**Junda Lou**

**5.1.2017**

**CS301.01**

**Project 1**

Report:

I wrote a program to save the random generated numbers (from 0-1000) to a test.txt as the same form as on the BlackBoard. For the calculations that n < 20, it only takes less than a second to give the results. As the number n increases, it takes longer and longer to show the pivoting every time and finally gives the result, 2 seconds for n = 50, and 15 seconds for n = 100. The speed is based on the CPU as well, but it depends more on the algorithm that I use in my main program. Each time I do the elimination with the pivot row that is selected, I have to iterate the whole matrix except the pivot row, so it is a O(n2 – n – 1) operation. Each time I get the ratio from each row, I have to iterate every row except those are already been selected as pivot rows, so it is initially a O(n2) operation, but it will decrease by n every time.

In addition to the time complexity, there is another big problem, which is the accuracy of the program, in another word, loss of significance. I found this problem as I started testing the example given on the BlackBoard. In C++, there is no fraction calculation, while the double data type will have a default number of significant digits for every variable, which means, it cannot represent numbers like 1/3 or 8/3 accurately. What is more, since 1/3 is less than 1, its significant digits starts after the decimal, which get itself 1 more digit after the decimal place compare to 8/3. When I try to add these 2 numbers together, it does not provide me 3 exactly. As the number of equations increase, the error will gets bigger and reflect in the results.

Thirdly, because I am testing the program using random numbers within a range of 0 – 1000, I found that I have a good chance getting no answers in my results. For a system of linear equations, n variables should have at least n equations to get unique answers for every variable. However, when eliminating equations and finally put them into row-echelon forms, some rows might have more than 1 coefficient left, which means more than 1 equation has the same coefficient for the same variables. For example, 2x+2y = 2 and x + y = 1, if you eliminate the first equation with the second, you will get 0 + 0 = 0 and x + y = 1, which cannot be solved. Because I limit the range of random generated numbers, I will have a better chance to fall into such situation as the number of equations grows.

I also tried to come up with a very basic Fraction struct in C++ to keep the fraction number all the way to the final form of the matrix, and do the double data type division at last to ensure accuracy, but I cannot optimize it on time, so it is very awkward doing the addition, which times the denominators of two fractions every time, regardless if they have shared factors. This is fatal for the program when dealing with a huge number of equations because the denominator will grow huge.