CS 530 - Fall 2018
Homework 2 - brief answers

Due: Thursday, October 4

Reading : Read the first 2 sections, (6 pages, up to page 162) of the posting on the course homepage on probabilistic algorithms.
The url is: http://cs-www.bu.edu/faculty/homer/530f18/rand-alg1.pdf

Problems:

1.

$$\text{Let } M = \begin{matrix} 2 & 2 & 1 \\ 6 & 6 & 5 \\ 0 & 1 & 4 \end{matrix}$$

i. Compute the determinant of M.

Answer: The determinant is -4.

ii. Use the LU decomposition algorithm to try to determine the LU decomposition of M. If the algorithm does give the decomposition, write the L and U you found. If the algorithm does not work here, show where you get stuck.

Answer: It does not have an LU decompositions as you get stuck when you compute the first Schur complement (the 2 by 2 one) with a 0 in the first row, first column.

iii. Does the matrix M have an LUP decomposition ? Briefly explain why or why not.

Answer: Yes, M is non-singular and all nonsingular matrices have LUP decompositions.

2. In this problem we want to show that computing the determinant of a matrix reduces to computing the LUP decomposition of that matrix.

(i). So you are give a matrix A and its decomposition into the matrices L,U and P with PA=LU as we've seen in class. (A may be or may not be non-singular.)
Use the decomposition to efficiently compute the determinant of matrix A. By efficiently I mean your method should be faster than using the definition of the determinant directly.

Answer: We have A = P^{-1}LU. So det(A) = det(P^{-1}LU) = det(P^{-1})(det(L))(det(U)). As P^{-1} is a permutation matrix we can calculate its det as either 1 pr -1. As L and U are triangular we can calculate their determinants as they are the products of their diagonal elements. Multiplying these elements together gives us the determinant of A.

If A is singular the above decomposition of A may not exist as the algorithm to decompose A will reveal by failing. In this case, det(A)=0.

(ii). Now assume that T(n) = the number of steps the LUP algorithm takes to find the LUP decomposition of any non-singular n x n matrix A. Estimate the number of steps D(n) your algorithm uses when it starts with matrix A and it then computes the LUP decomposition, and finally uses the decomposition to to compute the determinant for matrix A in terms of n and the function T(n).

You should make some assumption about the function T(n) and its growth rate.
This was similarly done in Theorem 28.1 of our text for the function I(n). Take a look at Thm. 28.1 and use similar assumptions about T(n). For example, it is assumed in 28.1 that I(3n) = O(I(n)). State what assumptions you use to get the complexity of I(n) in terms of T(n) in this problem.

Answer: D(n) = T(n)+the time to compute det(P^{-1})+the time to compute the product of the diagonal elements of L and of U

Now det(P^{-1}) can be computed by computing the parity of the number of row interchanges needed to go from the identity matrix to P^{-1}. This is at most n interchanges and can be found in O(n) steps. And the determinants of L and U can each be computed in n-1 mults.

So the total no of steps is = T(n) + O(n) steps. Hence as long as we assume T(n) $\geq$ O(n) this is O(T(n)) steps.

3. Solving systems of dependent equations.
Below is an A and b which give an example of 3 equations with 3 unknowns of the usual form Ax = b, but where the rank of your 3x3 matrix A is 2 (so A is singular).

```
     2  3  -2            5
A =  4  4  -2   and b =  3
    -2 -1   0            1
```

(i). Try to apply the LUP decomposition algorithm to A. Do you get an LU or an LUP decomposition for A ?
Answer:

You get an LU decomposition using the LU decomposition algorithm, so don't need P (or just let P = the identity matrix).

The algorithms results in A = LU where,

For L you get

```
 1 0 0
 2 1 0
-1-1 1
```

For U you get

```
2 3 -2
0-2  2
0 0  0
```

(ii). Do you think this same outcome happens every time you start with an A which is singular as above ? (Note that b does not play any role in your answer here.)

Answer:

No. For some singular A's you will get stuck in the LU algorithm because you will reach a step which tries to divide by 0. For these A the LU algorithm does not work, and you need to try LUP (where you still may get stuck as A is singular and may have a whole column of 0's) in which case you may need to try an ad hoc method not based on the algorithms we covered.

Specifically, state how many different solutions to these 3 equations there are when you fix b to be $(531)^T$ as above ? Explain your reasoning.

Answer: Since the first equation is the difference of the sum the second and third equation, the equations are not linearly independent. (Hence there is not a unique answer for each vector b.)

In this case we have an LU decomposition of A and we have b = (5 3 1). Given this you should try the usual backward and forward substitutions methods discussed in the lab sections in order to solve Ax=b by solving LUx=b.

When you do this you will get a contradiction, something like 0=1.
This means there are no solutions to the 3 equations when you have b = (5 3 1).

Would a different choice for b result in a different number of solutions to Ax=b?

Answer: Yes. For certain values of b there will be infinitely many solutions.

For example, if you have 3 equations in three unknowns given by A, but A is singular and the equations dependent on each other. In this case you would be able to reduce the three equations to 2 equations with 3 unknowns.

Each linear equation in 3 variables specifies a 2-dimensional plane. If the 2 planes specified by the 2 equations intersect in a line (this depends on the b you have). then there are infinitely many solutions.

4. Producing random permutations of V= $\{1, 2, ..., n\}$.
It should be possible for a a randomized algorithm R to produce every permutation of V, and have the property that each of the the n! permutations of V should be produced with the same 1/n! probability. So when you run R once, you get a random permutation of V.

Consider the following algorithm R:

Begin with a fixed permutation σ of the set V= $\{1, 2, ..., n\}$. (If you like just let σ be the identity function.)
You then "randomize" σ by:

1. Independently at random pick n integers $a_1, a_2, ..., a_n$ from the set V. (The integers you pick may be repeated in the list of a_i's.)
2. For each element i of V from 1 to n, swap $\sigma(i)$ with $\sigma(a_i)$ in V.
3. Output the σ that results.

(i). Is every permutation of {1,2,... n} generated by some run of this algorithm R? Why or why not?

Answer: Yes. To generate a permutation τ = (τ(1), τ(2), ..., τ(n)) we let $a_1, a_2, ..., a_n$ be the sequence of integers which at each step i moves the number $\tau(i)$ into the ith position, and so in the end τ is generated.

(ii). Explain why the permutation that is output in step 3 is not a random permutation of V.

Answer: (i). The number of permutations generated by all the different choices of the algorithm is n^n.

Why ? Every sequence of n integers $a_1, a_2, ..., a_n$ generates a permutation which is an output of the algorithm.

And given a specific permutation p of V, say p = $p_1, p_2, ..., p_n$ one can specify a list $a_1, a_2, ..., a_n$ which, when chosen by R is step 1, will cause R to output p.

There are n^n such sequences so n^n such outcomes. Each one of these outcomes has the same probability of happening when the algorithm is run.

(ii). The number of different permutations generated by the algorithm is n!

Each of n element sequences $a_1, a_2, ..., a_n$. can occur when the algorithm is run, making n^n outcomes of algorithm R. Each one of these outcomes is a permutation of V so there are more outcomes of R than n!, which is the number of different permutations of V.

So if each permutation has the same probablility of occurring as the outcome, it must be the case that n^n outcomes of R must be spread evenly over the n! permutation. Each permutation must account for n^n/n! outputs of R. And specifically it must be that n! divides n^n evenly (with no remainder). But, this statement is false for any $n \geq 3$.

Seeing this is almost immediate: Clearly n-1 divides n! evenly, but it does not divide evenly into n^n since the only factors of n^n are factors of n. In particular n-1 does not divide evenly into n^n. (Note: This is obvious if n-1 is even but needs a more detailed argument when n-1 is odd.)