2011 BIOSTAT 615/815 Homework #4

- Due is Tuesday November 15th, 08:40AM (before the class starts).
- You need to both (1) hand-in your hard copy of the source code (using smaller font is fine), and (2) submit your source codes (ready to compile and run) in a zip or tar.gz compressed format by email.

Problem 1. Simulation of mixture of normal distribution

Write a C++ program that generate random variables from a mixture of two normal distributions. The program takes 6 ot 7 input arguments excluding the program name, in orders of

1. n : Number of independent random variables to generate
2. α : Probability of sampling from the first normal distribution
3. μ_1 : Mean of the first distribution
4. σ_1^2 : Variance of the first distribution
5. μ_2 : Mean of the second distribution
6. σ_2^2 : Variance of the second distribution
7. (Optional) seed for the pseudo-random number generator. When omitted, use timestamp to assign the seed.

The probability density function of the mixture distribution is defined as

$$f(x) = \alpha\mathcal{N}(\mu_1, \sigma_1^2) + (1-\alpha)\mathcal{N}(\mu_2, \sigma_2^2)$$

Below is an example run. The output does not have to be the same as below.

```
$ ./normMixSimul 10 0.1 0 1 10 1
-0.538301
10.1272
-0.688996
10.951
10.4099
10.4103
9.60538
10.3299
8.53939
9.34737
$ ./normMixSimul 10 0.1 0 1 10 1
0.881429
9.8162
10.9309
11.4946
11.6844
-0.784039
9.07033
9.09102
-0.54902
10.8843
$ ./normMixSimul 10 0.1 0 1 10 1 243248
9.19609
10.4071
9.16301
10.1444
-1.67249
10.1388
9.10533
```

```
-1.38556
10.711
10.6561
$ ./normMixSimul 10 0.1 0 1 10 1 243248
9.19609
10.4071
9.16301
10.1444
-1.67249
10.1388
9.10533
-1.38556
10.711
10.6561
```

Problem 2. General Hidden Markov Models

Write a C++ program that performs a general Hidden Markov Model inference, from any transition and emission matrix, initial state distribution, and sequence of observed output. All inputs are whitespace or tab-delimited matrices. For HMM with $|S|$ states, $|O|$ outputs, and T time slots, the transition matrix should be $|S| \times |S|$ matrix of `double`, emission matrix should be $|S| \times |O|$ matrix of `double`, pi should be $|S| \times 1$ matrix (or vector) of `double`, and outputs should be $T \times 1$ matrix of `int`. (For 815 students, the program still should be able to handle very large number of input sequences)

Below is an example run of the program with the following parameters

$$
\begin{aligned}
A &= \begin{pmatrix} 0.98 & 0.01 & 0.01 \\ 0.01 & 0.98 & 0.01 \\ 0.01 & 0.01 & 0.98 \end{pmatrix} \\
B &= \begin{pmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \\
\pi &= (1/3\ 1/3\ 1/3)^T
\end{aligned}
$$

```
$ ./generalHMM trans.txt emis.txt pis.txt obs.txt
TIME    OBS     Pr(0)   Pr(1)   Pr(2)   MLSTATE
1       1       0.1100  0.0015  0.8885  2
2       1       0.1071  0.0003  0.8925  2
3       1       0.1077  0.0003  0.8920  2
4       1       0.1120  0.0014  0.8867  2
5       1       0.1209  0.0104  0.8687  2
6       1       0.1342  0.0886  0.7771  2
7       0       0.1335  0.7782  0.0883  1
8       0       0.1196  0.8701  0.0102  1
9       0       0.1104  0.8884  0.0012  1
10      0       0.1059  0.8939  0.0002  1
11      0       0.1052  0.8947  0.0000  1
12      0       0.1081  0.8918  0.0000  1
13      0       0.1157  0.8843  0.0000  1
14      0       0.1306  0.8694  0.0001  1
15      0       0.1581  0.8418  0.0001  1
16      0       0.2080  0.7917  0.0002  1
17      0       0.2982  0.7008  0.0010  1
18      0       0.4605  0.5324  0.0070  1
19      1       0.7517  0.1885  0.0598  0
20      1       0.8153  0.1253  0.0594  0
21      1       0.8380  0.1152  0.0468  0
22      0       0.8584  0.1278  0.0138  0
```

```
23      0       0.8652  0.1263  0.0084  0
24      0       0.8731  0.1180  0.0088  0
25      1       0.8853  0.0960  0.0187  0
26      0       0.8848  0.0950  0.0201  0
27      0       0.8802  0.0842  0.0356  0
28      1       0.8429  0.0171  0.1400  0
29      1       0.8183  0.0091  0.1726  0
30      1       0.8033  0.0098  0.1869  0
```

You may want to use the revised `Matrix615` class below to read matrix from files.

```
#ifndef __MATRIX_615_H
#define __MATRIX_615_H
#include <vector>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <boost/tokenizer.hpp>
#include <boost/lexical_cast.hpp>

// Matrix615 class for BIOSTAT615/815 class
template <class T>  // template for stroing generic data types
class Matrix615 {
public:
  // public member variables : vector of vector (like 2D array)
  std::vector< std::vector<T> > data;

  // constructor with specified size
  Matrix615(int nrow, int ncol, T val = 0) {
    resize(nrow, ncol, val);
  }

  // constuctor from file
  Matrix615(const char* fileName) {
    readFromFile(fileName);
  }

  // default constructor : Empty Matrix
  Matrix615() {}

  // returns # rows or # columns
  int rowNums() { return (int)data.size(); }
  int colNums() { return ( data.size() == 0 ) ? 0 : (int)data[0].size(); }

  // resize the matrix into specified size
  void resize(int nrow, int ncol, T val = 0) {
    data.resize(nrow); // make n rows
    for(int i=0; i < nrow; ++i) {
      data[i].resize(ncol,val); // make n cols with default value val
    }
  }

  // read the matrix from a file (requires boost library)
  void readFromFile(const char* fileName) {
    // open input file
    std::ifstream ifs(fileName);
    if ( ! ifs.is_open() ) {
      std::cerr << "Cannot open file " << fileName << std::endl;
      abort();
    }
```

```
    // set up the tokenizer
    std::string line;
    boost::char_separator<char> sep(" \t");
    typedef boost::tokenizer< boost::char_separator<char> > wsTokenizer;

    // clear the data first
    data.clear();
    int nr = 0, nc = 0;
    while( std::getline(ifs, line) ) {
      if ( line[0] == '#' ) continue; // skip meta-lines starting with #
      wsTokenizer t(line,sep);
      data.resize(nr+1);
      for(wsTokenizer::iterator i=t.begin(); i != t.end(); ++i) {
        data[nr].push_back(boost::lexical_cast<T>(i->c_str()));
        if ( nr == 0 ) ++nc;  // count # of columns at the first row
      }
      if ( nc != (int)data[nr].size() ) {
        std::cerr << "The input file is not rectangle at line " << nr << std::endl;
        abort();
      }
      ++nr;
    }
  }
};

// Auxilary function to fill a vector from a file
// USAGE : fillVectorFromFile<type> (v, fileName)
template <class T>
void readFromFile(std::vector<T>& v, const char* fileName) {
  // open input file
  std::ifstream ifs(fileName);
  if ( ! ifs.is_open() ) {
    std::cerr << "Cannot open file " << fileName << std::endl;
    abort();
  }

  v.clear();
  std::string tok;
  while( ifs >> tok ) {
    v.push_back(boost::lexical_cast<T>(tok));
  }
}

#endif // __MATRIX_615_H
```

Problem 3. Simulation from Hidden Markov Models

Write a program that generates simulated outcomes from a Hidden Markov Model. The program takes the 4 or 5 parameters : (1) Number of simulation outputs or time slots (2) input transition matrix (3) input emission matrix (4) initial state distribution (5) Optional input for random seed (use timestamp as seed if not given)

An example run from the sample A, B, π as Problem 2 is

```
$ ./simulHMM 10 trans.txt emis.txt pis.txt
SEQ    STATE   OUT
1       1       0
2       1       0
3       1       0
4       1       0
```

```
5       1       0
6       1       0
7       1       1
8       1       1
9       2       1
10      2       1
```

For 615 students, simulate 10 sets of 50 outputs (500 total). For 815 students, simulate one set of 20,000 inputs. Use these outputs as input of the Problem 2, and compare the true states and the inferred states by the Viterbi algorithm, and report how many states were concordant to each other and how many were discordant when using the A, B, π matrix in problem 2.

If you're using UNIX or Mac environment. You may want to use the following sequences of commands to accomplish the task. (Example is given for 20,000 inputs)

```
$ ./simulHMM 20000 trans.txt emis.txt pis.txt > simul.20k.txt
$ cut -f 3 simul.20k.txt | tail -n +2 > in.20k.txt
$ ./generalHMM trans.txt emis.txt pis.txt in.20k.txt > out.20k.txt
$ paste simul.20k.txt out.20k.txt | cut -f 2,9 | sort | uniq -c
7264 0  0
 426 0  1
 407 0  2
 340 1  0
5195 1  1
  25 1  2
 362 2  0
  41 2  1
5940 2  2
   1 STATE      MLSTATE
```

In the example above, 18399(=7264+5195+5940) out of 20000 (92%) states were correctly inferred. How many states were correctly inferred in your examples?