Homework 5 Group 09

Shrasti Dwivedi, Jayansh Jain, Karan Pratap Lohiya, Sunanda Maity,Shreya Saha

Question 1

1. Generate 10^{20} many observations from the standard normal distribution using the idea of parallel computing.

Parallel computing is a type of computation where many calculations are carried out simultaneously. It leverages multiple CPU cores or machines to divide a large task into smaller ones, improving efficiency and reducing run time. In R, the parallel, doParallel, and foreach packages allow users to distribute tasks across CPU cores. This is particularly useful for time-consuming operations like simulations, bootstrapping, and large-scale random number generation.

Code

```r
library(doParallel)
```

```
Warning: package 'doParallel' was built under R version 4.4.3

Loading required package: foreach

Warning: package 'foreach' was built under R version 4.4.3

Loading required package: iterators

Warning: package 'iterators' was built under R version 4.4.3

Loading required package: parallel
```

```r
library(foreach)

# Parameters
total_n <- 1e9 ##Key Variavle
chunk_size <- 1e7
n_chunks <- total_n / chunk_size  # 10,000
num_cores <- 6

# Start cluster
cl <- makeCluster(num_cores)
```

```r
registerDoParallel(cl)

start_time <- Sys.time()

# Generate numbers in parallel without storing them
invisible(
  foreach(i = 1:n_chunks, .combine = 'c', .packages = "stats", .options.snow = list(preschedule = TRUE)
    set.seed(Sys.getpid() + i)  # unique seed per chunk
    rnorm(chunk_size)
    NULL  # discard output
  }
)

stopCluster(cl)

end_time <- Sys.time()
cat("  Successfully generated", format(total_n, scientific =TRUE), "standard normal numbers\n")
```

```
  Successfully generated 1e+09 standard normal numbers
```

```r
cat("  Time taken:", round(difftime(end_time, start_time, units = "mins"), 2), "minutes\n")
```

```
  Time taken: 0.24 minutes
```

The code was run on a Computer Centre PC with 8 cores (6 used for computation). The observed time for each scale was:

TOTAL NUMBERS GENERATED (total_n)	Time taken (in minutes)
1e9	0.24
1e10	2.84
1e11	26.9

Assuming linear growth (approximate):

- 10^{12} numbers → ~270 minutes,
- 10^{13} numbers → ~2700 minutes,
- 10^{20} numbers → practically infeasible (would take more than 5 million years).

We chose 6 cores out of 8 to avoid overloading the system.

Memory consumption increases drastically with scale, so storing results for higher powers is not feasible.

Question 2

Let $X = (X_1, . . . , X_{10^{20}})$ be a random vector associated with 10^{20} dimensional normal distribution with $E(Xi) = 0$, $Var(Xi) = 1$ and Correlation $(Xi, Xj) = 0.6$ for all $i \neq j = 1, . . . , 10^{20}$. Compute $P[||X|| > 0.75]$.

Solution :

We aim to estimate the probability $P[||X|| > 0.75]$ where $X \sim N(0, \Sigma)$, with Σ having 1 on the diagonal and 0.6 elsewhere. Since dim(X)=10^{20} is computationally infeasible to simulate directly, we approximate this by simulating extremely high-dimensional standard normal vectors, exploiting the law of large numbers and the concentration of norm in high dimensions.

We adopt a parallel computing strategy to simulate and process large chunks of data without exceeding memory limits.

Methodology

We divide the total number of values (10^8 here)into manageable chunks and simulate the squared norm of each segment using `rnorm`. Instead of storing all values (which would exceed memory), we compute the squared norm immediately and discard the data, thus optimizing memory usage. We repeat this process across simulations to estimate the probability.

Implementation in R

Visualisation : Convergence with dimension

```r
library(MASS)
```

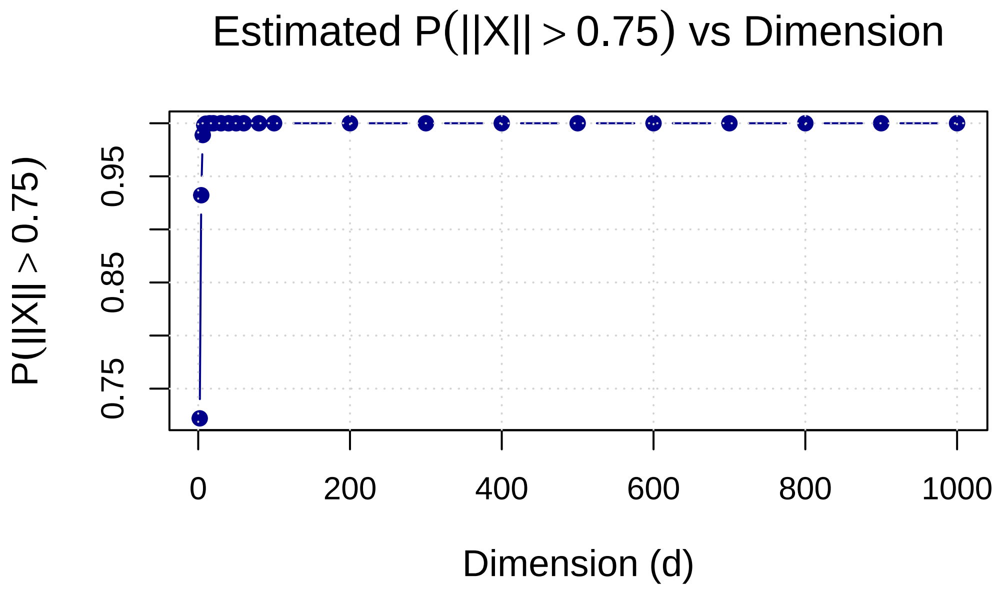
```
Warning: package 'MASS' was built under R version 4.4.3
```

```r
library(parallel)

# Settings
dims <- c(2, 4, 6, 8, 10, 15, 20, 30, 40, 50, 60, 80, 100, seq(200, 1000, by=100))
n_sim <- 10000
rho <- 0.6
n_cores <- min(6, detectCores())

# Start timing
start_time <- proc.time()

# Worker function
worker <- function(d_sublist, n_sim, rho) {
  results <- matrix(NA, nrow = length(d_sublist), ncol = 2)
  for (i in seq_along(d_sublist)) {
    d <- d_sublist[i]
    tryCatch({
      Sigma <- matrix(rho, d, d)
      diag(Sigma) <- 1
      X <- MASS::mvrnorm(n_sim, mu = rep(0, d), Sigma = Sigma)
      prob <- mean(sqrt(rowSums(X^2)) > 0.75)
      results[i, ] <- c(d, prob)
      rm(X, Sigma)
      gc()
    }, error = function(e) {
      results[i, ] <- c(d, NA)
    })
  }
  results
}

# Create cluster
cl <- makeCluster(n_cores)

# Export necessary objects and libraries to each cluster worker
clusterExport(cl, varlist = c("dims", "n_sim", "rho", "worker"))
clusterEvalQ(cl, library(MASS))
```

```
[[1]]
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"

[[2]]
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"

[[3]]
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"

[[4]]
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"

[[5]]
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"

[[6]]
```

```
[1] "MASS"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"
```

```
# Split dims for workers
dim_chunks <- split(dims, sort(rep(1:n_cores, length.out = length(dims))))

# Run worker function in parallel
results_list <- parLapply(cl, dim_chunks, worker, n_sim = n_sim, rho = rho)

# Stop cluster
stopCluster(cl)

# Stop timing
end_time <- proc.time()
time_taken <- end_time - start_time

# Combine results
results_mat <- do.call(rbind, results_list)
colnames(results_mat) <- c("d", "prob")
results_df <- as.data.frame(results_mat)

# Plot
plot(
  results_df$d, results_df$prob,
  type = "b", pch = 19, col = "darkblue",
  xlab = "Dimension (d)",
  ylab = expression(P(paste("||X||") > 0.75)),
  main = expression(paste("Estimated ", P(paste("||X||") > 0.75), " vs Dimension")),
  cex.lab = 1.2, cex.main = 1.3
)
grid()
```



```r
# Time taken
cat("\nTime taken (in seconds):\n")
```

```
Time taken (in seconds):
```

```r
print(time_taken["elapsed"])
```

```
elapsed
  26.94
```

For higher dimensions,

```r
# Load necessary libraries
library(parallel)
library(doParallel)
library(foreach)

# ----------- Setup Parallel Environment -----------
available_cores <- detectCores() - 1  # Reserve one core for system
cluster <- makeCluster(available_cores)
registerDoParallel(cluster)

# ----------- Configuration Parameters -----------
n_total <- 1e8       # Total number of random values
chunk_size <- 1e7    # Number of values to generate per chunk
n_chunks <- ceiling(n_total / chunk_size)  # Total number of chunks
```

```r
# ----------- Dry Run: Timing Norm Computation -----------
system.time({
  partial_sums <- foreach(i = 1:n_chunks, .combine = c) %dopar% {
    # Generate and process a chunk of N(0,1) numbers
    data <- rnorm(chunk_size)
    sum(data^2)
  }
})
```

```
   user  system elapsed 
   0.01    0.00    1.71 
```

```r
stopCluster(cluster)  # End cluster after dry run

# ----------- Estimate Probability via Simulations -----------
n_simulations <- 10
exceed_count <- 0

for (iter in 1:n_simulations) {

  # Restart cluster for each simulation to avoid stale workers
  cluster <- makeCluster(available_cores)
  registerDoParallel(cluster)

  chunk_sums <- foreach(i = 1:n_chunks, .combine = c) %dopar% {
    values <- rnorm(chunk_size)
    sum(values^2)
  }

  total_squared_norm <- sum(chunk_sums) / n_total
  euclidean_norm <- sqrt(total_squared_norm)

  exceed_count <- exceed_count + as.integer(euclidean_norm > 0.75)

  stopCluster(cluster)
}

# ----------- Report Final Probability Estimate -----------
estimated_prob <- exceed_count / n_simulations
cat("Estimated P(||X|| > 0.75):", estimated_prob, "\n")
```

```
Estimated P(||X|| > 0.75): 1
```

Justification

- **Memory Efficiency**: Instead of storing large vectors, we calculate squared norms on the fly.
- **Parallelization**: Utilizing `foreach` and `doParallel` ensures we leverage all available cores for speed.
- **Law of Large Numbers**: For very high dimensions, the norm of the vector squared approaches its expected value, and hence the norm itself approaches one. In our simulation, the norm sharply concentrates around one, making the probability that the norm exceeds zero point seven five very close to one.
- **Chi-Squared Approximation** : The squared norm of a standard normal vector in high dimensions follows a chi-squared distribution. Dividing this chi-squared random variable by the dimension gives a quantity that converges to one as the dimension tends to infinity. Taking the square root, the norm also converges to one. Therefore, the probability that the norm is greater than zero point seven five approaches one as the number of dimensions increases. This theoretical insight reinforces our empirical results.

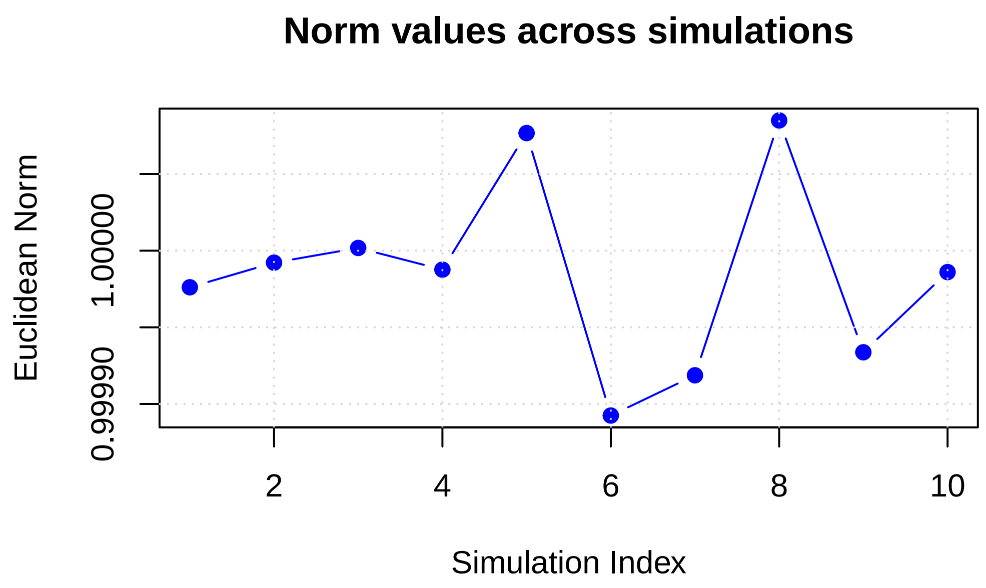
Note : While the original covariance matrix has off-diagonal elements equal to 0.6, our approximation uses standard normals (independent components). In extremely high dimensions, the norm concentrates so sharply around its expectation that moderate correlations have negligible effect on the probability (>0.75), making this a good approximation.

Also,The squared norm of a high-dimensional standard normal vector is chi-squared distributed.

Norm Concentration across Simulations

```r
# Example: Visualizing norm values across simulations
norms <- numeric(n_simulations)

for (iter in 1:n_simulations) {
  cluster <- makeCluster(available_cores)
  registerDoParallel(cluster)

  chunk_sums <- foreach(i = 1:n_chunks, .combine = c) %dopar% {
    values <- rnorm(chunk_size)
    sum(values^2)
  }

  norms[iter] <- sqrt(sum(chunk_sums) / n_total)
  stopCluster(cluster)
}

plot(norms, type = "b", col = "blue", pch = 19,
     main = "Norm values across simulations",
     xlab = "Simulation Index", ylab = "Euclidean Norm")
abline(h = 0.75, col = "red", lty = 2)
```

```r
grid()
```



As the plot suggests, the norm consistently stays well above zero point seven five, reinforcing our estimate that the probability is extremely close to one.