Problem Set 7

Thibault Doutre, Student ID 26980469

STAT243 : Statistical Computing
University of California, Berkeley

I worked on my own.

1

1 The goals of the simulation are:

- Assess the accuracy of the proposed asymptotic approximation in finite samples
- Examine the power of the EM test

The metrics they consider in assessing their methods are Type I errors and sensitivities (powers).

2 They had to choose the mixture parameters (alpha), the parameters of the gaussians (theta and sigma) and the sample size (n). The authors chose 8 different models with 1 or 2 levels olny for each input; this affects the statistical power of the test. Moreover, they chose 1 theta combianation for the mixture and only 2 type of mixtures (3 and 4 gaussians against 2).

3 It would have been good to have many inputs, or random inputs to asses the efficiency of the model. We actually don't know if they use these inputs due to the fact that it is well suited for the study or not.

4 In this paper the authors discretize the inputs, each into a very small number of levels:

- 2 levels for n
- 2 levels for the alphas
- 2 levels for the sigmas

Which is obviously not enough to conclude things like "Clearly, as the sample size increases, the power increases".
An option would be to have a l arger number of levels and carry out fractional factorial for choosing which treatment combinations to omit and so estimate the main effects. Another option would be to randomly choose each input in treating them as random variables (for the parameters of the mixture model for instance).

5 Their figures are not very efficient when presenting results because they don't

compare directly the typeI errors for m0=2 and m0=3. A good way to do that might be to plot a bunch of 4 boxplots with both data in them. Moreover, the tables are really hard to read, it would have been better to plot small graphs.

As for the standard errors, they are pretty large compared to the difference of the means between the first and the third iteration, especially for significance level=1%. They do not convince the reader that they did enough simulation because the levels are not large enough. Moreover only one value of theta is specified in the study.

6 This is true that the powers for n=400 are higher than for n=200 but one cannot conclude anything about global increase with n based on 2 points. We can see that the algorithm converges very quickly, 3 iterations. For the same sigmas, there is one better combination of alphas, and this is verified for n=200 and n=400. Idem when they change the variances.

7 estimated accuracy of results: barplots for typeI errors but missing for powers
descriptions of pseudorandom-number generators: NA
numerical algorithms: EM
computers: NA
programming languages: R
major software components that were used: NA

2

2.1

According to the algorithm provided in the unit 9, we have the following computations:

- 0 multiplication, 0 division
- 0 multiplication, $n-1$ divisions
- for $i = 2..n$
  - $i-1$ multiplications, 0 division
  - for $j = i+1..n$
    - $i-1$ multiplications, 1 division

Which is equivalent to:

- 0 multiplication, 0 division
- 0 multiplication, $n-1$ divisions
- for $i = 2..n$
  - $i-1$ multiplications, 0 division
  - $(i-1)(n-i)$ multiplications, $n-i$ divisions

Which is equivalent to:

- 0 multiplication, 0 division
- 0 multiplication, $n-1$ divisions
- $\sum_{i=2}^{n}(i-1)$ multiplications, 0 division
- $\sum_{i=2}^{n}(i-1)(n-i)$ multiplications, $\sum_{i=2}^{n}(n-i)$ divisions

Which is equivalent to:

- 0 multiplication, 0 division
- 0 multiplication, $n-1$ divisions
- $\frac{n(n-1)}{2}$ multiplications, 0 division
- $\frac{n(n-1)(n-2)}{6}$ multiplications, $\frac{(n-1)(n-2)}{2}$ divisions

Then, the total number of multiplications is: $\frac{n(n^2-1)}{6}$ and the total number of divisions is $\frac{n(n-1)}{2}$. As in the notes, we can see that the computational cost is $\frac{n^3}{6}+\frac{n^2}{2}+O(n)$.

2.2

We can indeed store the Cholesky upper triangular matrix U in the storage space that is used for the original matrix by overwriting the original matrix. Indeed, here is the algorithm based on the one provided in unit 9. The stars in the matrix show the components that have been modified at each step.

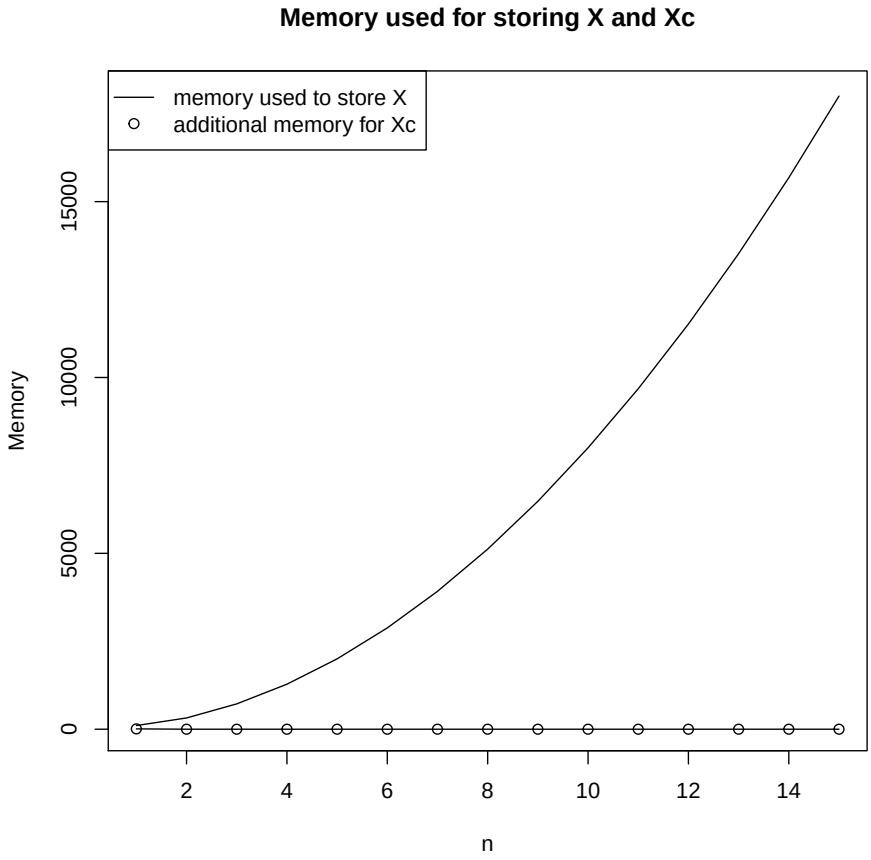
- $A_{11} = \sqrt{A_{11}}$
  $\begin{pmatrix} * & \dots & \cdot \\ \vdots & \ddots & \vdots \\ \cdot & \dots & \cdot \end{pmatrix}$
- For $j = 2..n$ $A_{1j} = A_{1j}/A_{11}$
  $\begin{pmatrix} * & \dots & \cdot \\ * & \ddots & \vdots \\ * & \dots & \cdot \end{pmatrix}$
- for $i = 2..n$
  - $A_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} A_{ki}^2}$
    $\begin{pmatrix} * & \dots & \cdot \\ * & * & \vdots \\ * & \dots & * \end{pmatrix}$
  - For $j = i+1..n$
    $* \ A_{ij} = \frac{A_{ij} - \sum_{k=1}^{i-1} A_{ki}A_{kj}}{A_{ii}}$
    $\begin{pmatrix} * & \dots & \cdot \\ * & * & * \\ * & \dots & * \end{pmatrix}$

  $\begin{pmatrix} * & * & * \\ * & * & * \\ * & \dots & * \end{pmatrix}$
- Return the upper triangular part of the matrix.

2.3

In order randomly generate a symmetric positive definite Matrix, I use Hadamard's lemma which states that a symmetric diagonally dominant matrix is invertible. Then, for different values of n, the size of this matrix, I compute both the memory used by storing X and Xc, respectively the covariance marix and the Upper triangular matrix issued from the Cholesky decomposition. We can observe that the memory used to create Xc is the same as the memory used to store X. This means that the "chol" algorithm from "base" in R is not overwriting the X matrix as specified before. Indeed, the algorithm allocate some space for the new Matrix and do the computations.

```
library(pryr)
memXc=c()
memX=c()
m0=0
for (n in seq(100,1500,100)){
  X0=matrix(runif(n*n),n,n)
  X=NULL
  m0=mem_used()
  X=(X0+t(X0))/n+diag(n)
  memX=c(memX,mem_used()-m0)
  m0=mem_used()
  X=chol(X)
  memXc=c(memXc,mem_used()-m0)
}
plot(memX/1000,type="l",xlab="n",ylab="Memory")
lines(memXc/1000,type="o")
legend("topleft",
       c("memory used to store X","additional memory for Xc"),
       pch=c(NA,1),lty=c(1,0))
title(main = "Memory used for storing X and Xc")
```



We can see that the additional memory used by the cholesky decomposition is not exactly zero, but 120 MB, which is much less than the memory used to store X. Apparently, the chol function uses more memory the first time it is computed in the loop.

```
memX

##  [1]   104248   320168   720168  1280168  2000168
   2880168
##  [7]  3920168  5120168  6480168  8000168  9680168
   11520168
## [13] 13520168 15680168 18000168

memXc
```

```
##   [1] 9376   120   120   120   120   120   120   120   120   120
       120
## [12]   120   120   120   120
```

3

3.1

```
# Initialize Definite positive matrix
n=5000
X0 = matrix(sample(0:9,n*n,replace=TRUE),n,n)
X = crossprod(X0,X0)

# Initialize vector
y = matrix(sample(0:9,n,replace=TRUE),ncol=1)

# Compute elapsed time
# Inverse and %*%
system.time({Xi=solve(X);xinverse=Xi%*%y})

##    user  system elapsed
## 219.025   0.567 219.921

# Solve
system.time({xsolve=solve(X,y)})

##    user  system elapsed
##  25.208   0.088  25.322

# Cholesky
system.time({U=chol(X);
xchol=backsolve(U, backsolve(U, y, transpose = TRUE))})

##    user  system elapsed
##  17.192   0.073  17.280
```

Here we have to look at the user time , which is the total time spent to compute the querries such as if everything were running in a single thread. The naive way should be 6 times slower than the Cholesky. Here we see the time to solve the equations and not only the time spent to do the decomposition or the inverse matrix. The solve function calls the internal function $.Internal(La\_solve(a, b, tol))$.

3.2

Are the results for b the same numerically for the different methods (up to machine precision)?

```
# Inverse and Solve
max(xinverse-xsolve)

## [1] 1.62359e-12

# Cholesky and Solve
max(xchol-xsolve)

## [1] 1.6477e-05

# Inverse and Cholesky
max(xinverse-xchol)

## [1] 1.771807e-05
```

I relate this to the condition number of the calculation:

```
# Inverse and Solve
min_cond1=sqrt(sum((xinverse-xsolve)^2))/sqrt(sum((xsolve)^2))*10^16
min_cond1

## [1] 49.81661

# Cholesky and Solve
min_cond2=sqrt(sum((xchol-xsolve)^2))/sqrt(sum((xinverse)^2))*10^16
min_cond2

## [1] 1073477077

# Inverse and Cholesky
min_cond3=sqrt(sum((xinverse-xchol)^2))/sqrt(sum((xchol)^2))*10^16
min_cond3

## [1] 1073477184
```

4

Since $\Sigma \in \mathbb{M}_p(\mathbb{R})$ is symmetric positive definite i.e. in $S_p^{++}(\mathbb{R})$, Σ^{-1} is also in $S_p^{++}(\mathbb{R})$. Using Cholesky decomposition, we have:

$$\exists P \in \mathbb{UT}_p^+(\mathbb{R}) \quad s.t. \quad \Sigma = P^T P. \tag{1}$$

Using the property that the inverse of an upper triangular matrix is an upper triangular matrix, we have:

$$\Sigma^{-1} = P^{-1}P^{-T} = U^T U \quad with \quad U = P^{-T} \in \mathbb{UT}_p^+(\mathbb{R}). \tag{2}$$

Therefore, the generalized least squares problem becomes:

$$\begin{aligned} & Find \quad \hat{\beta} \quad s.t. \quad X^T U^T U X \hat{\beta} = X^T U^T U Y. \\ i.e. \quad & Find \quad \hat{\beta} \quad s.t. \quad (UX)^T (UX)\hat{\beta} = (UX)^T (UY). \end{aligned} \tag{3}$$

Therefore, the generalized least squares model (GLS) is the ordinary least squares (OLS) with a change of basis characterized by U. In order to implement the GLS we first compute the Cholesky decomposition of Σ in order to have P. Then, taking the inverse of the transpose of P, we get U which is used to change X in UX and Y in UY. Finally, we use the Cholesky decomposition of X in order to solve the OLS efficiently, like in the previous problem.
Even more efficiently, we can use the *backsolve* function of R in order to get UX and UY directly from P.

```r
n=100
p=10
y=matrix(runif(n),nrow=n,ncol=1)
X0=matrix(runif(n*n),n,n)
S=(X0+t(X0))/n+diag(n)
X0=matrix(runif(n*n),n,n)
X=(X0+t(X0))/n+diag(n)

gls = function(X,Y,S){
  # X is a nxp matrix
  # Y is a vector of length n
  # S is a covariance matrix
  P=chol(S)
  Xbis=backsolve(P,X,transpose = TRUE)
  Ybis=backsolve(P,Y,transpose = TRUE)
  V=chol(tcrossprod(Xbis,Xbis))
 backsolve(Xbis, backsolve(Xbis, crossprod(Xbis,Ybis), transpose = TRUE))

}
head(gls(X,y,S))
```

```
##              [,1]
## [1,] -0.08432856
## [2,]  0.23824493
## [3,] -0.02879963
## [4,] -0.17175878
## [5,]  0.40518385
## [6,] -0.14102763
```

5

5.1

Let $v \in \mathbb{R}^{\mathbb{p}}$ be a right singular vector of $X \in \mathbb{M}_{n,p}(\mathbb{R})$ and $\lambda \in \mathbb{R}$ be the associated singular value.
$\exists u \in \mathbb{R}^{\mathrm{n}}$ such that:

$$Xv = \lambda u \tag{4}$$

$$X^T u = \lambda v \tag{5}$$

Then:

$$X^T X v = X^T(\lambda u) = \lambda X^T u = \lambda^2 v \tag{6}$$

Which proves that v is an eigen vector of $X^T X$ associated with the eigenvalue λ and associated eigenvalue is λ^2. There exists p distinct singular values for X, so we have found p distinct eigen values of $X^T X$. Therefore, reciprocally every eigenvalue of $X^T X$ can be expressed as the square of one right singular value of X.

5.2

Let $X \in \mathbb{M}_{n,p}(\mathbb{R})$ and note M the product $X^T X$. M is clearly symmetric and belongs to $\mathbb{M}_p(\mathbb{R})$. Therefore let's note $(v_i)_{1 \leq i \leq p}$ its eigen vectors respectively associated with the eigenvalues $(\sigma_i)_{1 \leq i \leq p}$. From before we have the existence of $(\lambda_i)_{1 \leq i \leq p}$ such that $\forall i, \lambda_i^2 = \sigma_i$.

$$\forall x \in \mathbb{R}^{\mathbb{p}}, \exists (\sigma_{\mathbb{i}})_{\mathbb{1} \leq \mathbb{i} \leq \mathbb{p}} \qquad \text{such that} \qquad x = \sum_{i=0}^{p} \alpha_i v_i \tag{7}$$

Then using the orthogonality of the eigen vectors of M we have $\forall x \in \mathbb{R}^{\mathbb{p}}$:

$$\begin{aligned} x^T M x &= (\sum_{i=0}^{p} \alpha_i v_i)^T M (\sum_{i=0}^{p} \alpha_i v_i) \\ &= (\sum_{i=0}^{p} \alpha_i v_i)^T (\sum_{i=0}^{p} \alpha_i \lambda_i^2 v_i) \\ &= \sum_{i=0}^{p} \alpha_i^2 \lambda_i^2 ||v_i||^2 \geq 0 \end{aligned} \tag{8}$$

Therefore M is a symmetric positive matrix.

5.3

Equivalently:

$$\begin{aligned} Xv = \lambda v &\iff Zv = \lambda v + cv \\ &\iff Zv = (\lambda + c)v \end{aligned} \tag{9}$$

Therefore the eigenvalues of Z are $\{\lambda + c | \lambda \in \mathbb{S}_p(\mathbb{R})\}$. In order to compute these eigenvalues we can add c to every eigenvalue of X. The complexity here is exactly n additions.