Fall 2012 BIOSTAT 615/815 Problem Set #4

Due is Saturday November 10th, 2012 11:59PM by google document (shared to hmkang@umich.edu and atks@umich.edu) containing the source code and answers to the questions. Also email of the compressed tar.gz file (for Problem 1 and 3) containing all the source codes and the full R package (for Problem 2) as separate attachments.

All BIOSTAT615 and BIOSTAT815 students need to solve all three problems.

Problem 1. Histogram

Write a program that reads a file and produce a text version of histogram. Two required input arguments are (1) the input file name, and (2) the interval between breakpoints. Below is an example run.

```
$ ./hw-4-3
Usage: ./hw-4-3 [in.txt] [unit]

$ ./hw-4-3 rnorm.txt 0.5
[-4,-3.5):      1
[-3.5,-3):      9
[-3,-2.5):      49
[-2.5,-2):      168
[-2,-1.5):      427
[-1.5,-1):      906
[-1,-0.5):      1451
[-0.5,0):       1932
[0,0.5):        1900
[0.5,1):        1512
[1,1.5):        920
[1.5,2):        478
[2,2.5):        196
[2.5,3):        40
[3,3.5):        7
[3.5,4):        4
```

Some hints are below:

- Do not output the bins with zero observations.
- Using `std::map` would make things easier and efficient.
- To assign a floating-point value into a bin, the following snipplet will do it. `floor` function rounds the number down to the closest integer no greater than the value. You don't need to be concerned about overflow of underflow. (e.g. the case where the bin number becomes out of integer bound)

```
int bin = (int)floor(value/unit); // don't forget to include <cmath>
```

Problem 2. Hidden Markov Model Simulator

(a) Write a R function that simulates a hidden Markov model to produce random outcomes.

```
hmm.simulator <- function(pi, A, B, T, seed = NA) {
  ## pi = n * 1 vector
  ## A  = n * n matrix
  ## B  = n * m matrix
  ## T  = scalar
  out <- vector(length=T)
  if ( !is.na(seed) ) {
    ## attempt to set a random seed
  }
  ### FILL IN THE FUNCTION BELOW TO PRODUCE RANDOM OUT BASED ON HMM(pi,A,B)
  return(out); ### out
```

```
}
```

(b) Write a equivalent C++ program that simulate a hidden Markov model to produce random outcomes. Your interface to run the software must take each π, A, B as separate input file (which is readable by `Matrix615` class), and a integer argument T specifying the number of random outcomes to produce. You need to use `Eigen` and `boost` library for matrix operation and random number generation.

```
$ hw-4-1
  Usage: hw-4-1 [pi.txt] [A.txt] [B.txt] [T]
$ hw-4-1 pi.txt A.txt B.txt 10
0
1
0
1
1
0
0
0
0
0
```

Problem 3. Hidden Markov Model R package

Write an R package `hmm615` satisfying the following conditions

1. The package should be a "full" package, which is installable in a compatible platform by other users.
2. C++ implementation of hidden Markov model must be used.
3. Do not include `boost` or `Eigen` libraries in your implementation because it would make the installation trickier. You can simply achieve this by removing the functions that is not critical to this version (e.g. `Matrix615::readFromFile()` or `Matrix615::cloneToEigen`).
4. The package should contain a function `hmm(pi,A,B,obs)`, which take hidden Markov model parameters `pi, A, B`, and a vector of observations `obs`. The function must return two outcomes as list. (1) The forward-backward probability of each time and state given the observations and parameters. (2) The viterbi path of most likely states given the observations and parameters.
5. The results with valid output must be correct. Below is an example parameters, and input/outputs.

```
$ R CMD build hmm615
* checking for file 'hmm615/DESCRIPTION' ... OK
* preparing 'hmm615':
* checking DESCRIPTION meta-information ... OK
* cleaning src
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'hmm615_0.0.1.tar.gz'
$ R
> install.packages('hmm615_0.0.1.tar.gz')
> library(hmm615)
> pi <- c(0.5,0.5)
> A <- matrix(c(0.9,0.1,0.1,0.9),2,2,byrow=T)
> B <- matrix(c(0.5,0.5,0.9,0.1),2,2,byrow=T)
> obs <- c(0,1,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,0,0)
> r <- hmm615(pi, A, B, obs)
> r
$prob
          [,1]      [,2]
```

```
 [1,] 0.7202627 0.2797373
 [2,] 0.8541389 0.1458611
 [3,] 0.8420813 0.1579187
 [4,] 0.8976041 0.1023959
 [5,] 0.8508445 0.1491555
 [6,] 0.8700653 0.1299347
 [7,] 0.7593468 0.2406532
 [8,] 0.7313155 0.2686845
 [9,] 0.4340004 0.5659996
[10,] 0.2787784 0.7212216
[11,] 0.2020084 0.7979916
[12,] 0.1722147 0.8277853
[13,] 0.1771819 0.8228181
[14,] 0.2189465 0.7810535
[15,] 0.3146321 0.6853679
[16,] 0.2141165 0.7858835
[17,] 0.1684038 0.8315962
[18,] 0.1587517 0.8412483
[19,] 0.1812028 0.8187972

$path
 [1] 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
```

For your convenience, below is a skeleton of the R/C++ interface. What do need to do is to modify `HMM615.h` and `Matrix615.h` as needed, and use them to fill in the missing portion of the skeleton.

```
#include "HMM615.h"
#include "Matrix615.h"

#include <R.h>
#include <Rinternals.h>
#include <Rdefines.h>

extern "C" {
  SEXP hmm615(SEXP pi, SEXP A, SEXP B, SEXP obs) {
    SEXP prob, path, list, list_names;
    int n = 0, m = 0, T = 0;
    n = length(pi);
    T = length(obs);

    if ( isMatrix(A) ) {
      int *dimX = INTEGER(coerceVector(getAttrib(A,R_DimSymbol),INTSXP));
      if ( ( dimX[0] == n ) && ( dimX[1] == n ) ) { // do nothing
      }
      else {
        error("Dimension of A is expected to be %d by %d, but observed %d by %d",n,n,dimX[0],dimX[1]);
      }
    }
    else {
      error("A is not a matrix type");
    }

    if ( isMatrix(B) ) {
      int *dimX = INTEGER(coerceVector(getAttrib(B,R_DimSymbol),INTSXP));
      if ( dimX[0] == n ) {
         m = dimX[1];
      }
      else {
         error("Number of row in B is expected to be %d, but observed %d",n,dimX[0]);
      }
```

```
    }
    else {
      error("B is not a matrix type");
    }

    PROTECT( pi = AS_NUMERIC(pi) );
    PROTECT( A = AS_NUMERIC(A) );
    PROTECT( B = AS_NUMERIC(B) );
    PROTECT( obs = AS_NUMERIC(obs) );
    PROTECT( prob = allocMatrix(REALSXP, T, n) );
    PROTECT( path = allocVector(INTSXP, T) );

    double* p_pi = NUMERIC_POINTER(pi);
    double* p_A = NUMERIC_POINTER(A);
    double* p_B = NUMERIC_POINTER(B);
    double* p_obs = NUMERIC_POINTER(obs);
    double* p_prob = REAL(prob);
    int* p_path = INTEGER(path);

    // Part that does not need knowledge for R/C++ interface:
    // what you need to do is
    // create a HMM615 object
    // fill in the input parameter and data
    // run forwardBackward() and viterbi() algorithm
    // copy the outputs to p_prob and p_path

    char *names[2] = {"prob","path"};
    PROTECT(list_names = allocVector(STRSXP,2));
    SET_STRING_ELT(list_names, 0, mkChar(names[0]));
    SET_STRING_ELT(list_names, 1, mkChar(names[1]));
    PROTECT(list = allocVector(VECSXP,2));
    SET_VECTOR_ELT(list, 0, prob);
    SET_VECTOR_ELT(list, 1, path);
    setAttrib(list, R_NamesSymbol, list_names);

    UNPROTECT(8);

    return (list);
  }
}
```

NOTE: Your homework submission must include installable file `hmm615_0.0.1.tar.gz` (which is produced by `R CMD build hmm615`) as a separate attachement.