Biostat 615 Homework #5 (total 30 pts)

Due by Friday December 18th, 2015 noon as a compressed file "hw5.tar.gz" containing the required C++ source code or R code files for the following problems. Note that:

- You can either write a C++ program or an R script to solve each problem. Please select THREE problems to solve but implement AT LEAST ONE in C++.
- If you choose to write a C++ program, please note that:
  - Your programs should get input from either `cin` or `argv` according to the specific description given in each problem.
  - All `C++` programs (e.g., `program.cpp`) will be compiled and tested on the server `scs.itd.umich.edu` using the following command:

```
g++ -O -o program -I ~jiankang/Public/include program.cpp
```

  - You can include libraries `boost` and `Eigen` in your program
  - If you use your own header files that is not part of the C++ standard libraries, `boost` or `Eigen`, you should include the header files in your submission and make sure your program can be compiled using the above command without errors.
- If you choose to write an `R` script, please note that
  - Your `R` script should get input values from a file.
  - All `R` scripts (e.g., `program.R`) will be tested on the server `scs.itd.umich.edu` using the following command:

```
R --slave --args test.input < program.R
```

  - The file `test.input` contains the input values
  - In `program.R`, you need to use the following code to obtain input values

```
args = commandArgs(trailingOnly = TRUE)
input = scan(file=args[1],what=character())
# You can change option "what" for other data types
# e.g."double()" or "integer()"
```

  - If you would like to use `Rcpp` package, in your submitted `program.R`, you need to include the following code

```
library(Rcpp,lib.loc="~jiankang/Public/include")
```

  - To print out results, use function `cat()` instead of `print()`. For example

```
cat(10) #rather than using print(10)
```

- Make your algorithm as efficient as you can, as some of the test cases may be computationally challenging and your program will be terminated if it does not finish after running for 10 seconds and you will lose the points for those test cases. However, you should always turn in a program even if it is not very efficient so that you can at least get partial points.

- Five test cases will be provided for each problem. Please download them from the CANVAS.
- The first grading will start on Wednesday Dec 2, 2015.

Problem 1 - Mixture of Normal and Uniform Distributions (10 pts)

Suppose $x_1, \ldots, x_n$ are i.i.d. samples from a mixture of a Normal distribution and a uniform distribution. That is, $x_i \sim N(\mu, \sigma^2)$ with probability $0 < p < 1$ and $x_i \sim U(a, b)$ with probability $1 - p$. Suppose that we know μ is an integer and $\sigma = 1$, but we know nothing about p, a and b. Your task is to write a C++ program `mixNormUnif.cpp` or an `R script mixNormUnif.R` which, given $x_1, \ldots, x_n$, estimates and prints μ. Since μ should always be an integer, do NOT print in floating point format such as 1.0. Your C++ program should get its input from `cin`. The input is a sequence of n real numbers $x_1, \ldots, x_n$. Your `R` script will read these numbers from an input file. Example run for C++ program:

```
user@host:~/Private/biostat615$ ./mixNormUnif < test1.1.input
2
user@host:~/Private/biostat615$ ./mixNormUnif < test1.2.input
10
```

Example runs for `R` scripts:

```
user@host:~/Private/biostat615$ R --slave --args test1.1.input < mixNormUnif.R
2
user@host:~/Private/biostat615$ R --slave --args test1.2.input < mixNormUnif.R
10
```

The test input file "`test1.1.input`" has the following values:

```
1 2 2 3
```

The test input file "`test1.2.input`" has the following values:

```
0 1 2 3 4 5 6 7 9 9.5 10 10.5 11
```

Problem 2. Matrix Multiplication (10 pts)

Write a C++ program `matMult.cpp` or an `R` script `matMult.R` which, given a sequences of non-negative integers $a_1, a_2, \ldots, a_n$, where $n \leq 10,000$, and three positive integers k, i and j, computes and prints the last four digits (omit leading zeros, prints 0 if all zeros) of the (i, j)-th element of the k-th power of the following $n \times n$ matrix:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ a_2 & a_1 & a_2 & \cdots & a_{n-1} \\ a_3 & a_2 & a_1 & \cdots & a_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_n & a_{n-1} & a_{n-2} & \cdots & a_1 \end{pmatrix}$$

Your C++ program should get its input from `cin`. The input is a sequence of integers in the order of $k, i, j, a_1, a_2, \ldots, a_n$. Your `R` script will read these numbers from an input file. Please note that you may not even have enough memory to store the entire matrix if n is large.

Example runs for C++ program:

```
user@host:~/Private/biostat615$ ./matMult < test2.1.input
4
user@host:~/Private/biostat615$ ./matMult < test2.2.input
8116
```

Example runs for `R` scripts:

```
user@host:~/Private/biostat615$ R --slave --args test2.1.input < matMult.R
4
user@host:~/Private/biostat615$ R --slave --args test2.2.input < matMult.R
8116
```

The test input file "`test2.1.input`" has the following values:

```
1 2 3 5 4 3 2 1
```

The test input file "`test2.2.input`" has the following values:

```
8 1 1 3 2 1
```

Problem 3. Solving Linear Equation (10 pts)

Write a C++ program `linearSol.cpp` or `R` script `linearlSol.R` which, given a sequences of integers $b_1, b_2, \ldots, b_n$, computes and prints the maximum element of the solution vector x to the linear equation $Ax = b$, where A is an $n \times n$ matrix:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2 & \cdots & 2 \\ 1 & 2 & 3 & \cdots & 3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & \cdots & n \end{pmatrix}$$

and b is a $n \times 1$ column vector of $[b_1, b_2, \ldots, b_n]^T$. Your C++ program should get its input from `cin`. The input is a sequence of integers in the order of $b_1, b_2, \ldots, b_n$. Your `R` script should get its input from an input file. Please note that you may not even have enough memory to store the entire matrix if n is large. Also, the solution should always be an integer so do NOT print in floating point format such as 1.0.
Example runs for C++ program.

```
user@host:~/Private/biostat615$ ./linearSol < test3.1.input
1
user@host:~/Private/biostat615$ ./linearSol < test3.2.input
15
```

Example runs for the `R` script:

```
user@host:~/Private/biostat615$ R --slave --args test3.1.input < matMult.R
1
user@host:~/Private/biostat615$ R --slave --args test3.2.input < matMult.R
15
```

The test input file "`test3.1.input`" has the following values:

```
1 1 1 1
```

The test input file "`test3.2.input`" has the following values:

```
1 10 4 -4
```

Problem 4. Partially Observed Hidden Markov Model (10 pts)

Continuing on the biased coin example in lecture 15, now suppose that the coin jumps between two different biased states: state #1 and state #2. Suppose that the initial probability is $\pi = (p, 1-p)$, i.e., the coin is initially (for the first toss) in state #1 with probability p and in state #2 with probability $1-p$. The transition probabilities between adjacent tosses are given by the following matrix (e.g., if the coin is in state #1 for the i-th toss, then the coin will jump to state #2 with probability $1-a_1$ for the $(i+1)$-th toss).

$$A = \begin{pmatrix} a_1 & 1-a_1 \\ a_2 & 1-a_2 \end{pmatrix}$$

Furthermore, the emission probabilities are given by the following matrix (e.g., the probability of observing an outcome of head (denoted as `H`) for each toss is b_1 in state #1 and the probability of observing an outcome of tail (denoted as `T`) for each toss is $1-b_2$ in state #2).

$$B = \begin{pmatrix} b_1 & 1-b_1 \\ b_2 & 1-b_2 \end{pmatrix}$$

Write a C++ program `partialHMM.cpp` or an `R` script `partialHMM.R` which, given p, a_1, a_2, b_1, b_2 and a sequence of partially observed outcomes ("`H`" and "`T`" represent observed outcomes and "`.`" represents unobserved outcome) for a sequence of tosses, computes and prints a complete sequence of outcomes by filling in each unobserved outcome with the most likely outcome for that toss (i.e., the outcome with the largest marginal probability). Print `T` if the marginal probability for an unobserved outcome is greater than or equal to 0.5 for outcome `T`, and print `H` otherwise. Your C++ program should get its input from `argv`. Your `R` script should get this value from an input file. You can assume that p, a_1, a_2, b_1 and b_2 are real numbers between 0 and 1 (with 0 and 1 included). You may modify the forward-backward algorithm given in lectures.
Example runs for C++ program:

```
user@host:~/Private/biostat615$ ./partialHMM 0.5 0.95 0.2 0.5 0.9 H..T
HHHT
user@host:~/Private/biostat615$ ./partialHMM 0.5 0 1 1 0 T...
THTH
```

Example runs for the `R` script:

```
user@host:~/Private/biostat615$ R --slave --args test4.1.args < partialHMM.R
HHHT
user@host:~/Private/biostat615$ R --slave --args test4.2.args < partialHMM.R
THTH
```

The test input file "`test4.1.args`" has the following values:

```
0.5 0.95 0.2 0.5 0.9 H..T
```

The test input file "`test4.2.args`" has the following values:

```
0.5 0 1 1 0 T...
```