A, augmenting with A, and converting the right half to I, which converts A^2 to A^{-1} in turn. Squaring itself is an expensive operation which can be optimized from $\mathcal{O}(n^3)$ using better algorithms, e.g. Strassen's algorithm, which reduces the time for that operation. Similarly, the Gauss Jordan elimination can be optimized by using the similar LU decomposition which, while asymptotically similar, generates easier to invert right and left factor matrices.

2.2 Problem 2 - Part 3

Suppose A is invertible, therefore A^T is also invertible. The row operations that take A^T to I take I to $(A^T)^{-1}$ or $(A^{-1})^T$. Therefore, if we know how to convert A^T to I, we can convert A to I, because the transpose of $(A^T)^{-1}$ is A^{-1}. Row operations for A^T are equivalent to column operations in A. The row operations that convert $[A^T|I]$ to $[I|(A^T)^{-1}]$ thus correspond to column operations in $[A|I]$ that convert it to $[A^{-1}|I]$.

2.3 Problem 2 - Bonus Question

2.3.1 Using only two row operations

This is possible if the operation that is removed is swapping two rows. Swapping two rows can be implemented as follows:

$$\begin{aligned}
A[i] &= A[i] + 1 \times B[i] \\
B[i] &= -1 \times B[i] \\
B[i] &= B[i] + 1 \times A[i] \\
A[i] &= A[i] - 1 \times B[i]
\end{aligned}$$

This can be used to iterate over the length of a row to effectively swap two rows.

2.3.2 Time complexity

The algorithm has two basic parts. Of these, squaring the matrix naively takes $\mathcal{O}(n^3)$ time and row-reduction using Gauss Jordan Elimination takes $\mathcal{O}(n^3)$ mathematical operations. For each row (and column), we need a pivot, so in total, there are n pivots. Each of these require some cn^2 operations each, which add up to cn^3 on average). Since both parts require $\mathcal{O}(n^3)$ time, the total time taken is $\mathcal{O}(n^3)$ as well.