Homework 5

Group 12

1.

Generating a very large number of observations from standard normal distribution (e.g., 10^{20}) is **not feasible** with current computing resources due to both memory and computational time constraint.

Memory Limitation

- Each double-precision number takes 8 bytes.
- Storing 10^{20} numbers would require 8×10^{20} bytes, or 8×10^{8} terabytes (800 exabytes).

Practical Strategy

To approach this problem in practice, we **divide the total number of required observations into manageable chunks** (e.g., 10^{7} per chunk), which can fit into memory.

- **Using parallel computing** to generate each chunk simultaneously on different CPU cores.
- **Discard each chunk after use** to avoid memory overload.

Parallel Algorithm Outline

1. **Set the total number of observations** to generate.
2. **Choose a chunk size** that fits in memory (e.g., 10^{7}).
3. **Calculate the number of chunks**:

$$\text{Number of chunks} = \frac{\text{Total number of observations}}{\text{chunk size}}$$

4. **For each chunk (in parallel),**
    - Generate observations from standard normal population for that chunk.
    - Remove the chunk from memory before proceeding.
    - We were asked only to generate the sample, so we should store the sample but due to memory constraint we are bound to remove the chunks of observations after generation. We can store summary statistics for each chunk if it is needed, provided for that atleast the system has sufficient memory.

R Code

```r
library(parallel)
library(foreach)
library(doParallel)


total.obsns <- 1e11  # number of obsns to generate
chunk.size <- 1e7    # number of ovsns to generate per chunk
n.chunks <- total.obsns / chunk.size   # number of chunks
n.cores <- detectCores() - 2    # number of cores using

cl <- makeCluster(n.cores)
registerDoParallel(cl)

# tracking the required time
start <- Sys.time()

# generating observations in parallel
foreach(i = 1:n.chunks) %dopar% {
  set.seed(i)
  x <- rnorm(chunk.size)    # generating from N(0,1)
  rm(x)                    # removing from memory
}

end <- Sys.time()
req.time <- end - start
req.time
```

Observations

- **Chunking** allows us to work within hardware limitations.
- **Parallelization** speeds up the process by distributing chunks across multiple CPU cores.
- We observe that the time taken to generate sample of size 10^7, 10^8, 10^9, 10^{10} & 10^{11} are 1.3 secs, 1.6 secs, 1.52 mins, 2.09 mins and 17.87 mins respectively.
- **We can say It is impossible to generate and store** 10^{20} observations in practice.
- Thereby,we conclude that Even with parallel computing and the fastest supercomputers, generating and processing 10^{20} numbers would take thousands of years.

2.

Theoretical Background

Given that,

$$X = (X_1, \ldots, X_k) \sim N_k(\mathbf{0}_{k\times 1}, \Sigma_{k\times k})$$

Where

$$\Sigma_{k\times k} = \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{bmatrix}_{k\times k} = (1-\rho)I_k + \rho \mathbf{1}_k \mathbf{1}_k',$$

Given that, $k = 10^{20}$ and $\rho = 0.6$.

We have the following result,

$$\|X\|^2 = \sum_{i=1}^{k} x_i^2 \overset{d}{=} \sum_{i=1}^{k} \lambda_i Z_i^2,$$

where $\lambda_i' s$ are eigenvalues of Σ and $Z_i \overset{\text{i.i.d.}}{\sim} N(0,1), i = 1, 2, \ldots, k$.

- I_k has all eigenvalues 1.
- $\mathbf{1}_k \mathbf{1}_k'$ has k as its only nonzero eigenvalue with algebraic multiplicity 1.

So,

$$\lambda_1 = (1-\rho) + \rho k = 1 + (k-1)\rho$$

$$\lambda_2 = \lambda_3 = \cdots = \lambda_k = 1 - \rho$$

Probability Calculation

We want to estimate,

$$\begin{aligned} \mathbb{P}(\|\chi\| > 0.75) &= \mathbb{P}(\|\chi\|^2 > 0.75^2) \\ &= \mathbb{P}\left(\lambda_1 Z_1^2 + \lambda_2 \sum_{i=2}^{k} Z_i^2 > 0.75^2\right), \end{aligned}$$

where $Z_i \overset{\text{i.i.d.}}{\sim} N(0,1), i = 1, 2, \ldots, k$.

Monte Carlo Estimate

So, Monte Carlo estimate of $P(\|\chi\| > 0.75)$ is, given by

$$\frac{1}{M} \sum_{m=1}^{M} I_{\left\{\lambda_1 Z_{m1}^2 + \lambda_2 \sum_{i=2}^{k} Z_{mi}^2 > 0.5625\right\}},$$

where, $M =$ Monte Carlo replication number.

Taking $M = 500$.

Algorithm

1. **Set** `track = 0`.

2. For each Monte Carlo sample:

   - Draw $Z_i \overset{iid}{\sim} N(0,1), i = 1, \ldots, k$ one by one.
   - Compute $\lambda_1 Z_1^2 + \lambda_2 \sum_{i=2}^{l} Z_i^2 = c$, say, $l = 2, \ldots, k$.
   - If $c > 0.75^2$, stop generating Z_i's and set `track = track + 1`.

3. Repeat step 2, M times.

4. **Return** `track / M`.

R Code

```r
set.seed(123)
k <- 1e20    # dimension of the multivariate normal
rho <- 0.6    # correlation
lambda_1 <- 1 + (k-1)*0.6
lambda_2 <- 0.4

m <- 500    # Monte Carlo replication number
track <- 0
for(i in 1:m)
{
  z <- rnorm(1)
  iter <- 1    # to track the #of z's generated
  parial.sq.norm <- lambda_1 * z^2     # tracks partial norm of the k-dimensional observation
  if(parial.sq.norm > 0.75^2)
  {
    track <- track + 1
    next
  }

  while(parial.sq.norm <= 0.75^2 & iter <= k)
  {
    z <- rnorm(1)
    iter <- iter + 1
    parial.sq.norm <-  parial.sq.norm + lambda_2 * z^2
    if(parial.sq.norm > 0.75^2)
    {
      track <- track + 1
      break
    }
  }
}

prob.est <- track / m   #  estimated required probability
```

Result

The estimated probability with $M = 500$ is,

```
prob.est
```

```
[1] 1
```

Conclusion

The sequential-stopping Monte Carlo method efficiently estimates $P(\|X\| > 0.75)$ in dimension 10^{20}, confirming the probability is essentially 1.