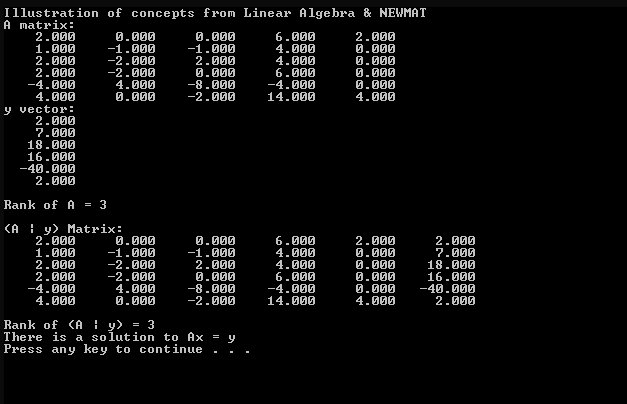
**IE531: Algorithms for Data Analytics Spring, 2018**

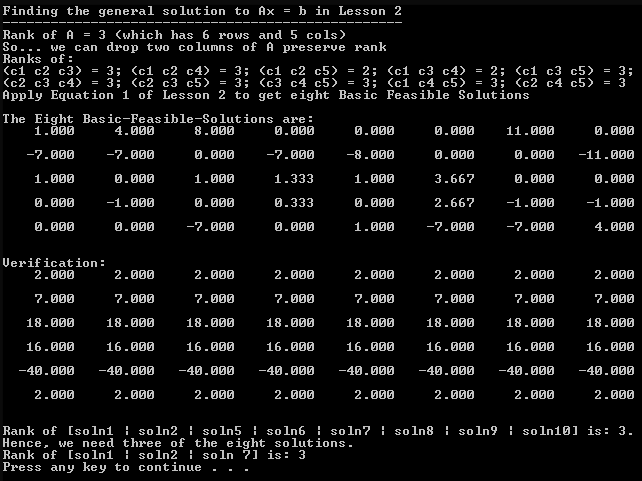
**Karthik Venkata(kvn3)**

**Homework 1: Review of Linear Algebra, Probability & Statistics and Computing**

1. **(a).** Upon modifying and running the Linear Algebra 1.cpp program, the output obtained was the follows:



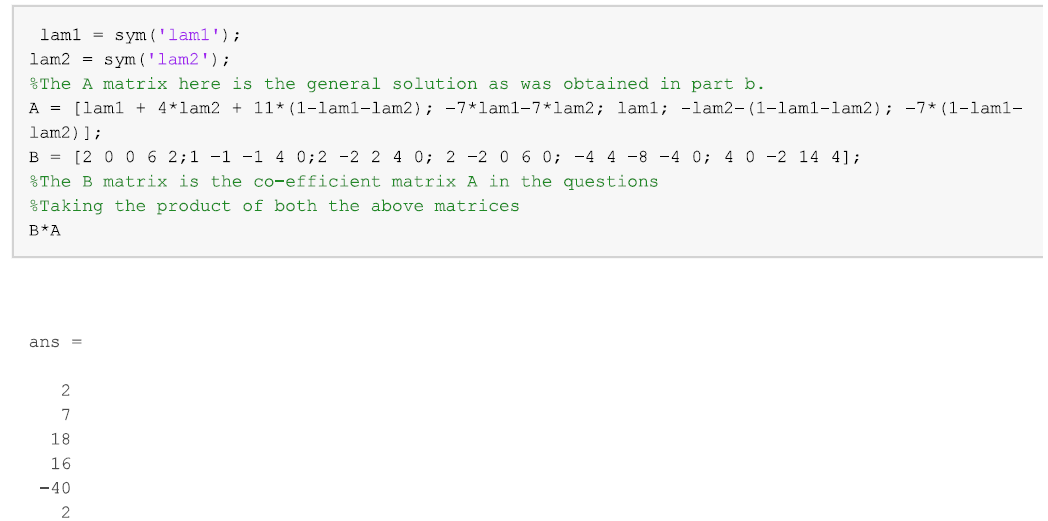
1. **(b)** The output generated upon running the modified version of the Linear\_Algebra\_2.cpp for computing the solutions of the given set of equations is-



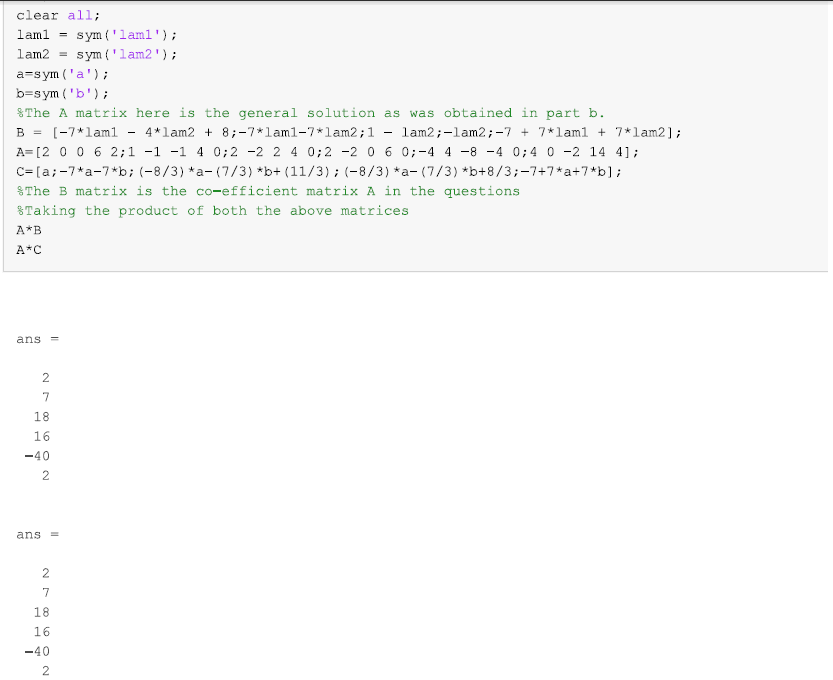
Therefore, we find that the rank of the set of column (solution) vectors 1, 2 and 7 is 3 and the rank of the entire set of solutions (column vectors) is also 3. Hence, the basis set which gives the complete solution set consists of columns 1, 2 and 7. The general solution to these set of equations (an affine combination of all the solutions as obtained above) is as follows-

The general solution to these set of equations is of the form-

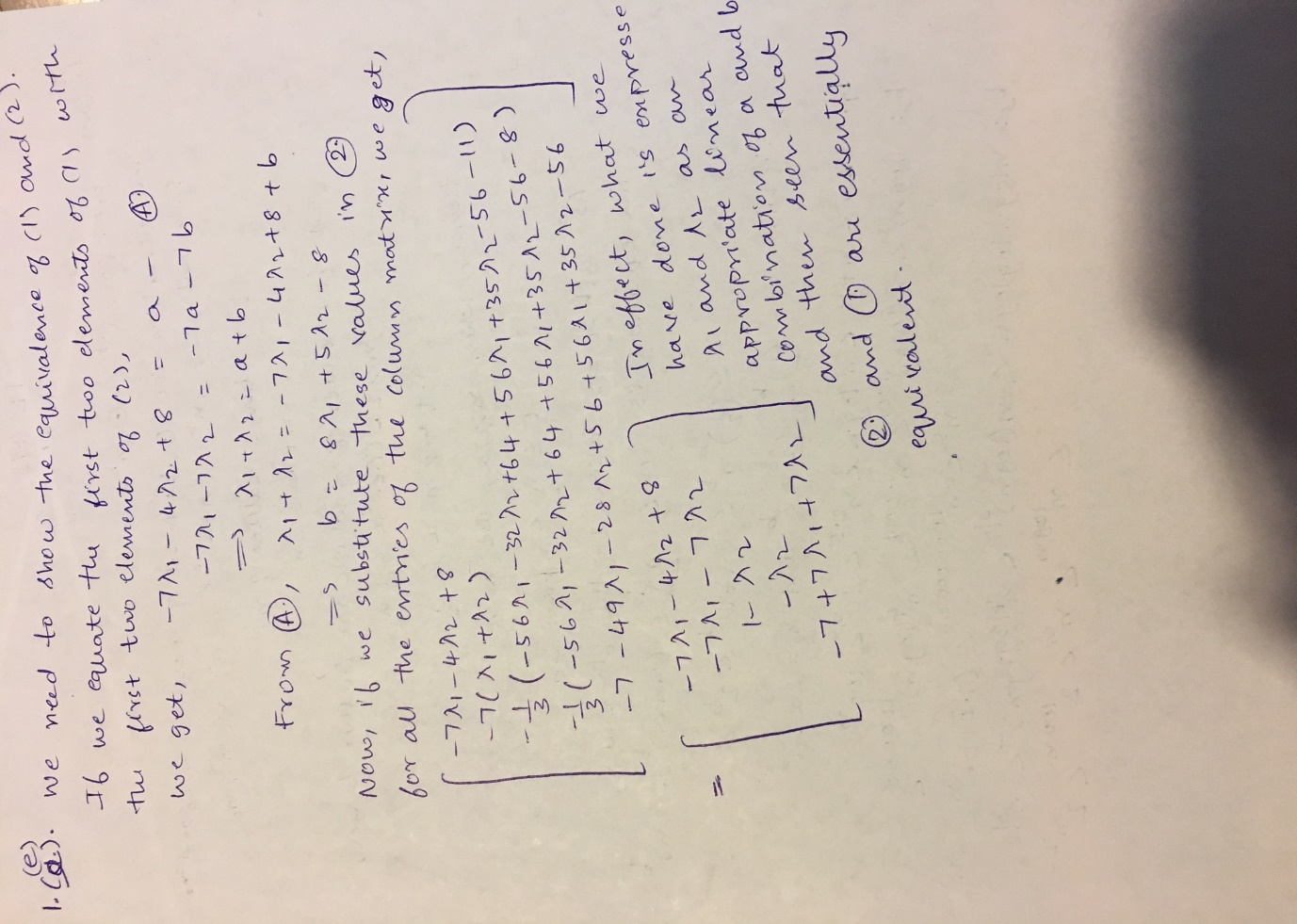
1.(c). The general formula obtained in part (b) was plugged into the equation **Ax = y** and was verified using MATLAB. The MATLAB snapshot of the same is as below:



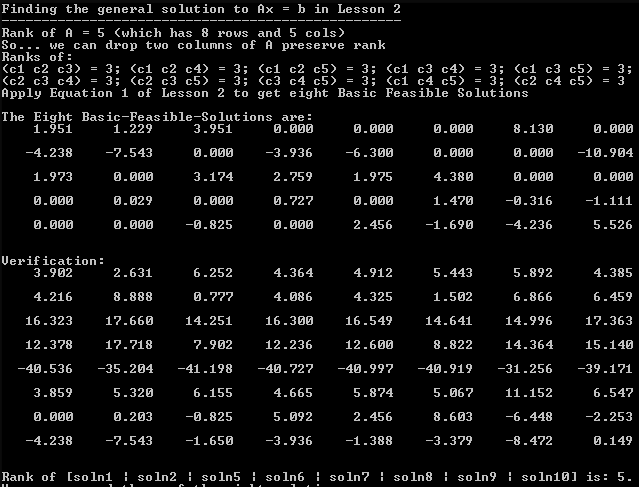
1.(d). Both the given set of solutions were verified by plugging into the general equation **AX = y** andthe following MATLAB snapshot shows the same:



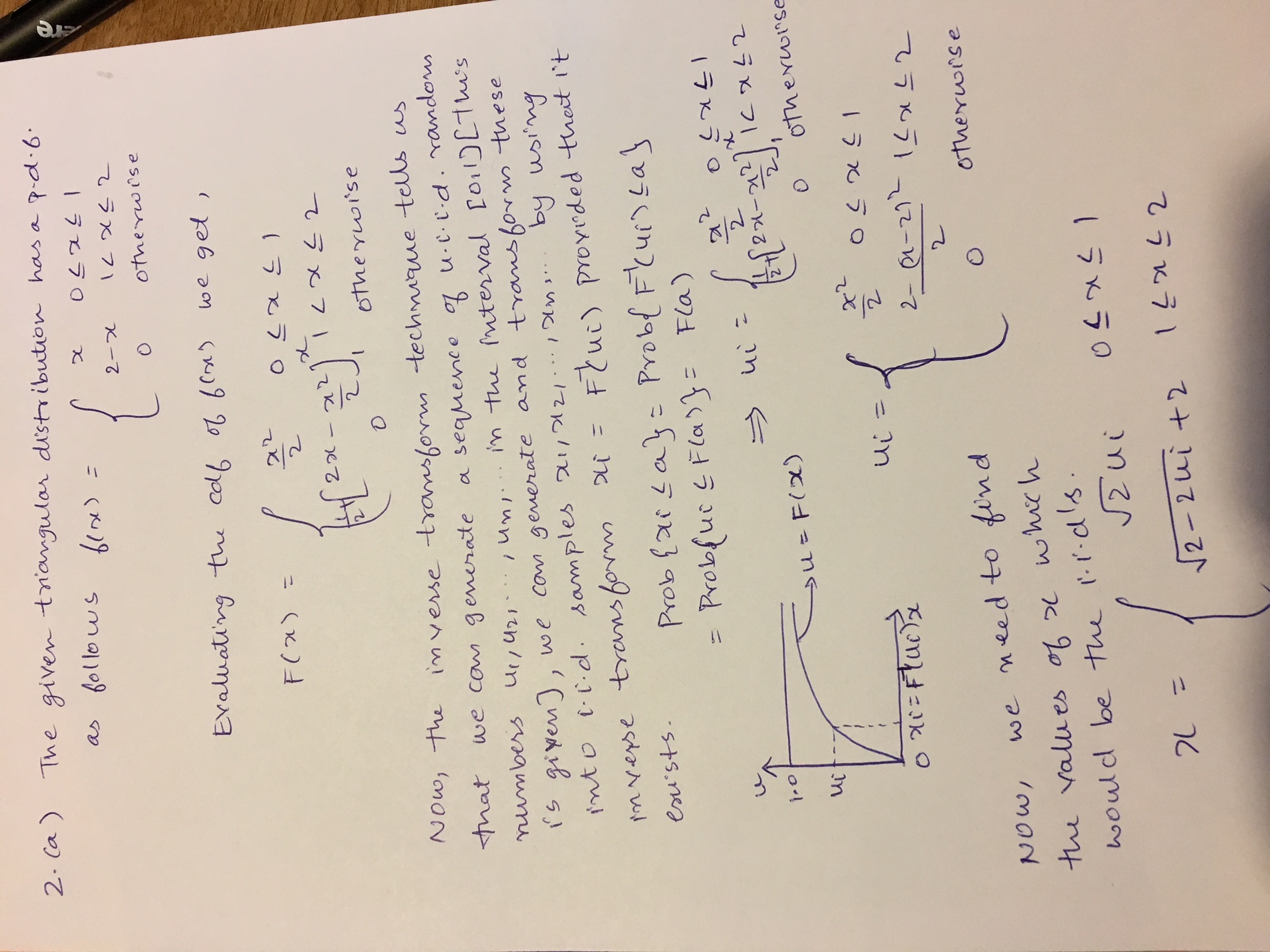
**Correction**-‘A’ matrix is the co-efficient matrix that is given in the question. ‘B’ and ‘C’ are the equivalent set of solution matrices that have been given.

(e). 

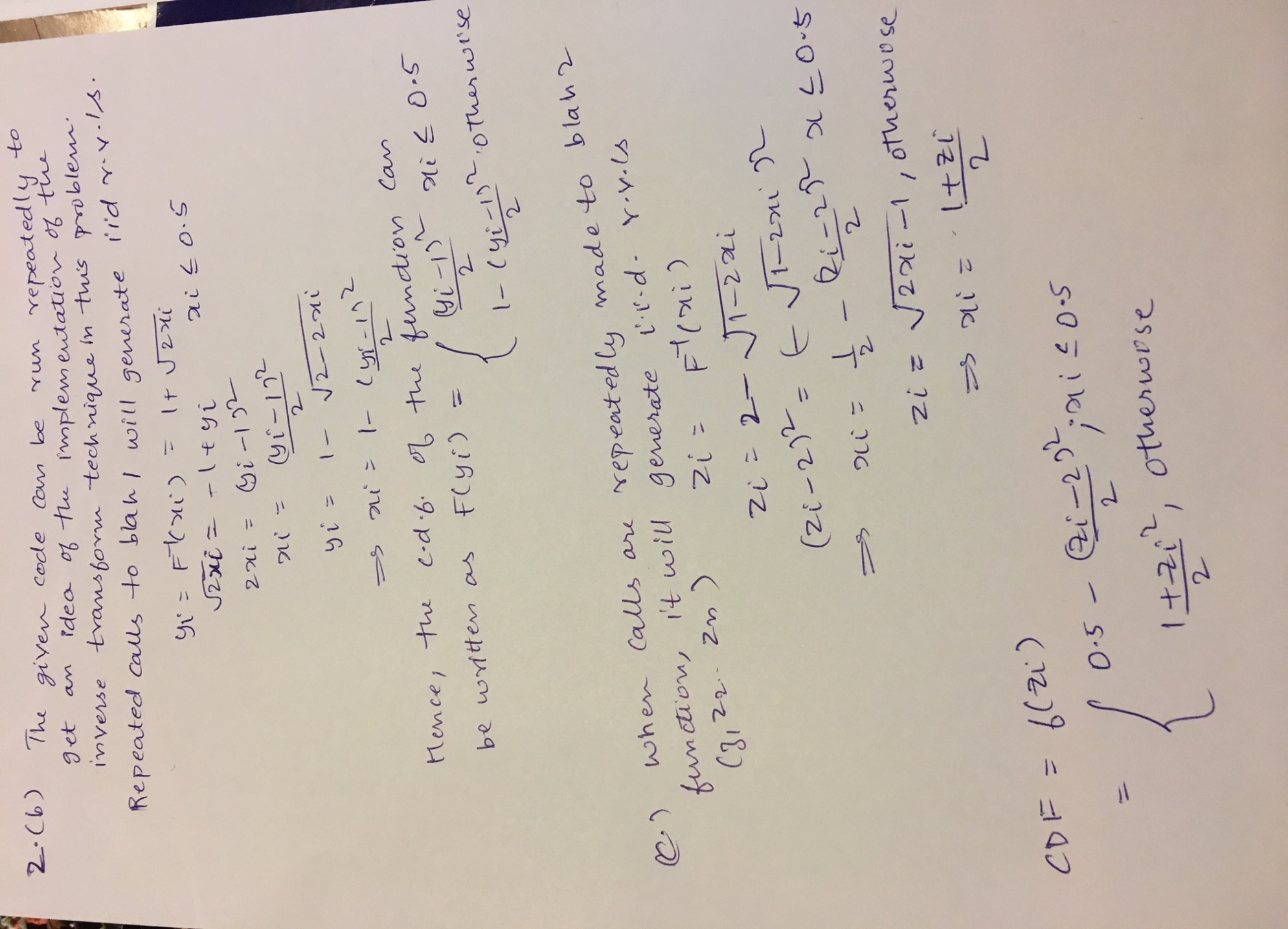
(f). We see that as the rank of the A matrix is five which equals the number of columns. Hence, the given set of equations must have a unique solution as the coefficient matrix A is full rank. The output obtained from visual studio is as follows-



1. **(a)** The cdf of the given functions can be found out by integrating the probability distributions in the given range doing which we obtain-

****

**2.(b). and 2(c).**

****

1. **(a)** This is a typical example of recursion in which the outputs would be as follows-

3a(I). kablooey is the output. As one enters the braces of the f() function, the if loop is skip until the very end when the ‘\n’ character is actually reached(when the user actually terminates the entry of the characters by entering ‘\n’). While this happens, the cout keeps printing all the letters sequentially in the order they were entered and the f() is called after that and this process continues. Once ‘\n’ is encountered, we exit the function loop. Here, it is getting a character then prints it and makes a function call.

3a(II). yeoolbak is the output. In this case, the cout does not get executed till the time ‘\n’ is encountered and in contrast with the part I, this time the printing out of the values starts with the last letter and keeps going backward. Here, before the character is printed the function is called once more. Hence, it gets stored in a recursive fashion before everything gets printed out. This results in the characters being printed in the reverse order when compared to the earlier case.

**3.(b).** The idea of this algorithm is: if m>n, GCD (m,n) is the same as GCD(m-n,n). If m/d both leave no remainder, then (mn)/d leaves no remainder.

An example using the substitution model:

gcd(468, 24)

gcd(444, 24)

gcd(420, 24)

...

gcd(36, 24)

gcd(12, 24) (Now n is bigger)

gcd(12, 12) (Same)

=> 12

This recursive algorithm essentially takes a pair of numbers and then making a comparison between the two numbers, subtracts the smaller one from the larger one and keeps on doing this until both of them are equal or one of them reduces to 1 which eventually gives us the greatest common divisor.

Generic Proof-The validity of the Euclidean algorithm can be proven by a two-step argument. In the first step, the final nonzero remainder *rN*−1 is shown to divide both *a* and *b*. Since it is a common divisor, it must be less than or equal to the greatest common divisor *g*. In the second step, it is shown that any common divisor of *a* and *b*, including *g*, must divide *rN*−1; therefore, *g* must be less than or equal to *rN*−1. These two conclusions are inconsistent unless *rN*−1 = *g*.

To demonstrate that *rN*−1 divides both *a* and *b* (the first step), *rN*−1 divides its predecessor *rN*−2

*rN*−2 = *qN* *rN*−1

since the final remainder *rN* is zero. *rN*−1 also divides its next predecessor *rN*−3

*rN*−3 = *qN*−1 *rN*−2 + *rN*−1

because it divides both terms on the right-hand side of the equation. Iterating the same argument, *rN*−1 divides all the preceding remainders, including *a* and *b*. None of the preceding remainders *rN*−2, *rN*−3, etc. divide *a* and *b*, since they leave a remainder. Since *rN*−1 is a common divisor of *a* and *b*, *rN*−1 ≤ *g*.

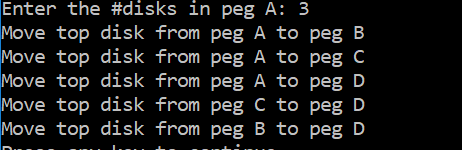
In the second step, any natural number *c* that divides both *a* and *b* (in other words, any common divisor of *a* and *b*) divides the remainders *rk*. By definition, *a* and *b* can be written as multiples of *c*: *a* = *mc* and *b* = *nc*, where *m* and *n* are natural numbers. Therefore, *c* divides the initial remainder *r*0, since *r*0 = *a* − *q*0*b* = *mc* − *q*0*nc* = (*m* − *q*0*n*)*c*. An analogous argument shows that *c* also divides the subsequent remainders *r*1, *r*2, etc. Therefore, the greatest common divisor *g* must divide *rN*−1, which implies that *g* ≤ *rN*−1. Since the first part of the argument showed the reverse (*rN*−1 ≤ *g*), it follows that *g* = *rN*−1. Thus, *g* is the greatest common divisor of all the succeeding pairs:

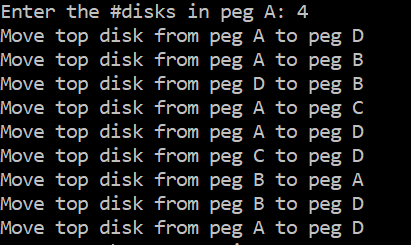
*g* = gcd(*a*, *b*) = gcd(*b*, *r*0) = gcd(*r*0, *r*1) = … = gcd(*rN*−2, *rN*−1) = *rN*−1.

*(Reference-Wikipedia Euclid’s algorithm)*

**3.(c).** EDCB The putchar() function prints the string values and in the first iteration, num%10 returns 4 which when used with putchar(4-corresponds to the 4th alphabet after A gives us E). num/10 gives us 123 which is then given as an input to putchar again and used recursively. In the second iteration, 123 is started of as an input, num%10 gives us 3 which is then input into the putchar which returns D(the 3rd letter after A) and the recursion follows through. In the next iteration, num/10 gives us 12 which when used with the function and running num%10 gives us 2 which when used with putchar gives us C(Going 2 places before A). In the final iteration, similarly we obtain B following the same line of logic and adding 1 to the final putchar call we get B, hence the output EDCB.

**3.(d).** There are illegal moves from n=5` to n=10.





Illegal move as the move from A to B is repeated without the disk in B getting relocated after the first move. The other moves have all been circled in red for all the other cases.

