IE531: Algorithms for Data Analytics
Spring, 2018

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

Due Date: February 16, 2018

©Prof. R.S. Sreenivas

Instructions

1. You can modify any of the C++ code on Compass to solve these problems, if you want. It might help you with honing your programming skills. If these attempts (at using C++ code) is turning out to to be intense, you can use MATLAB just this once.
2. You will submit a PDF-version of your answers on Compass on-or-before midnight of the due date.

Instructions

1. **Linear Algebra**: (*30 points*) Consider the following set of linear equations involving six equations in five unknowns $x_1, x_2, \ldots, x_5$.

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 & 6 & 2 \\ 1 & -1 & -1 & 4 & 0 \\ 2 & -2 & 2 & 4 & 0 \\ 2 & -2 & 0 & 6 & 0 \\ -4 & 4 & -8 & -4 & 0 \\ 4 & 0 & -2 & 14 & 4 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 2 \\ 7 \\ 18 \\ 16 \\ -40 \\ 2 \end{pmatrix}}_{\mathbf{y}}$$

   (a) (*2 points*) Show that there a solution to these equations.

   (b) (*10 points*) Present a general formula for the set of all solutions that satisfy these equations.

   (c) (*3 points*) Verify the general formula using MATLAB.

   (d) (*5 points*) Show that

$$\begin{pmatrix} -7\lambda_1 - 4\lambda_2 + 8 \\ -7\lambda_1 - 7\lambda_2 \\ 1 - \lambda_2 \\ -\lambda_2 \\ -7 + 7\lambda_1 + 7\lambda_2 \end{pmatrix} \tag{1}$$

   and

$$\begin{pmatrix} a \\ -7a - 7b \\ -\frac{8}{3}a - \frac{7}{3}b + \frac{11}{3} \\ -\frac{8}{3}a - \frac{7}{3}b + \frac{8}{3} \\ -7 + 7a + 7b \end{pmatrix}. \tag{2}$$

   are solutions to the equation $\mathbf{Ax} = \mathbf{y}$ identified earlier.

(e) (*5 points*) Present a cogent argument in support of the claim that even though the solution identified by equations 1 and 2 look different, they are in effect the same.

(f) (*5 points*) Show that if add the constraints

$$\begin{aligned} 7x_4 + x_5 &= 7 \\ x_2 + 2x_5 &= 7, \end{aligned}$$

in addition to those already in the requirement that $\mathbf{Ax} = \mathbf{y}$ as described in problem 1, there can be only one solution to the combined set of equation-s/constraints.

2. (*30 points*) **Generating RVs using the Inverse Transform Technique**: You may want to use the various C++ code samples on Compass (along with some graphing/plotting software) to verify/experiment-with things before you put your answers down for this problem.

(a) (*10 points*) The *triangular distribution* has a probability density function

$$f(x) = \begin{cases} x & 0 \leq x \leq 1 \\ 2-x & 1 < x \leq 2 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

In three/four sentences describe how you can generate i.i.d. r.v's that are distributed according to equation 3.

(b) (*10 points*) What is the i.i.d. distribution generated by repeated calls to the code shown below?

```
double blah1( )
{
 (double) x = get_uniform();
 if (x <= 0.5)
                return (1 + $\sqrt{2x}$);
 else
       return (1 − $\sqrt{2-2x}$);
}
```

(c) (*10 points*) What is the i.i.d. distribution generated by repeated calls to the code shown below?

```
double blah2( )
{
 (double) x = get_uniform();
 if (x <= 0.5)
                return (2 − $\sqrt{1-2x}$);
 else
       return ( $\sqrt{2x-1}$);
}
```

3. (*40 points*) **General Programming Concepts**:

   (a) (*10 points*) Consider the two versions of the code shown in figure 1. Suppose I type the string "kablooey" as input, give me the reasoning behind the output each of the programs generate[1].

```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    cout << c;
    f();
}
int main()
{
    f();
    cout << endl;
}
```

(a) Problem 3a(I)

```
#include <iostream>
#include <cmath>
using namespace std ;
void f()
{
    char c;
    cin.get(c);
    // this line reads character "c"
    if ('\n' == c)
        // '\n' is "return/newline"
        return;
    f();
    cout << c;
}
int main()
{
    f();
    cout << endl;
}
```

(b) Problem 3a(II)

Figure 1: (Two versions of the) Code for problem 3a.

   (b) (*10 points*) The *greatest common divisor* (GCD) of two positive numbers m and n is the largest number k that divides m and n. That is, $\frac{m}{k}$ and $\frac{n}{k}$ leaves no remainder. Consider the C++ code shown in figure 2. Using the fact that the GCD of two numbers does not change if the smaller number is subtracted from the larger number, show that the recursive funtion gcd(m, n) in the code shown in figure 2 implements the GCD computation.

   (**Note**: Please do not give me an illustrative example as a "proof" of that the code does what it is supposed to do. *I am looking for a rigorous attempt at showing the correctness of the code for any pair of numbers*.)

   (c) (*10 points*) What is the output of the C++ code shown in figure 3. Make sure you give me your reasoning for your answer.

[1] I am looking for an answer that justifies why we get to see the output generated by each of these programs.

```
#include <iostream>
using namespace std;
long gcd(long m, long n)
{
        if (m == n)
                return n;
        else if (m < n)
                return gcd(m,n-m);
        else
                return gcd(m-n, n);
}
int main()
{
        cout << gcd(259, 111) << endl;
}
```

Figure 2: Problem 3b.

```
#include <iostream>
using namespace std;
void print_integer (int num)
{
    putchar (num % 10 + 'A');
        if (num / 10)
                print_integer(num / 10);
}
int main()
{
        print_integer (1234);cout << endl;
}
```

Figure 3: Problem 3c.

What is the output of the above program? What is the reasoning behind the output that this program generates?

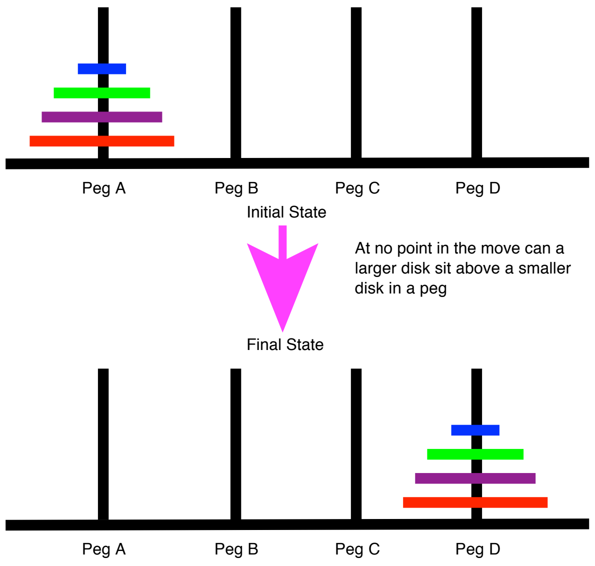
(d) (*10 points*) My review of computing contains the *Frame-Stewart* solution to the k-peg version of the *Tower of Hanoi* problem. Consider the following solution to the 4-peg version of the problem.

Alternate Solution to 4-Peg Tower of Hanoi Problem

```
1: int $k = \lfloor \sqrt{2 * n} \rfloor$;
2: Move_Using_Four_Pegs ($n - k$, source, intermediate_1, intermediate_2, destination);
```

```
3: Move_Using_Four_Pegs ($k-1$, source, intermediate_2, destination, intermediate_1);
4: cout << "Move the top disk from peg " << source << " to peg " << destination << endl;
5: Move_Using_Four_Pegs ($k-1$, intermediate_2, destination, intermediate_1, source);
6: Move_Using_Four_Pegs ($n-k$, intermediate_1, destination, intermediate_2, source);
```

Comment on the correctness of this algorithm. Will this work? (**You might want to spend a little time thinking about this problem before you wrote down your answer**)


Peg A
Peg B
Peg C
Peg D
Initial State
At no point in the move can a larger disk sit above a smaller disk in a peg
Final State
Peg A
Peg B
Peg C
Peg D