Solutions to Exercises

Portfolio Optimization: Theory and Application
Chapter 3 – Financial Data: I.I.D. Modeling

Daniel P. Palomar (2025). *Portfolio Optimization: Theory and Application.*
Cambridge University Press.

portfoliooptimizationbook.com

Exercise 3.1: Unbiasedness and consistency of sample mean estimator

Consider a univariate Gaussian-distributed i.i.d. time series with mean 0.01 and variance 1, $x_t \sim \mathcal{N}(0.01, 1)$, $t = 1, \ldots, T$.

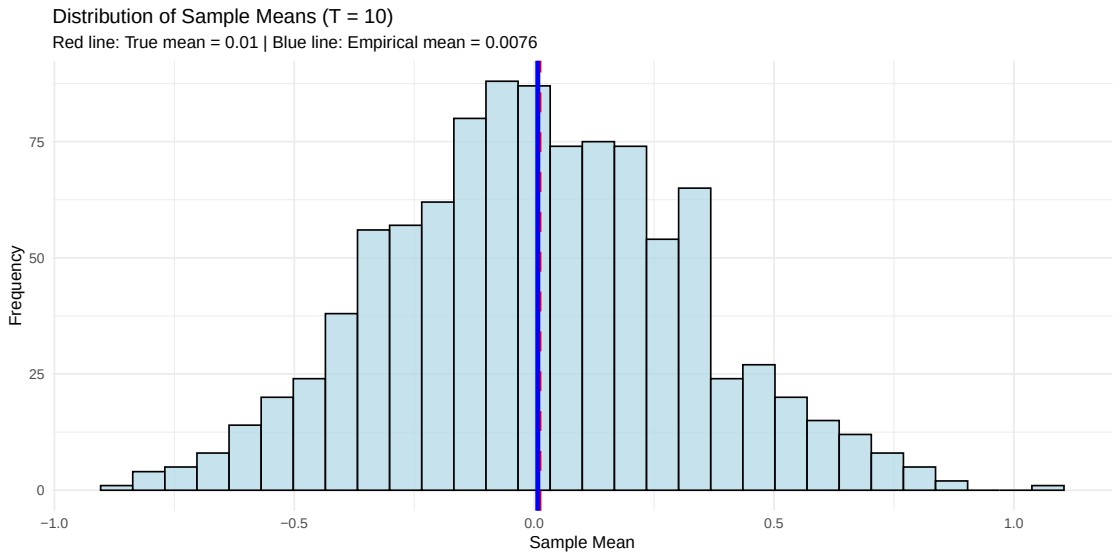
a. Generate data for $T = 10$ and compute the sample mean. Repeat the experiment multiple times and plot the histogram of the estimated mean value. Confirm that the expected value of the histogram coincides with the true mean value.
b. Now repeat the experiment with $T = 20$ observations and compare the histograms (also compute the standard deviation of each histogram).
c. Finally, repeat the experiment multiple times, for different numbers of observations $T = 10, 20, \ldots, 100$, and plot the mean squared error of the estimation as a function of T.

Solution

```
library(ggplot2)
library(dplyr)
library(gridExtra)

# Set parameters
true_mean <- 0.01
true_variance <- 1
true_sd <- sqrt(true_variance)
n_experiments <- 1000  # Number of repetitions for each T

# Set seed for reproducibility
set.seed(123)
```

a. Generate data for $T = 10$ and compute the sample mean. Repeat the experiment multiple times and plot the histogram of the estimated mean value. Confirm that the expected value of the histogram coincides with the true mean value.

```r
# Part A: Generate data for T=10 and compute sample means
T1 <- 10
sample_means_T10 <- replicate(n_experiments, {
  data <- rnorm(T1, mean = true_mean, sd = true_sd)
  mean(data)
})

# Create histogram for T=10
hist_data_T10 <- data.frame(sample_means = sample_means_T10)

ggplot(hist_data_T10, aes(x = sample_means)) +
  geom_histogram(bins = 30, fill = "lightblue", color = "black", alpha = 0.7) +
  geom_vline(aes(xintercept = true_mean), color = "red", linewidth = 1.5, linetype = "dashed") +
  geom_vline(aes(xintercept = mean(sample_means)), color = "blue", linewidth = 1.5) +
  labs(title = "Distribution of Sample Means (T = 10)",
       x = "Sample Mean",
       y = "Frequency",
       subtitle = paste("Red line: True mean =", true_mean,
                        "| Blue line: Empirical mean =", round(mean(sample_means_T10), 4))) +
  theme_minimal()
```

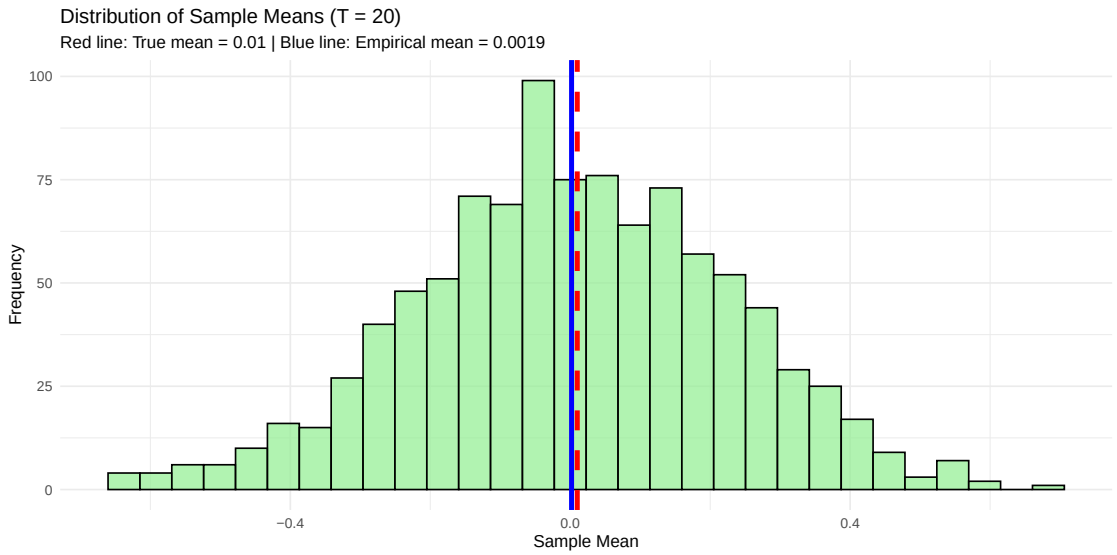


```r
# Verify unbiasedness
cat("Part A Results (T = 10):\n")
cat("True mean:", true_mean, "\n")
cat("Empirical mean of sample means:", round(mean(sample_means_T10), 6), "\n")
cat("Standard deviation of sample means:", round(sd(sample_means_T10), 6), "\n")
cat("Theoretical standard error:", round(true_sd/sqrt(T1), 6), "\n\n")
```

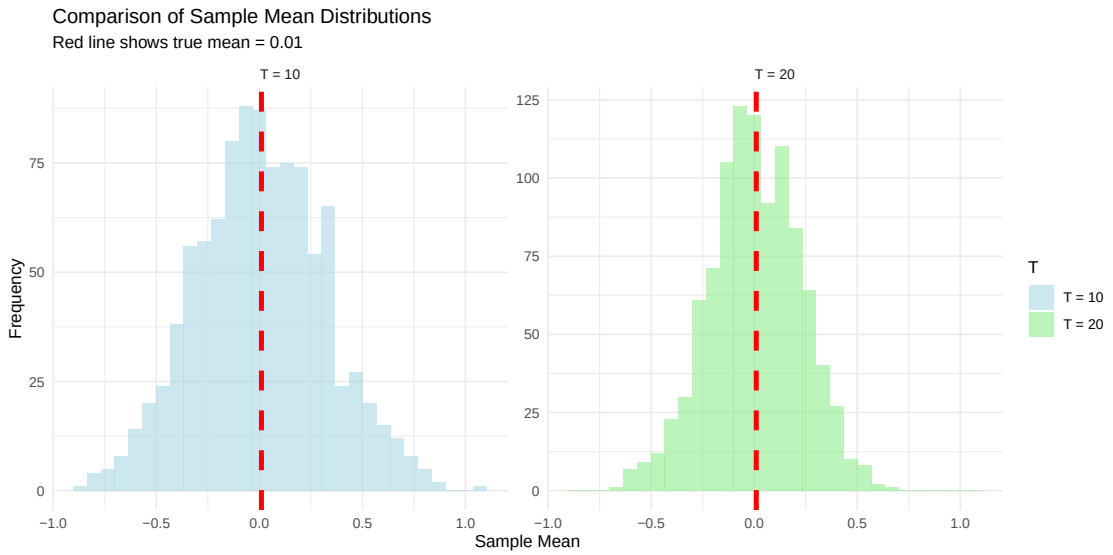
```
Part A Results (T = 10):
True mean: 0.01
Empirical mean of sample means: 0.007628
Standard deviation of sample means: 0.314214
Theoretical standard error: 0.316228
```

b. Now repeat the experiment with $T = 20$ observations and compare the histograms (also compute the standard deviation of each histogram).

```r
# Part B: Generate data for T=20 and compare
T2 <- 20
sample_means_T20 <- replicate(n_experiments, {
  data <- rnorm(T2, mean = true_mean, sd = true_sd)
  mean(data)
})

# Create histogram for T=20
hist_data_T20 <- data.frame(sample_means = sample_means_T20)

ggplot(hist_data_T20, aes(x = sample_means)) +
  geom_histogram(bins = 30, fill = "lightgreen", color = "black", alpha = 0.7) +
  geom_vline(aes(xintercept = true_mean), color = "red", linewidth = 1.5, linetype = "dashed") +
  geom_vline(aes(xintercept = mean(sample_means)), color = "blue", linewidth = 1.5) +
  labs(title = "Distribution of Sample Means (T = 20)",
       x = "Sample Mean",
       y = "Frequency",
       subtitle = paste("Red line: True mean =", true_mean,
                        "| Blue line: Empirical mean =", round(mean(sample_means_T20), 4))) +
  theme_minimal()
```




```
# Combined comparison plot
combined_data <- rbind(
  data.frame(sample_means = sample_means_T10, T = "T = 10"),
  data.frame(sample_means = sample_means_T20, T = "T = 20")
)

ggplot(combined_data, aes(x = sample_means, fill = T)) +
  geom_histogram(bins = 30, alpha = 0.6, position = "identity") +
  geom_vline(aes(xintercept = true_mean), color = "red", linewidth = 1.5, linetype = "dashed") +
  scale_fill_manual(values = c("lightblue", "lightgreen")) +
  labs(title = "Comparison of Sample Mean Distributions",
       x = "Sample Mean",
       y = "Frequency",
       subtitle = "Red line shows true mean = 0.01") +
  theme_minimal() +
  facet_wrap(~T, scales = "free_y")
```




```
# Print comparison statistics
cat("Part B Results - Comparison:\n")
cat("T = 10: Mean =", round(mean(sample_means_T10), 6),
    ", SD =", round(sd(sample_means_T10), 6), "\n")
cat("T = 20: Mean =", round(mean(sample_means_T20), 6),
    ", SD =", round(sd(sample_means_T20), 6), "\n")
cat("Theoretical SD for T=10:", round(true_sd/sqrt(T1), 6), "\n")
cat("Theoretical SD for T=20:", round(true_sd/sqrt(T2), 6), "\n\n")
```

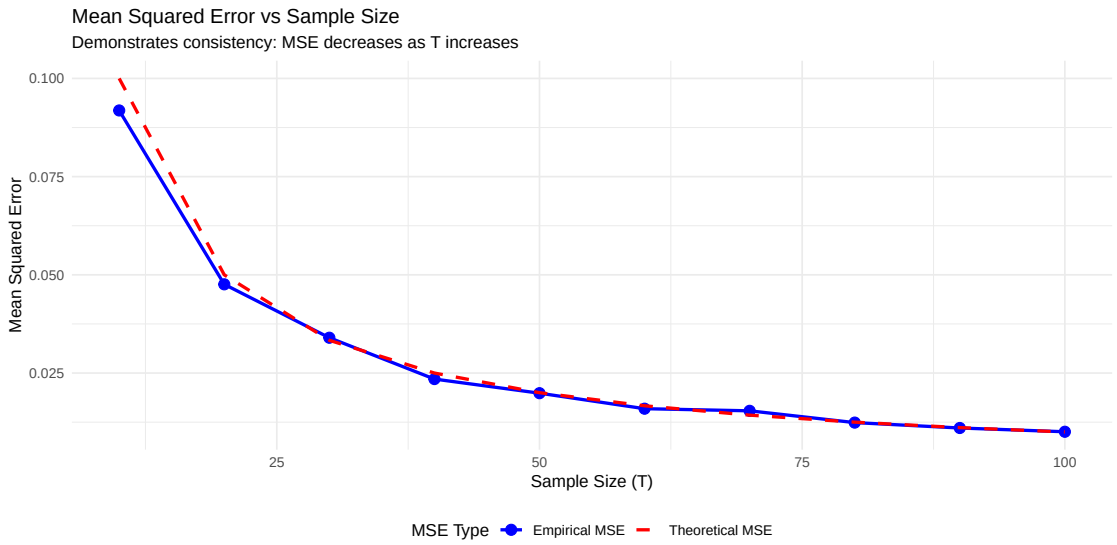
```
Part B Results - Comparison:
T = 10: Mean = 0.007628 , SD = 0.314214
T = 20: Mean = 0.001897 , SD = 0.22328
Theoretical SD for T=10: 0.316228
Theoretical SD for T=20: 0.223607
```

c. Finally, repeat the experiment multiple times, for different numbers of observations $T = 10, 20, \ldots, 100$, and plot the mean squared error of the estimation as a function of T.

```r
# Part C: MSE as function of T
T_values <- seq(10, 100, by = 10)
mse_results <- data.frame(T = T_values, MSE = numeric(length(T_values)))

for (i in seq_along(T_values)) {
  T_current <- T_values[i]

  sample_means_current <- replicate(n_experiments, {
    data <- rnorm(T_current, mean = true_mean, sd = true_sd)
    mean(data)
  })

  # Calculate MSE
  mse_results$MSE[i] <- mean((sample_means_current - true_mean)^2)

  cat("T =", T_current, ", MSE =", round(mse_results$MSE[i], 6), "\n")
}
```

```
T = 10 , MSE = 0.091842
T = 20 , MSE = 0.047585
T = 30 , MSE = 0.03399
T = 40 , MSE = 0.023468
T = 50 , MSE = 0.019875
T = 60 , MSE = 0.015924
T = 70 , MSE = 0.015382
T = 80 , MSE = 0.012385
T = 90 , MSE = 0.011011
T = 100 , MSE = 0.010055
```

```r
# Theoretical MSE (which equals variance of sample mean)
mse_results$Theoretical_MSE <- true_variance / mse_results$T

# Plot MSE vs T
ggplot(mse_results, aes(x = T)) +
  geom_point(aes(y = MSE, color = "Empirical MSE"), size = 3) +
  geom_line(aes(y = MSE, color = "Empirical MSE"), linewidth = 1) +
  geom_line(aes(y = Theoretical_MSE, color = "Theoretical MSE"),
            linewidth = 1, linetype = "dashed") +
  scale_color_manual(values = c("Empirical MSE" = "blue", "Theoretical MSE" = "red")) +
  labs(title = "Mean Squared Error vs Sample Size",
       x = "Sample Size (T)",
       y = "Mean Squared Error",
       color = "MSE Type",
       subtitle = "Demonstrates consistency: MSE decreases as T increases") +
  theme_minimal() +
  theme(legend.position = "bottom")
```




```
# Summary table
cat("\nPart C Results - MSE Summary:\n")
print(mse_results)
```

```
Part C Results - MSE Summary:
     T        MSE Theoretical_MSE
1   10 0.09184212      0.10000000
2   20 0.04758472      0.05000000
3   30 0.03399019      0.03333333
4   40 0.02346757      0.02500000
5   50 0.01987454      0.02000000
6   60 0.01592392      0.01666667
7   70 0.01538181      0.01428571
8   80 0.01238546      0.01250000
9   90 0.01101063      0.01111111
10 100 0.01005520      0.01000000
```

Exercise 3.2: Bias of sample covariance matrix

Suppose we have T i.i.d. N-dimensional observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ distributed as $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

a. Derive the following expected value based on the true $\boldsymbol{\mu}$:

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}}\right].$$

b. Derive the following expected value based now on the sample mean $\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t$:

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^{\mathsf{T}}\right].$$

c. Discuss the appropriate normalization factor, $1/(T-1)$ or $1/T$, to be used in the expression of the sample covariance matrix.

Solution

a. Expected value with true mean:

Using the linearity of expectation:

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}}\right] = \sum_{t=1}^{T}\mathbb{E}[(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}}] = \sum_{t=1}^{T}\boldsymbol{\Sigma} = T \times \boldsymbol{\Sigma}.$$

Thus, to get an unbiased estimator of the covariance matrix, one should include the scaling factor $1/T$.

b. Expected value with sample mean:

Once can easily verify the identity:

$$\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^{\mathsf{T}} = \sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}} - T(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})^{\mathsf{T}}.$$

Taking expectations leads to

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^{\mathsf{T}}\right] = \mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}}\right] - T \times \mathbb{E}[(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})^{\mathsf{T}}]$$

From part (a), we know the first term equals $T \times \boldsymbol{\Sigma}$. For the second term, since

$$\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t$$

and

$$\mathbb{E}[\hat{\boldsymbol{\mu}}] = \boldsymbol{\mu},$$

we have that

$$\mathbb{E}[(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})^\mathsf{T}] = \mathrm{Cov}(\hat{\boldsymbol{\mu}}) = \mathrm{Cov}\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t\right) = \frac{1}{T^2}\sum_{t=1}^{T}\mathrm{Cov}(\boldsymbol{x}_t) = \frac{T\boldsymbol{\Sigma}}{T^2} = \frac{\boldsymbol{\Sigma}}{T}.$$

Therefore:

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}\right] = T\cdot\boldsymbol{\Sigma} - T\cdot\frac{\boldsymbol{\Sigma}}{T} = T\cdot\boldsymbol{\Sigma} - \boldsymbol{\Sigma} = (T-1)\cdot\boldsymbol{\Sigma}.$$

Thus, to get an unbiased estimator of the covariance matrix, one should include the scaling factor $1/(T-1)$.

c. Appropriate normalization factor

In the case of known mean vector $\boldsymbol{\mu}$, the unbiased estimator of the covariance matrix is

$$\hat{\boldsymbol{\Sigma}}_{\text{unbiased}} = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T},$$

whereas in the case of a mean vector estimated via the sample mean $\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t$, it is

$$\hat{\boldsymbol{\Sigma}}_{\text{unbiased}} = \frac{1}{T-1}\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}.$$

The reason why in the case of the sample mean the normalization factor is $1/(T-1)$ instead of $1/T$ is that the sample mean $\hat{\boldsymbol{\mu}}$ is estimated from the same data used to compute the covariance matrix. This introduces a dependency that reduces the apparent variability, since the sample mean is the value that minimizes the sum of squared deviations. We "lose" one degree of freedom by estimating the mean, hence the factor $T-1$ instead of T.
The factor $1/(T-1)$ is sometimes referred to as Bessel's correction.
Of course, the difference becomes negligible as $T \to \infty$.

Exercise 3.3: Location estimators

Consider a two-dimensional ($N = 2$) Gaussian-distributed i.i.d. time series with zero mean and identity covariance matrix, $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, $t = 1, \ldots, T$.

a. Generate data for $T = 20$ and estimate the mean vector $\boldsymbol{\mu}$ via the sample mean, the median, and the spatial median. Visualize the results in a scatter plot.
b. Repeat the experiment multiple times, for different numbers of observations $T =$

$10, 20, \ldots, 100$, and plot the mean squared error as a function of T.

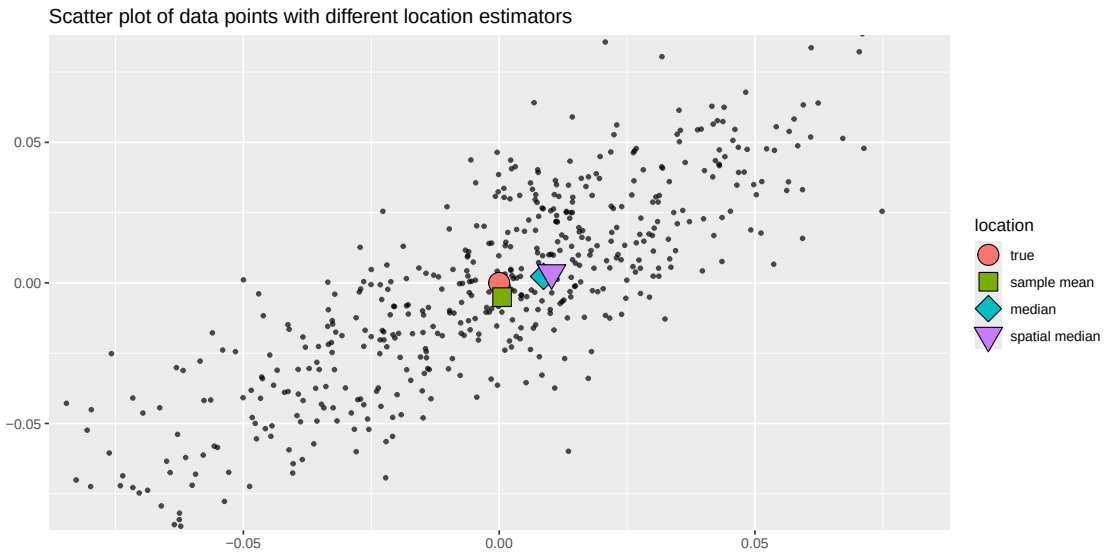
Solution

First generate synthetic data:

```
library(mvtnorm)

mu_true = c(0, 0)
Sigma_true <- cbind(c(0.0012, 0.0011), c(0.0011, 0.0014))

# Generate synthetic Gaussian data
set.seed(42)
X <- rmvnorm(n = 1000, mean  = mu_true, sigma = Sigma_true)
```

a. Illustration of different location estimators in 2D:

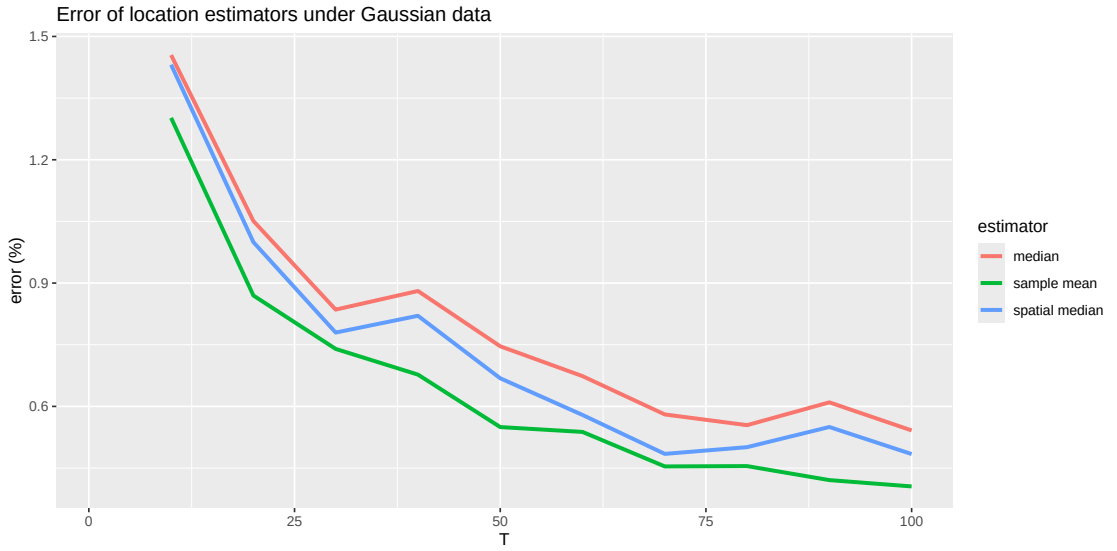
```
library(ellipse)
library(ICSNP)
library(ggplot2)

T <- 20

# estimators
mu_mean <- colMeans(X[1:T, ])
mu_median <- apply(X[1:T, ], 2, median)
mu_spatial_median <- ICSNP::spatial.median(X[1:T, ])
df <- data.frame("location" = c("true", "sample mean", "median", "spatial median"),
                 x = c(mu_true[1], mu_mean[1], mu_median[1], mu_spatial_median[1]),
                 y = c(mu_true[2], mu_mean[2], mu_median[2], mu_spatial_median[2]))
df$location <- factor(df$location, levels = unique(df$location))

# scatter plot
colnames(Sigma_true) <- rownames(Sigma_true) <- NULL
ggplot(data.frame(x = X[1:500, 1], y = X[1:500, 2]), aes(x, y)) +
  geom_point(alpha = 0.7, size = 1) +
  geom_point(data = df, aes(x, y, fill = location, shape = location), size = 6) +
  scale_shape_manual(values = c(21, 22, 23, 25)) +
  coord_cartesian(xlim = c(-0.08, 0.08), ylim = c(-0.08, 0.08)) +
  labs(title = "Scatter plot of data points with different location estimators",
       x = NULL, y = NULL)
```



b. Estimation error of location estimators versus number of observations:

```r
# main loop
df <- data.frame()
T_sweep <- seq(from = 10, by = 10, to = 100)
for(T in T_sweep) {
  for (i in 1:100) {
    X_ <- X[sample(nrow(X), T), ]

    # sample mean
    mu <- colMeans(X_)
    df <- rbind(df, data.frame("T"            = T,
                               "distribution" = "Gaussian",
                               "method"       = "sample mean",
                               "error mu"     = norm(mu - mu_true, "2"),
                               check.names    = FALSE))
    # median
    mu <- apply(X_, 2, median)
    df <- rbind(df, data.frame("T"            = T,
                               "distribution" = "Gaussian",
                               "method"       = "median",
                               "error mu"     = norm(mu - mu_true, "2"),
                               check.names    = FALSE))
    # spatial median
    mu <- ICSNP::spatial.median(X_)
    df <- rbind(df, data.frame("T"            = T,
                               "distribution" = "Gaussian",
                               "method"       = "spatial median",
                               "error mu"     = norm(mu - mu_true, "2"),
                               check.names    = FALSE))
  }
}
```

```r
df |>
  group_by(method, T, distribution) |>
  summarize("error mu" = 100*mean(`error mu`)) |>
  ungroup() |>
  ggplot(aes(x = T, y = `error mu`, color = method,)) +
  geom_line(linewidth = 1.2) +
  labs(color = "estimator") +
  coord_cartesian(xlim = c(1, 100)) +
  labs(title = "Error of location estimators under Gaussian data", x = "T", y = "error (%)")
```



Exercise 3.4: Location estimators with outliers

Consider a two-dimensional ($N = 2$) Gaussian-distributed i.i.d. time series with zero mean and identity covariance matrix, $\boldsymbol{x}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $t = 1, \ldots, T$.

a. Generate data for $T = 20$ and estimate the mean vector $\boldsymbol{\mu}$ via the sample mean, the median, and the spatial median. Visualize the results in a scatter plot. Repeat the experiment multiple times and compute the mean squared error of the estimators.
b. Then, add some small percentage of outliers in the observations, for example, distributed as $\boldsymbol{x}_t \sim \mathcal{N}(0.1 \times \mathbf{1}, \boldsymbol{I})$, and compute again the mean squared error of the estimators.
c. Finally, repeat the experiment multiple times and plot the estimation error as a function of the percentage of outliers. Observe the robustness of the three estimators against outliers and discuss.

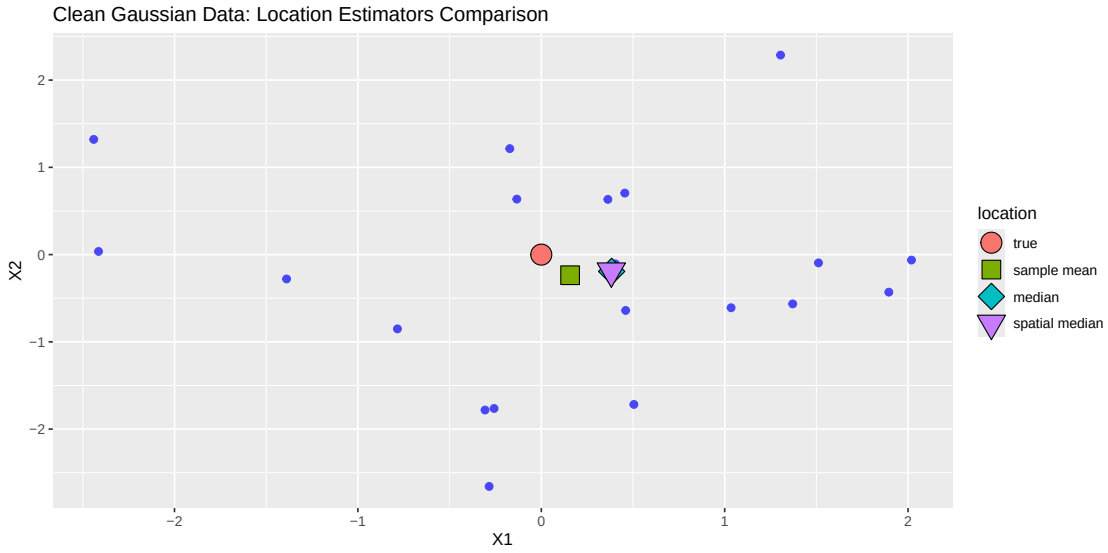
Solution

First generate synthetic data:

```r
library(mvtnorm)
library(ellipse)
library(ICSNP)
library(ggplot2)
library(dplyr)

# True parameters
mu_true <- c(0, 0)
Sigma_true <- diag(2)  # Identity covariance matrix

set.seed(42)
```

a. Clean data analysis. First, let's establish baseline performance with clean Gaussian data:

```r
# Visualization for one example
T <- 20

X_example <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)
mu_mean_ex <- colMeans(X_example)
mu_median_ex <- apply(X_example, 2, median)
mu_spatial_median_ex <- ICSNP::spatial.median(X_example)

df_estimators <- data.frame(
  location = c("true", "sample mean", "median", "spatial median"),
  x = c(mu_true[1], mu_mean_ex[1], mu_median_ex[1], mu_spatial_median_ex[1]),
  y = c(mu_true[2], mu_mean_ex[2], mu_median_ex[2], mu_spatial_median_ex[2])
)
df_estimators$location <- factor(df_estimators$location, levels = unique(df_estimators$location))

ggplot(data.frame(x = X_example[, 1], y = X_example[, 2]), aes(x, y)) +
  geom_point(alpha = 0.7, size = 2, color = "blue") +
  geom_point(data = df_estimators, aes(x, y, fill = location, shape = location), size = 6) +
  scale_shape_manual(values = c(21, 22, 23, 25)) +
  labs(title = "Clean Gaussian Data: Location Estimators Comparison",
       x = "X1", y = "X2")
```




```
T <- 20
n_experiments <- 100

# Generate clean data and compute MSE for each estimator
mse_clean <- data.frame()

for(i in 1:n_experiments) {
  # Generate clean Gaussian data
  X_clean <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

  # Compute estimators
  mu_mean <- colMeans(X_clean)
  mu_median <- apply(X_clean, 2, median)
  mu_spatial_median <- ICSNP::spatial.median(X_clean)

  # Compute squared errors
  mse_clean <- rbind(mse_clean, data.frame(
    experiment = i,
    outlier_pct = 0,
    method = c("sample mean", "median", "spatial median"),
    mse = c(norm(mu_mean - mu_true, "2")^2,
            norm(mu_median - mu_true, "2")^2,
            norm(mu_spatial_median - mu_true, "2")^2)
  ))
}
```

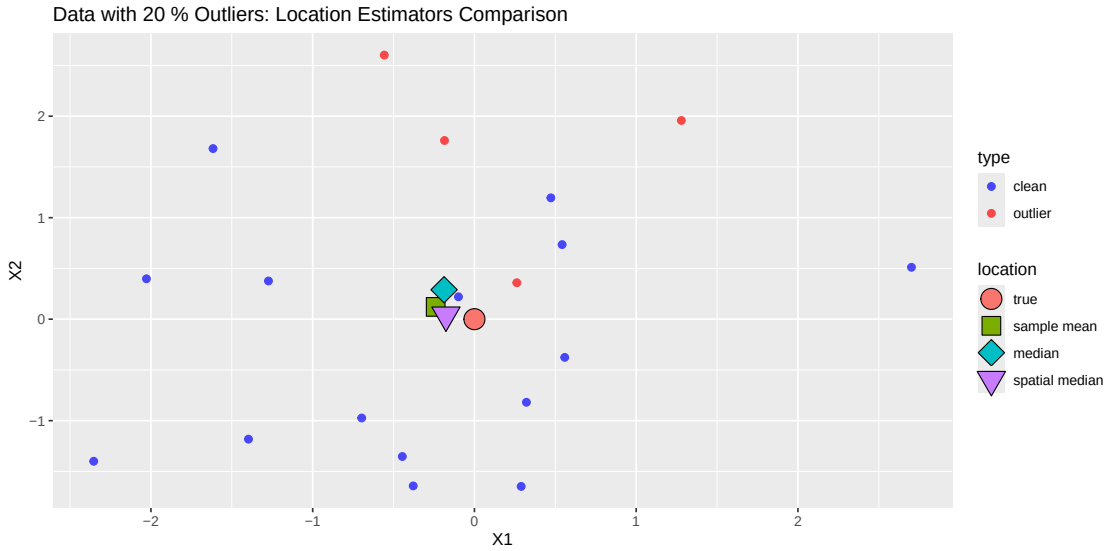
```r
# Print average MSE for clean data
cat("Quantiles of MSE (T =", T, "):\n")
mse_clean_summary <- mse_clean %>%
  group_by(method) %>%
  summarize(avg_mse = mean(mse),
            std_mse = sd(mse),
            .groups = 'drop')

print(mse_clean_summary)
```

```
Quantiles of MSE (T = 20 ):
# A tibble: 3 x 3
  method         avg_mse std_mse
  <chr>            <dbl>   <dbl>
1 median           0.137  0.139
2 sample mean      0.113  0.0989
3 spatial median   0.116  0.112
```

b. Adding outliers. Now let's introduce outliers and compare the robustness:

```r
# Function to generate data with outliers
generate_data_with_outliers <- function(T, outlier_pct) {
  n_outliers <- round(T * outlier_pct / 100)
  n_clean <- T - n_outliers

  # Generate clean data
  X <- rmvnorm(n = n_clean, mean = mu_true, sigma = Sigma_true)

  # Generate outliers: shifted mean
  if(n_outliers > 0) {
    outlier_mean <- 1.0 * c(1, 1)  # larger than specified in the exercise
    X_outliers <- rmvnorm(n = n_outliers, mean = outlier_mean, sigma = Sigma_true)
    X <- rbind(X, X_outliers)
  }
  return(X)
}
```

```r
# Test with 10% outliers
outlier_pct <- 20
X_with_outliers <- generate_data_with_outliers(T, outlier_pct)

# Compute estimators
mu_mean_out <- colMeans(X_with_outliers)
mu_median_out <- apply(X_with_outliers, 2, median)
mu_spatial_median_out <- ICSNP::spatial.median(X_with_outliers)

# Visualization with outliers
n_outliers <- round(T * outlier_pct / 100)
df_data <- data.frame(
  x = X_with_outliers[, 1],
  y = X_with_outliers[, 2],
  type = c(rep("clean", T - n_outliers), rep("outlier", n_outliers))
)

df_estimators_out <- data.frame(
  location = c("true", "sample mean", "median", "spatial median"),
  x = c(mu_true[1], mu_mean_out[1], mu_median_out[1], mu_spatial_median_out[1]),
  y = c(mu_true[2], mu_mean_out[2], mu_median_out[2], mu_spatial_median_out[2])
)
df_estimators_out$location <- factor(df_estimators_out$location, levels = unique(df_estimators_out$

ggplot(df_data, aes(x, y, color = type)) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("clean" = "blue", "outlier" = "red")) +
  geom_point(data = df_estimators_out, aes(x, y, fill = location, shape = location),
             size = 6, color = "black") +
  scale_shape_manual(values = c(21, 22, 23, 25)) +
  labs(title = paste("Data with", outlier_pct, "% Outliers: Location Estimators Comparison"),
       x = "X1", y = "X2")
```




```
# Compute MSE with outliers over multiple experiments
T <- 20
outlier_pct <- 20
n_experiments <- 100
mse_with_outliers <- data.frame()

for(i in 1:n_experiments) {
  # Generate data with outliers
  X_with_outliers <- generate_data_with_outliers(T, outlier_pct)

  # Compute estimators
  mu_mean <- colMeans(X_with_outliers)
  mu_median <- apply(X_with_outliers, 2, median)
  mu_spatial_median <- ICSNP::spatial.median(X_with_outliers)

  # Compute squared errors
  mse_with_outliers <- rbind(mse_with_outliers, data.frame(
    experiment = i,
    outlier_pct = outlier_pct,
    method = c("sample mean", "median", "spatial median"),
    mse = c(norm(mu_mean - mu_true, "2")^2,
            norm(mu_median - mu_true, "2")^2,
            norm(mu_spatial_median - mu_true, "2")^2)
  ))
}
```

```r
library(tidyr)

# Print average MSE with outliers
cat("Average MSE with", outlier_pct, "% outliers (T =", T, "):\n")
mse_outliers_summary <- mse_with_outliers %>%
  group_by(method) %>%
  summarize(avg_mse = mean(mse),
            std_mse = sd(mse),
            .groups = 'drop')

print(mse_outliers_summary)

# Compare with clean data results
cat("\nComparison: Clean vs Outliers\n")
comparison <- rbind(
  mse_clean %>% group_by(method) %>%
    summarize(avg_mse = mean(mse), condition = "clean", .groups = 'drop'),
  mse_with_outliers %>% group_by(method) %>%
    summarize(avg_mse = mean(mse), condition = "outliers", .groups = 'drop')
)

comparison_wide <- comparison %>%
  pivot_wider(names_from = condition, values_from = avg_mse) %>%
  mutate(ratio = outliers / clean)

print(comparison_wide)
```

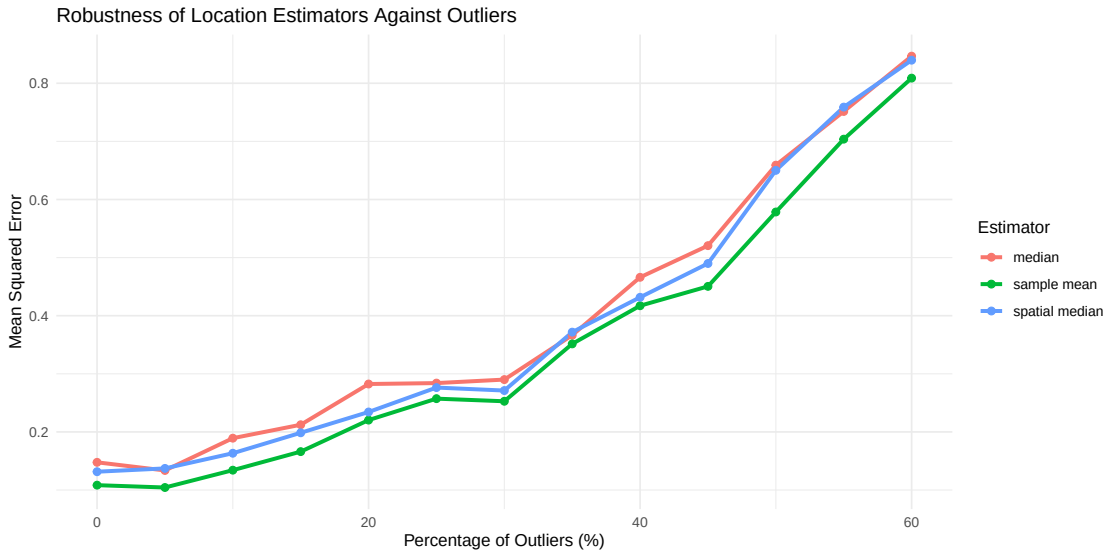
```
Average MSE with 20 % outliers (T = 20 ):
# A tibble: 3 x 3
  method         avg_mse std_mse
  <chr>            <dbl>   <dbl>
1 median           0.209   0.202
2 sample mean      0.194   0.173
3 spatial median   0.201   0.170

Comparison: Clean vs Outliers
# A tibble: 3 x 4
  method         clean outliers ratio
  <chr>          <dbl>    <dbl> <dbl>
1 median         0.137    0.209  1.52
2 sample mean    0.113    0.194  1.71
3 spatial median 0.116    0.201  1.73
```

c. Robustness Analysis. Finally, let's analyze how estimation error varies with the percentage of outliers:

```r
# Sweep over different outlier percentages
outlier_percentages <- seq(0, 60, by = 5)
n_experiments <- 100

results <- data.frame()

for(outlier_pct in outlier_percentages) {
  cat("Processing outlier percentage:", outlier_pct, "%\n")

  for(i in 1:n_experiments) {
    # Generate data with outliers
    X <- generate_data_with_outliers(T, outlier_pct)

    # Compute estimators
    mu_mean <- colMeans(X)
    mu_median <- apply(X, 2, median)
    mu_spatial_median <- ICSNP::spatial.median(X)

    # Compute squared errors
    results <- rbind(results, data.frame(
      experiment = i,
      outlier_pct = outlier_pct,
      method = c("sample mean", "median", "spatial median"),
      mse = c(norm(mu_mean - mu_true, "2")^2,
              norm(mu_median - mu_true, "2")^2,
              norm(mu_spatial_median - mu_true, "2")^2)
    ))
  }
}
```

```
Processing outlier percentage: 0 %
Processing outlier percentage: 5 %
Processing outlier percentage: 10 %
Processing outlier percentage: 15 %
Processing outlier percentage: 20 %
Processing outlier percentage: 25 %
Processing outlier percentage: 30 %
Processing outlier percentage: 35 %
Processing outlier percentage: 40 %
Processing outlier percentage: 45 %
Processing outlier percentage: 50 %
Processing outlier percentage: 55 %
Processing outlier percentage: 60 %
```

```r
# Plot results
results %>%
  group_by(method, outlier_pct) %>%
  summarize(mean_mse = mean(mse), .groups = 'drop') %>%
  ggplot(aes(x = outlier_pct, y = mean_mse, color = method)) +
  geom_line(linewidth = 1.2) +
  geom_point(size = 2) +
  labs(title = "Robustness of Location Estimators Against Outliers",
       x = "Percentage of Outliers (%)",
       y = "Mean Squared Error",
       color = "Estimator") +
  theme_minimal()
```



Exercise 3.5: Derivation of sample mean as location estimator

Given the observations $\boldsymbol{x}_t$, $t = 1, \ldots, T$, the sample mean can be derived as the solution to the following optimization problem:

$$\underset{\boldsymbol{\mu}}{\text{minimize}} \quad \sum_{t=1}^{T} \|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2^2.$$

a. Is this problem convex? What class of optimization problem is it?
b. Derive the solution in closed form by setting the gradient with respect to $\boldsymbol{\mu}$ to zero.

Solution

a. Convexity:

This optimization problem is indeed convex, because the objective function is the sum of squared Euclidean distances from the observations $\boldsymbol{x}_t$ to the optimization variable $\boldsymbol{\mu}$, so it is actually a convex quadratic problem. In fact, it is a least squares problem.

b. Closed-form solution:

We rewrite the quadratic objective function as

$$\begin{aligned} f(\boldsymbol{\mu}) &= \sum_{t=1}^{T} \|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2^2 \\ &= \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T} (\boldsymbol{x}_t - \boldsymbol{\mu}) \\ &= \sum_{t=1}^{T} \left(\boldsymbol{x}_t^\mathsf{T} \boldsymbol{x}_t - 2\boldsymbol{x}_t^\mathsf{T} \boldsymbol{\mu} + \boldsymbol{\mu}^\mathsf{T} \boldsymbol{\mu} \right) \end{aligned}$$

with gradient with respect to $\boldsymbol{\mu}$:

$$\nabla_{\boldsymbol{\mu}} f(\boldsymbol{\mu}) = \sum_{t=1}^{T} (0 - 2\boldsymbol{x}_t + 2\boldsymbol{\mu}) = \sum_{t=1}^{T} 2(\boldsymbol{\mu} - \boldsymbol{x}_t) = 2T\boldsymbol{\mu} - 2\sum_{t=1}^{T} \boldsymbol{x}_t.$$

Finally, setting the gradient to zero leads to the desired result:

$$\boldsymbol{\mu} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{x}_t,$$

which is precisely the sample mean, confirming that the sample mean is the optimal solution to the least squares location estimation problem. The result shows that the sample mean minimizes the sum of squared deviations from all observations, making it the natural choice for estimating the central location of a dataset.

Exercise 3.6: Computation of spatial median as location estimator

Given the observations $\boldsymbol{x}_t$, $t = 1, \ldots, T$, the spatial median can be derived as the solution to the following optimization problem:

$$\underset{\boldsymbol{\mu}}{\text{minimize}} \quad \sum_{t=1}^{T} \|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2.$$

a. Is this problem convex? What class of optimization problem is it?
b. Can a closed-form solution be obtained as in the case of the sample mean?
c. Develop an iterative algorithm to compute the spatial median by solving a sequence of weighted sample means. Hint: find a majorizer of the ℓ_2-norm in the form of a squared ℓ_2-norm and then employ the majorization–minimization framework.

Solution

a. Convexity:

The spatial median optimization problem is convex. This is because each term $\|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2$ in the objective function is a norm, which is inherently convex, and the sum of convex functions remains convex.
Unlike the sample mean problem, which is a quadratic optimization problem, the spatial median involves minimizing a sum of ℓ_2-norms rather than squared ℓ_2-norms. This makes the problem a second-order cone problem (SOCP).

b. Closed-form solution:

No, a closed-form solution cannot be obtained for the spatial median problem. This is a fundamental difference from the sample mean case. While the sample mean has a simple closed-form expression due to the differentiability of the squared norm, the spatial median problem involves non-differentiable ℓ_2-norms at points where $\boldsymbol{\mu} = \boldsymbol{x}_t$, making analytical solution impossible in general.

c. Iterative Algorithm via Majorization-Minimization:

To develop an iterative algorithm, we employ the majorization-minimization (MM) framework by finding a suitable majorizer for the ℓ_2-norm.
Majorizer Construction: For the ℓ_2-norm $\|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2$, we can construct a majorizer using the weighted squared norm. At iteration k with current estimate $\boldsymbol{\mu}_k$, the majorizer is

$$\|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2 \leq \frac{1}{2w_t^{(k)}} \|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2^2 + \text{constant},$$

where the weight is defined as

$$w_t^{(k)} = \frac{1}{\|\boldsymbol{x}_t - \boldsymbol{\mu}_k\|_2}.$$

Algorithm Development: The majorized objective function becomes:

$$\sum_{t=1}^{T} \frac{1}{2w_t^{(k)}} \|\boldsymbol{x}_t - \boldsymbol{\mu}\|_2^2$$

Minimizing this weighted sum of squared norms gives us the update rule for a weighted sample mean:

$$\boldsymbol{\mu}_{k+1} = \frac{\sum_{t=1}^{T} w_t^{(k)} \boldsymbol{x}_t}{\sum_{t=1}^{T} w_t^{(k)}}$$

Complete Algorithm: The iterative algorithm for computing the spatial median is

1. **Initialize** $\boldsymbol{\mu}_0$ (e.g., using the sample mean)
2. **Repeat until convergence:**
   - Compute weights:

$$w_t^{(k)} = \frac{1}{\|\boldsymbol{x}_t - \boldsymbol{\mu}_k\|_2}, \quad \text{for all } t = 1, \ldots, T$$

   - Update:

$$\boldsymbol{\mu}_{k+1} = \frac{\sum_{t=1}^{T} w_t^{(k)} \boldsymbol{x}_t}{\sum_{t=1}^{T} w_t^{(k)}}$$

This algorithm is essentially **Weiszfeld's algorithm**, a well-known iterative method for computing the spatial median. The key insight is that at each iteration, we solve a weighted least squares problem where observations closer to the current estimate receive higher weights, naturally leading to a robust location estimator that is less sensitive to outliers compared to the sample mean. The algorithm converges to the spatial median, which minimizes the sum of Euclidean distances and provides a more robust measure of central tendency than the sample mean, particularly in the presence of outliers.

Exercise 3.7: ML estimation of covariance matrix

Consider an N-dimensional i.i.d. time series with zero mean and identity covariance matrix, $\boldsymbol{x}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}),\ t = 1, \ldots, T$.

a. Generate Gaussian data for $N = 10$ and $T = 50$ and estimate the covariance matrix $\boldsymbol{\Sigma}$ via the Gaussian ML estimator and the heavy-tailed ML estimator. Run the experiment multiple times and compute the mean squared error of the estimators.

b. Now repeat the whole experiment but generating instead heavy-tailed data (e.g., following a t distribution) with the same mean and covariance matrix. Observe the robustness of the two estimators against heavy tails and discuss.

Solution

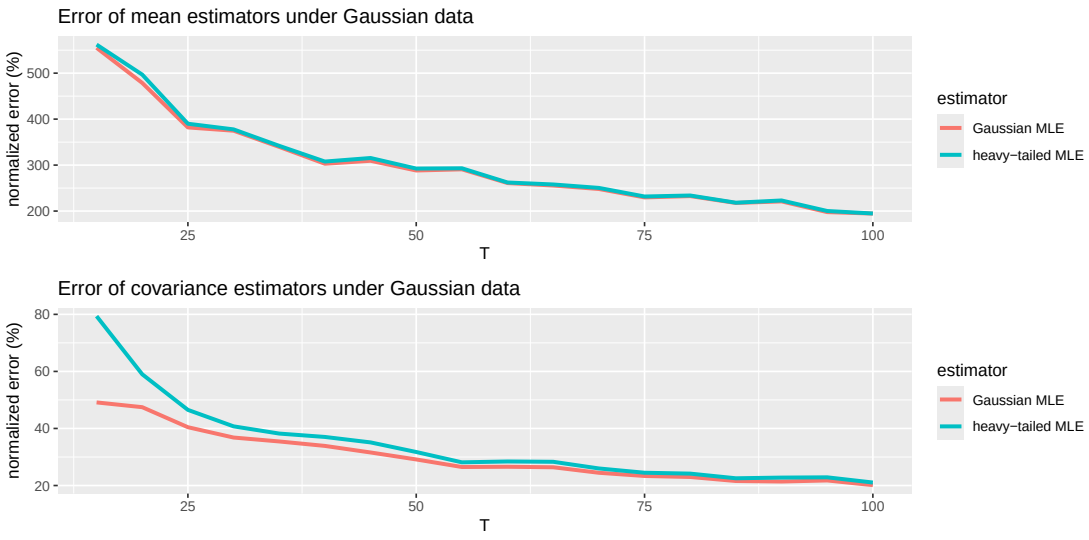
We will instead choose realistic parameters:

```
library(xts)
library(pob)
set.seed(123)

N <- 10

# get realistic mu and Sigma
X <- diff(log(SP500_2015to2020$stocks))[-1, 1:100]
X <- X[, sample(ncol(X), N)]
nu <- 4
mu_true <- colMeans(X)
Sigma_true <- cov(X)
scatter_true <- (nu-2)/nu * Sigma_true  # cov=nu/(nu-2)*scatter
```

a. Sample estimators under Gaussian data:

```r
library(mvtnorm)
library(fitHeavyTail)
library(ICSNP)

# generate synthetic data
set.seed(42)
T_max <- 100
X_Gaussian   <- rmvnorm(n = 10*T_max, mean  = mu_true, sigma = Sigma_true)

# main loop
df <- data.frame()
T_sweep <- seq(from = 15, by = 5, to = T_max)
for(T in T_sweep) {
  for (i in 1:100) {
    X_ <- X_Gaussian[sample(nrow(X_Gaussian), T), ]

    # Gaussian MLE
    mu    <- colMeans(X_)
    Sigma <- (T-1)/T * cov(X_)
    df <- rbind(df, data.frame(
      "T"            = T,
      "method"       = "Gaussian MLE",
      "error mu"     = norm(mu - mu_true, "2")/norm(mu_true, "2"),
      "MAE mu"       = sum(abs(mu - mu_true))/sum(abs(mu_true)),
      "error Sigma"  = norm(Sigma - Sigma_true, "F")/norm(Sigma_true, "F"),
      check.names    = FALSE))

    # heavy-tailed MLE
    fitted <- fit_mvt(X_, nu = "iterative", nu_iterative_method = "POP")
    df <- rbind(df, data.frame(
      "T"            = T,
      "method"       = "heavy-tailed MLE",
      "error mu"     = norm(fitted$mu - mu_true, "2")/norm(mu_true, "2"),
      "MAE mu"       = sum(abs(fitted$mu - mu_true))/sum(abs(mu_true)),
      "error Sigma"  = norm(fitted$cov - Sigma_true, "F")/norm(Sigma_true, "F"),
      check.names    = FALSE))
  }
}
df$method <- factor(df$method, levels = unique(df$method))

df <- df |>
  group_by(method, T) |>
  summarize("error mu"    = 100*mean(`error mu`),
            "RMSE mu"     = 100*sqrt(mean(`error mu`)),
            "MAE mu"      = 100*mean(`MAE mu`),
            "error Sigma" = 100*mean(`error Sigma`)) |>
  ungroup()
```

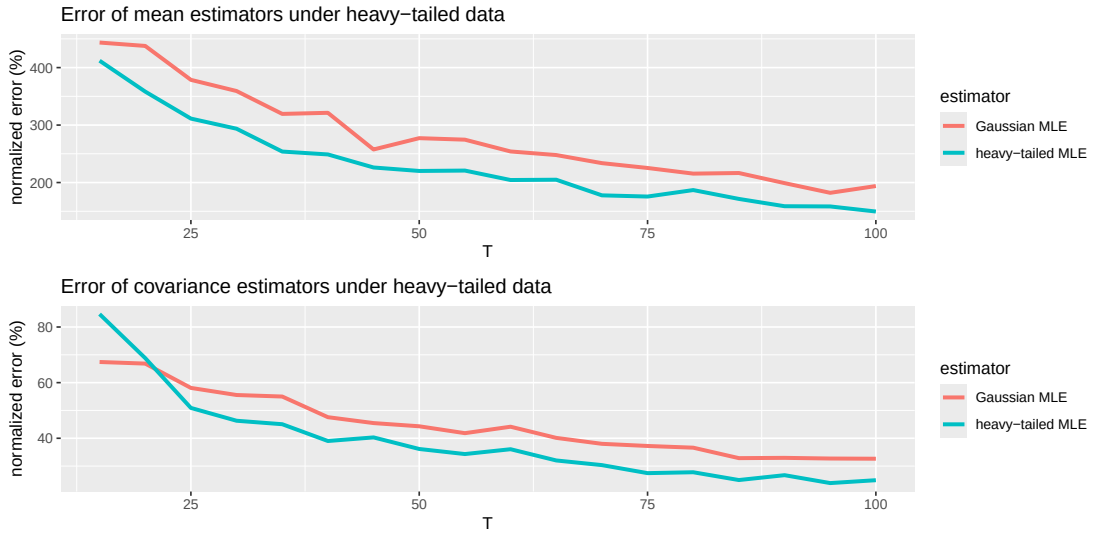
```r
library(ggplot2)
library(patchwork)              # for combining plots

# plot
p_error_mu <- df |>
  ggplot(aes(x = T, y = `error mu`, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_color_discrete(name = "estimator",
                       labels = c("Gaussian MLE", "heavy-tailed MLE")) +
  labs(title = "Error of mean estimators under Gaussian data",
       x = "T", y = "normalized error (%)")

p_error_Sigma <- df |>
  ggplot(aes(x = T, y = `error Sigma`, color = method)) +
  geom_line(linewidth = 1.2) +
  labs(color = "estimator") +
  labs(title = "Error of covariance estimators under Gaussian data",
       x = "T", y = "normalized error (%)")

p_error_mu / p_error_Sigma
```



b. Sample estimators under heavy-tailed data:

```r
library(mvtnorm)
library(fitHeavyTail)
library(ICSNP)

# generate synthetic data
set.seed(42)
T_max <- 100
X_heavy_tail <- rmvt(n = 10*T_max,    delta = mu_true, sigma = scatter_true, df = nu)

# main loop
df <- data.frame()
T_sweep <- seq(from = 15, by = 5, to = T_max)
for(T in T_sweep) {
  for (i in 1:100) {
    X_ <- X_heavy_tail[sample(nrow(X_heavy_tail), T), ]

    # Gaussian MLE
    mu    <- colMeans(X_)
    Sigma <- (T-1)/T * cov(X_)
    df <- rbind(df, data.frame(
      "T"            = T,
      "method"       = "Gaussian MLE",
      "error mu"     = norm(mu - mu_true, "2")/norm(mu_true, "2"),
      "MAE mu"       = sum(abs(mu - mu_true))/sum(abs(mu_true)),
      "error Sigma"  = norm(Sigma - Sigma_true, "F")/norm(Sigma_true, "F"),
      check.names    = FALSE))

    # heavy-tailed MLE
    fitted <- fit_mvt(X_, nu = "iterative", nu_iterative_method = "POP")
    df <- rbind(df, data.frame(
      "T"            = T,
      "method"       = "heavy-tailed MLE",
      "error mu"     = norm(fitted$mu - mu_true, "2")/norm(mu_true, "2"),
      "MAE mu"       = sum(abs(fitted$mu - mu_true))/sum(abs(mu_true)),
      "error Sigma"  = norm(fitted$cov - Sigma_true, "F")/norm(Sigma_true, "F"),
      check.names    = FALSE))
  }
}
df$method <- factor(df$method, levels = unique(df$method))

df <- df |>
  group_by(method, T) |>
  summarize("error mu"    = 100*mean(`error mu`),
            "RMSE mu"     = 100*sqrt(mean(`error mu`)),
            "MAE mu"      = 100*mean(`MAE mu`),
            "error Sigma" = 100*mean(`error Sigma`)) |>
  ungroup()
```

```r
# plot
p_error_mu <- df |>
  ggplot(aes(x = T, y = `error mu`, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_color_discrete(name = "estimator", labels = c("Gaussian MLE", "heavy-tailed MLE")) +
  labs(title = "Error of mean estimators under heavy-tailed data",
       x = "T", y = "normalized error (%)")

p_error_Sigma <- df |>
  ggplot(aes(x = T, y = `error Sigma`, color = method)) +
  geom_line(linewidth = 1.2) +
  labs(color = "estimator") +
  labs(title = "Error of covariance estimators under heavy-tailed data",
       x = "T", y = "normalized error (%)")

p_error_mu / p_error_Sigma
```



Exercise 3.8: Derivation of Gaussian ML estimators

Given T N-dimensional observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, the Gaussian ML estimation for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is formulated as

$$\underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\text{minimize}} \quad \log \det(\boldsymbol{\Sigma}) + \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_t - \boldsymbol{\mu}).$$

Derive the estimators by setting the gradient of the objective function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{-1}$ to zero.

Solution

Defining the precision matrix $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$, we can write the objective function as

$$f(\boldsymbol{\mu}, \boldsymbol{\Theta}) = -\log \det(\boldsymbol{\Theta}) + \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Theta} (\boldsymbol{x}_t - \boldsymbol{\mu}),$$

which is convex in $\boldsymbol{\mu}$ and in $\boldsymbol{\Theta}$.

- The gradient with respect to $\boldsymbol{\mu}$ is

$$\begin{aligned} \nabla_{\boldsymbol{\mu}} f(\boldsymbol{\mu}, \boldsymbol{\Theta}) &= \nabla_{\boldsymbol{\mu}} \left[\frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Theta} (\boldsymbol{x}_t - \boldsymbol{\mu}) \right] \\ &= \frac{1}{T} \sum_{t=1}^{T} (-2\boldsymbol{\Theta}(\boldsymbol{x}_t - \boldsymbol{\mu})) \\ &= -\frac{2}{T} \boldsymbol{\Theta} \sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu}). \end{aligned}$$

  Setting this gradient to zero leads to the solution (independent of $\boldsymbol{\Theta}$)

$$\sum_{t=1}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu}) = \mathbf{0}$$

  and the closed-form expression for the ML estimator for the mean is

$$\hat{\boldsymbol{\mu}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{x}_t.$$

- To derive the solution for the covariance matrix, it is convenient to rewrite the objective as

$$\begin{aligned} f(\boldsymbol{\mu}, \boldsymbol{\Theta}) &= -\log\det(\boldsymbol{\Theta}) + \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t - \boldsymbol{\mu}) \\ &= -\log\det(\boldsymbol{\Theta}) + \frac{1}{T}\sum_{t=1}^{T}\mathrm{tr}[\boldsymbol{\Theta}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}] \\ &= -\log\det(\boldsymbol{\Theta}) + \mathrm{tr}\left[\boldsymbol{\Theta}\frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}\right] \\ &= -\log\det(\boldsymbol{\Theta}) + \mathrm{tr}\left[\boldsymbol{\Theta}\boldsymbol{S}\right], \end{aligned}$$

  where we have used the cyclic property of the trace operator, $\mathrm{tr}(\boldsymbol{AB}) = \mathrm{tr}(\boldsymbol{BA})$, and we have introduced the sample covariance matrix

$$\boldsymbol{S} = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}.$$

  The gradient with respect to $\boldsymbol{\Theta}$ is

$$\nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\mu}, \boldsymbol{\Theta}) = -\boldsymbol{\Theta}^{-1} + \boldsymbol{S}.$$

  Setting this gradient to zero,

$$-\boldsymbol{\Theta}^{-1} + \boldsymbol{S} = \boldsymbol{0},$$

  leads to the closed-form solution:

$$\boldsymbol{\Theta}^{-1} = \boldsymbol{S}.$$

  The final expression for the ML estimator for the covariance matrix (evaluated at the optimal sample mean) is

$$\hat{\boldsymbol{\Sigma}} = \boldsymbol{S} = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}.$$

Summarizing, the Gaussian maximum likelihood estimators are:

- **Mean estimator:**

$$\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t$$

- **Covariance estimator:**

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}$$

These results show that the ML estimator for the mean is simply the sample mean, while the ML estimator for the covariance matrix is the sample covariance matrix (using the ML estimate of the mean). Note that this covariance estimator uses the divisor T rather than $T - 1$, making it the maximum likelihood estimator rather than the unbiased estimator.

Exercise 3.9: Derivation of heavy-tailed ML estimators

Given T N-dimensional observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, the heavy-tailed ML estimation (under the t distribution with degrees of freedom ν) for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is formulated as

$$\underset{\boldsymbol{\mu},\boldsymbol{\Sigma}}{\text{minimize}} \quad \log\det(\boldsymbol{\Sigma}) + \frac{\nu+N}{T}\sum_{t=1}^{T}\log\left(1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_t-\boldsymbol{\mu})\right).$$

Derive the fixed-point equations characterizing the estimators by setting the gradient of the objective function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{-1}$ to zero.

Solution

Defining the precision matrix $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$, we can write the objective function as

$$f(\boldsymbol{\mu},\boldsymbol{\Theta}) = -\log\det(\boldsymbol{\Theta}) + \frac{\nu+N}{T}\sum_{t=1}^{T}\log\left(1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})\right)$$

- The gradient with respect to $\boldsymbol{\mu}$ is

$$\begin{aligned}\nabla_{\boldsymbol{\mu}} f(\boldsymbol{\mu},\boldsymbol{\Theta}) &= \frac{\nu+N}{T}\sum_{t=1}^{T}\nabla_{\boldsymbol{\mu}}\log\left(1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})\right)\\ &= \frac{\nu+N}{T}\sum_{t=1}^{T}\frac{1}{1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})}\cdot\frac{1}{\nu}\cdot(-2\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu}))\\ &= -\frac{2(\nu+N)}{\nu T}\sum_{t=1}^{T}\frac{\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})}{1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})}.\end{aligned}$$

  Setting this gradient to zero leads to

$$\sum_{t=1}^{T}\frac{\boldsymbol{x}_t-\boldsymbol{\mu}}{1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})} = \boldsymbol{0}.$$

  Defining the weight $w_t = \frac{\nu+N}{\nu+(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_t-\boldsymbol{\mu})}$, we can write the fixed-point equation for $\boldsymbol{\mu}$ as

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{t=1}^{T} w_t\boldsymbol{x}_t}{\sum_{t=1}^{T} w_t}.$$

- To derive the solution for the covariance matrix, it is convenient to rewrite the objective using the trace operator as

$$f(\boldsymbol{\mu},\boldsymbol{\Theta}) = -\log\det(\boldsymbol{\Theta}) + \frac{\nu+N}{T}\sum_{t=1}^{T}\log\left(1+\frac{1}{\nu}\mathrm{tr}[\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}]\right).$$

  The gradient with respect to $\boldsymbol{\Theta}$ is

$$\nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\mu},\boldsymbol{\Theta}) = -\boldsymbol{\Theta}^{-1} + \frac{\nu+N}{\nu T}\sum_{t=1}^{T}\frac{(\boldsymbol{x}_t-\boldsymbol{\mu})(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}}{1+\frac{1}{\nu}(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t-\boldsymbol{\mu})}$$

Setting this gradient to zero lead to the fixed-point equation:

$$\boldsymbol{\Theta}^{-1} = \frac{\nu + N}{\nu T} \sum_{t=1}^{T} \frac{(\boldsymbol{x}_t - \boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}}{1 + \frac{1}{\nu}(\boldsymbol{x}_t - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Theta}(\boldsymbol{x}_t - \boldsymbol{\mu})}$$

Using the same weight definition $w_t = \frac{\nu+N}{\nu+(\boldsymbol{x}_t-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_t-\boldsymbol{\mu})}$, the fixed-point equation for $\boldsymbol{\Sigma}$ becomes:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{T} \sum_{t=1}^{T} w_t (\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}.$$

Summarizing, the heavy-tailed ML estimators are characterized by the following fixed-point equations:

- **Weight computation:**

$$w_t = \frac{\nu + N}{\nu + (\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}\hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})}$$

- **Mean estimator:**

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{t=1}^{T} w_t \boldsymbol{x}_t}{\sum_{t=1}^{T} w_t}$$

- **Covariance estimator:**

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{T} \sum_{t=1}^{T} w_t (\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}})^\mathsf{T}$$

These equations must be solved iteratively since the weights w_t depend on both $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$, creating a coupled system. The weights give less influence to outlying observations (those with large Mahalanobis distances), which is the key mechanism by which the t distribution provides robustness against outliers compared to the Gaussian distribution.

Exercise 3.10: Shrinkage James–Stein estimator for the sample mean

Consider a Gaussian-distributed i.i.d. N-dimensional time series with zero mean and identity covariance matrix, $\boldsymbol{x}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}),\ t = 1, \ldots, T$.

a. Generate data for $N = 10$ and $T = 20$, and estimate the mean vector with the sample mean and with the shrinkage James–Stein estimator.
b. Run the experiment multiple times and compute the mean squared error of the estimators.
c. Finally, repeat the experiment multiple times, for different numbers of observations $T =$

$10, 20, \ldots, 100$, and plot the mean squared error as a function of T.

Solution

For convenience, we will define a function to compute the James-Stein shrinkage estimator:

```
shrink_mu <- function(X, mu, mu_target, cov) {
  T <- nrow(X)
  N <- ncol(X)

  # shrinkage
  rho <- (N + 2) /
    ((N + 2) + T * as.numeric((mu - mu_target) %*% solve(cov, mu - mu_target)))
  rho <- max(0, min(1, rho))
  mu_shrinked <- (1 - rho) * mu + rho * mu_target
  return(mu_shrinked)
}
```

a. Single experiment:

```r
library(mvtnorm)

# Parameters
N <- 10
T <- 20

mu_true <- rep(0, N)  # True mean vector
mu_target <- rep(0, N)  # Target for shrinkage (zero vector)
Sigma_true <- diag(N)

# Generate sample
X <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

# Compute estimators
mu <- colMeans(X)
mu_JS <- shrink_mu(X, mu = colMeans(X), mu_target = mu_target, cov = Sigma_true)

# Compute MSE for both estimators
mse_sample <- sum((mu - mu_true)^2)
mse_js <- sum((mu_JS - mu_true)^2)

# Display
cat("Results from single experiment with T =", T, ":\n")
cat("\nMean Squared Error (MSE):\n")
cat("MSE of Sample Mean:", round(mse_sample, 4), "\n")
cat("MSE of James-Stein:", round(mse_js, 4), "\n")
cat("Improvement ratio (Sample MSE / JS MSE):", round(mse_sample / mse_js, 4), "\n")
```

```
Results from single experiment with T = 20 :

Mean Squared Error (MSE):
MSE of Sample Mean: 0.2693
MSE of James-Stein: 0.0258
Improvement ratio (Sample MSE / JS MSE): 10.4216
```

b. Multiple experiments with fixed T:

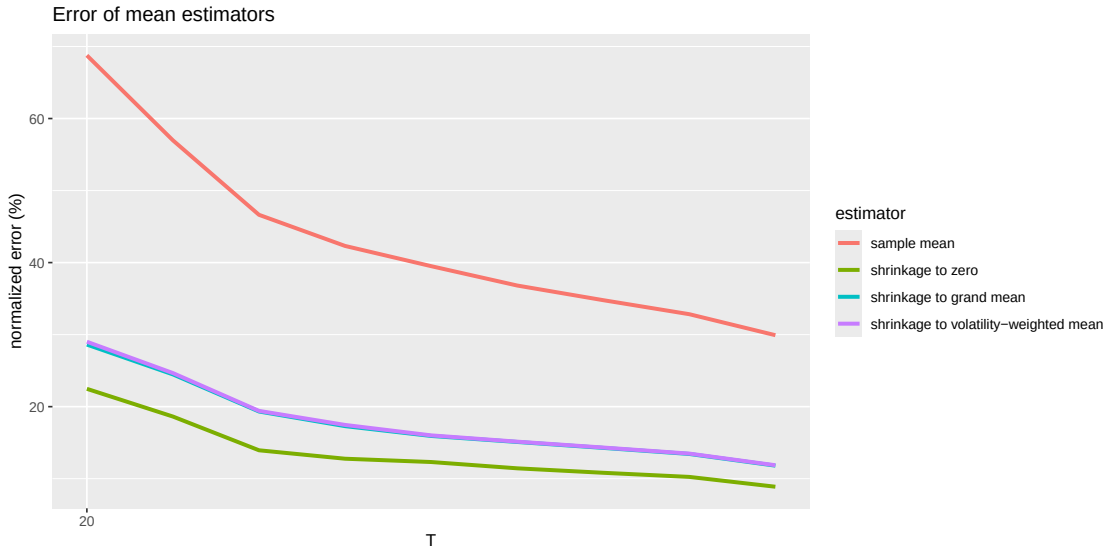
```r
set.seed(42)  # For reproducibility
num_experiments <- 100
N <- 10
T <- 20

mu_true <- rep(0, N)
mu_target <- rep(0, N)
Sigma_true <- diag(N)

# Storage for MSE values
mse_sample_vec <- numeric(num_experiments)
mse_js_vec <- numeric(num_experiments)

# Run multiple experiments
for (i in 1:num_experiments) {
  # Generate sample
  X <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

  # Compute estimators
  mu <- colMeans(X)
  mu_JS <- shrink_mu(X, mu = colMeans(X), mu_target = mu_target, cov = Sigma_true)

  # Compute MSE for both estimators
  mse_sample_vec[i] <- sum((mu - mu_true)^2)
  mse_js_vec[i] <- sum((mu_JS - mu_true)^2)
}

# Compute average MSE across experiments
avg_mse_sample <- mean(mse_sample_vec)
avg_mse_js <- mean(mse_js_vec)

cat("Results from", num_experiments, "experiments with T =", T, ":\n")
cat("Average MSE of Sample Mean:", round(avg_mse_sample, 4), "\n")
cat("Average MSE of James-Stein:", round(avg_mse_js, 4), "\n")
cat("Average improvement ratio:", round(avg_mse_sample / avg_mse_js, 4), "\n")
cat("Standard deviation of Sample MSE:", round(sd(mse_sample_vec), 4), "\n")
cat("Standard deviation of JS MSE:", round(sd(mse_js_vec), 4), "\n")
```

```
Results from 100 experiments with T = 20 :
Average MSE of Sample Mean: 0.4993
Average MSE of James-Stein: 0.1219
Average improvement ratio: 4.0968
Standard deviation of Sample MSE: 0.2376
Standard deviation of JS MSE: 0.1159
```

c. Multiple experiments as a function of T:

```r
set.seed(42)

# main loop
df <- data.frame()
T_sweep <- seq(from = 20, by = 10, to = 100)
for(T in T_sweep) {
  for (i in 1:200) {
    X_ <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

    Sigma <- cov(X_)
    Sigma_reg <- Sigma + mean(diag(Sigma)) * diag(N)    # regularize Sigma for stability

    # sample mean
    mu    <- colMeans(X_)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "mu",
                               "method"       = "sample mean",
                               "error"        = norm(mu - mu_true, "2"),
                               check.names    = FALSE))
    # sample mean + shrinkage to zero
    mu_JS <- shrink_mu(X_, mu = colMeans(X_), mu_target = rep(0, N), cov = Sigma_reg)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "mu",
                               "method"       = "shrinkage to zero",
                               "error"        = norm(mu_JS - mu_true, "2"),
                               check.names    = FALSE))
    # sample mean + shrinkage to grand mean
    mu_JS <- shrink_mu(X_, mu = colMeans(X_), mu_target = rep(mean(mu), N), cov = Sigma_reg)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "mu",
                               "method"       = "shrinkage to grand mean",
                               "error"        = norm(mu_JS - mu_true, "2"),
                               check.names    = FALSE))
    # sample mean + shrinkage to volatility weighted mean
    vol <- sqrt(diag(Sigma))  # target is: mean(mu/vol) * vol
    mu_JS <- shrink_mu(X_, mu = colMeans(X_), mu_target = mean(mu/vol) * vol, cov = Sigma_reg)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "mu",
                               "method"       = "shrinkage to volatility-weighted mean",
                               "error"        = norm(mu_JS - mu_true, "2"),
                               check.names    = FALSE))
  }
}
```

```r
# plot
df$method <- factor(df$method, levels = unique(df$method))

df <- df |>
  group_by(parameter, method, T) |>
  summarize("error" = 100*mean(`error`)) |>
  ungroup()

df[df$parameter == "mu", ] |>
  ggplot(aes(x = T, y = `error`, color = method)) +
  geom_line(linewidth = 1.2) +
  labs(color = "estimator") +
  scale_x_continuous(breaks = seq(from = T_sweep[1], to = last(T_sweep), by = 100)) +
  labs(title = "Error of mean estimators", x= "T", y = "normalized error (%)")
```



Exercise 3.11: Shrinkage sample covariance matrix estimator

Consider a Gaussian-distributed i.i.d. N-dimensional time series with zero mean and identity covariance matrix, $\boldsymbol{x}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $t = 1, \ldots, T$.

a. Generate data for $N = 10$ and $T = 20$, and estimate the covariance matrix with the sample

covariance matrix and with the shrinkage Ledoit–Wolf estimator.

b. Run the experiment multiple times and compute the mean squared error of the estimators.

c. Finally, repeat the experiment multiple times, for different numbers of observations $T = 10, 20, \ldots, 100$, and plot the mean squared error as a function of T.

Solution

For convenience, we will define a function to compute the Ledoit–Wolf shrinkage estimator:

```
shrink_Sigma <- function(X, S, Sigma_target) {
  T <- nrow(X)
  N <- ncol(X)
  Xc <- X - matrix(colMeans(X), T, N, byrow = TRUE)

  # shrinkage
  rho <- min(1, ((1/T^2) * sum(rowSums(Xc^2)^2) - 1/T * sum(S^2)) / sum((S - Sigma_target)^2))
  Sigma_sh <- (1 - rho)*S + rho*Sigma_target
  return(Sigma_sh)
}
```

a. Single experiment:

```r
library(mvtnorm)

# Parameters
N <- 10
T <- 20

mu_true <- rep(0, N)
Sigma_true <- diag(N)

# Generate sample
X <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

# Compute estimators
S <- cov(X)
Sigma_target <- mean(diag(S)) * diag(N)
Sigma_LZ_sh <- shrink_Sigma(X, S, Sigma_target)

# Compute MSE for both estimators
mse_sample_cov <- norm(S - Sigma_true, "F")/norm(Sigma_true, "F")
mse_LZ_sh <- norm(Sigma_LZ_sh - Sigma_true, "F")/norm(Sigma_true, "F")

# Display
cat("Results from single experiment with T =", T, ":\n")
cat("\nNormalized Mean Squared Error (NMSE):\n")
cat("NMSE of Sample Covariance matrix:", round(mse_sample_cov, 4), "\n")
cat("NMSE of Ledoit-Wolf shrinkage:", round(mse_LZ_sh, 4), "\n")
cat("Improvement ratio:", round(mse_sample_cov / mse_LZ_sh, 4), "\n")
```

```
Results from single experiment with T = 20 :

Normalized Mean Squared Error (NMSE):
NMSE of Sample Covariance matrix: 0.6583
NMSE of Ledoit-Wolf shrinkage: 0.2747
Improvement ratio: 2.3969
```

b. Multiple experiments with fixed T:

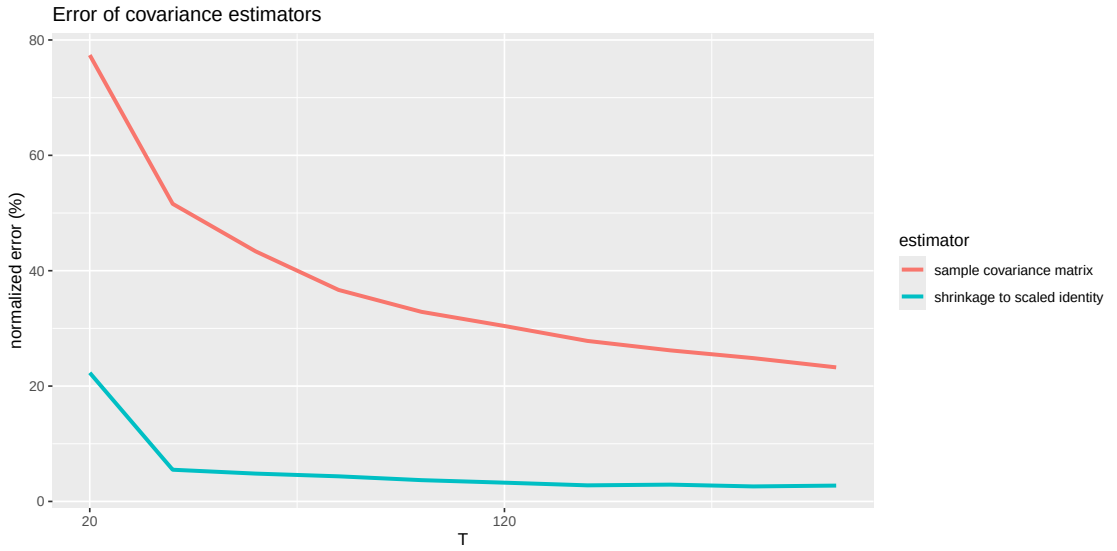
```r
set.seed(42)  # For reproducibility
num_experiments <- 100
N <- 10
T <- 20

mu_true <- rep(0, N)
Sigma_true <- diag(N)

# Storage for MSE values
mse_sample_cov <- numeric(num_experiments)
mse_LZ_sh <- numeric(num_experiments)

# Run multiple experiments
for (i in 1:num_experiments) {
  # Generate sample
  X <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

  # Compute estimators
  S <- cov(X)
  Sigma_target <- mean(diag(S)) * diag(N)
  Sigma_LZ_sh <- shrink_Sigma(X, S, Sigma_target)

  # Compute MSE for both estimators
  mse_sample_cov[i] <- norm(S - Sigma_true, "F")/norm(Sigma_true, "F")
  mse_LZ_sh[i] <- norm(Sigma_LZ_sh - Sigma_true, "F")/norm(Sigma_true, "F")
}

# Compute average MSE across experiments
avg_mse_sample_cov <- mean(mse_sample_cov)
avg_mse_LZ_sh <- mean(mse_LZ_sh)

# Display
cat("Results from", num_experiments, "experiments with T =", T, ":\n")
cat("Average NMSE of Sample Covariance matrix:", round(avg_mse_sample_cov, 4), "\n")
cat("Average NMSE of Ledoit-Wolf shrinkage:", round(avg_mse_LZ_sh, 4), "\n")
cat("Average improvement ratio:", round(avg_mse_sample_cov / avg_mse_LZ_sh, 4), "\n")
```

```
Results from 100 experiments with T = 20 :
Average NMSE of Sample Covariance matrix: 0.7697
Average NMSE of Ledoit-Wolf shrinkage: 0.1912
Average improvement ratio: 4.0255
```

c. Multiple experiments as a function of T:

```r
set.seed(42)

# main loop
df <- data.frame()
T_sweep <- seq(from = 20, by = 20, to = 200)
for(T in T_sweep) {
  for (i in 1:200) {
    X_ <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)

    # SCM
    Sigma <- cov(X_)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "Sigma",
                               "method"       = "sample covariance matrix",
                               "error"        = norm(Sigma - Sigma_true, "F")/norm(Sigma_true, "F")
                               check.names    = FALSE))

    # SCM + Ledoit-Wolf shrinked to scaled identity
    S <- cov(X_)
    Sigma_target <- mean(diag(S)) * diag(N)
    Sigma_LZ_sh <- shrink_Sigma(X, S, Sigma_target)
    df <- rbind(df, data.frame("T"            = T,
                               "parameter"    = "Sigma",
                               "method"       = "shrinkage to scaled identity",
                               "error"        = norm(Sigma_LZ_sh - Sigma_true, "F")/norm(Sigma_true
                               check.names    = FALSE))
  }
}
```

```r
# plot
df$method <- factor(df$method, levels = unique(df$method))

df <- df |>
  group_by(parameter, method, T) |>
  summarize("error" = 100*mean(`error`)) |>
  ungroup()

df[df$parameter == "Sigma", ] |>
  ggplot(aes(x = T, y = `error`, color = method)) +
  geom_line(linewidth = 1.2) +
  labs(color = "estimator") +
  scale_x_continuous(breaks = seq(from = T_sweep[1], to = last(T_sweep), by = 100)) +
  labs(title = "Error of covariance estimators", x= "T", y = "normalized error (%)")
```



Exercise 3.12: Factor model estimator

Consider a Gaussian-distributed i.i.d. N-dimensional time series with zero mean and covariance matrix with a single-factor structure $\boldsymbol{\Sigma} = \boldsymbol{\beta}\boldsymbol{\beta}^\mathsf{T} + \boldsymbol{I}$ (e.g., with $\boldsymbol{\beta} = \mathbf{1}$), $\boldsymbol{x}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, $t = 1, \ldots, T$.

a. Generate data for $N = 10$ and $T = 20$, and estimate the covariance matrix with the sample covariance matrix and with the single-factor model structure (e.g., with PCA).
b. Run the experiment multiple times and compute the mean squared error of the estimators.
c. Finally, repeat the experiment multiple times, for different numbers of observations $T = 10, 20, \ldots, 100$, and plot the mean squared error as a function of T.

Solution

a. Single experiment:

```r
# Parameters
N <- 10
T <- 20

mu_true <- rep(0, N)
beta <- runif(N)
Sigma_true <- beta %o% beta + diag(N)

# Generate synthetic data
X <- rmvnorm(n = T, mean = mu_true, sigma = Sigma_true)
```

```r
# Sample covariance matrix
S <- cov(X)

# Single-factor model estimator using PCA
pca_result <- prcomp(X)
loadings_pc1 <- pca_result$rotation[, 1]
variance_noise <- mean(pca_result$sdev[-1]^2)
variance_pc1 <- pca_result$sdev[1]^2 - variance_noise

# Single-factor model covariance matrix
beta_hat <- loadings_pc1 * sqrt(variance_pc1)
Sigma_1factor <- beta_hat %o% beta_hat + variance_noise*diag(N)
```

```r
# Compute MSE for both estimators
mse_scm <- norm(S - Sigma_true, "F")/norm(Sigma_true, "F")
mse_1factor <- norm(Sigma_1factor - Sigma_true, "F")/norm(Sigma_true, "F")

# Display
cat("Results from single experiment with T =", T, ":\n")
cat("\nNormalized Mean Squared Error (NMSE):\n")
cat("NMSE of Sample Covariance matrix:", round(mse_scm, 4), "\n")
cat("NMSE of PCA with single factor:", round(mse_1factor, 4), "\n")
cat("Improvement ratio:", round(mse_scm / mse_1factor, 4), "\n")
```

```
Results from single experiment with T = 20 :

Normalized Mean Squared Error (NMSE):
NMSE of Sample Covariance matrix: 0.5425
NMSE of PCA with single factor: 0.3128
Improvement ratio: 1.7345
```

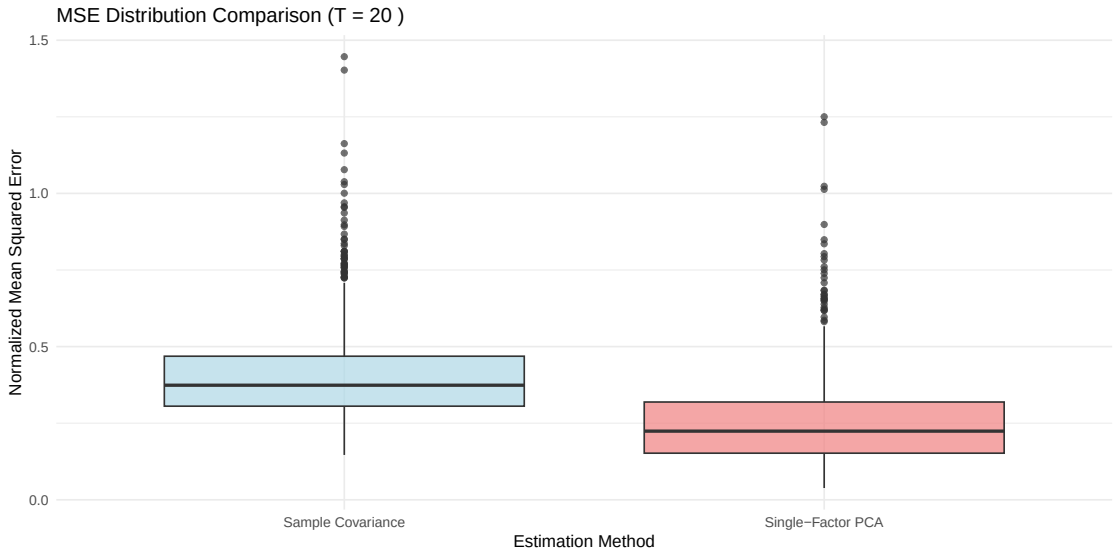
b. Multiple experiments with fixed T:

```r
library(mvtnorm)

# Function to run a single experiment
run_single_experiment <- function(N, T, beta) {
  # True covariance matrix
  Sigma_true <- beta %o% beta + diag(N)

  # Generate synthetic data
  X <- rmvnorm(n = T, mean = rep(0, N), sigma = Sigma_true)

  # Sample covariance matrix
  S <- cov(X)

  # Single-factor model estimator using PCA
  pca_result <- prcomp(X)
  loadings_pc1 <- pca_result$rotation[, 1]
  variance_noise <- mean(pca_result$sdev[-1]^2)
  variance_pc1 <- pca_result$sdev[1]^2 - variance_noise

  # Single-factor model covariance matrix
  beta_hat <- loadings_pc1 * sqrt(variance_pc1)
  Sigma_1factor <- beta_hat %o% beta_hat + variance_noise * diag(N)

  # Compute normalized MSE for both estimators
  mse_scm <- norm(S - Sigma_true, "F")^2 / norm(Sigma_true, "F")^2
  mse_1factor <- norm(Sigma_1factor - Sigma_true, "F")^2 / norm(Sigma_true, "F")^2

  return(list(mse_scm = mse_scm, mse_1factor = mse_1factor))
}
```

```r
# Parameters
N <- 10
T <- 20
beta <- runif(N)
num_experiments <- 1000

# Run multiple experiments
set.seed(123)
results <- replicate(num_experiments, run_single_experiment(N, T, beta), simplify = FALSE)

# Extract MSE values
mse_scm_values <- sapply(results, function(x) x$mse_scm)
mse_1factor_values <- sapply(results, function(x) x$mse_1factor)

# Compute mean MSE
mean_mse_scm <- mean(mse_scm_values)
mean_mse_1factor <- mean(mse_1factor_values)

# Display results
cat("Results from", num_experiments, "experiments with T =", T, ":\n")
cat("\nMean Normalized Mean Squared Error (NMSE):\n")
cat("NMSE of Sample Covariance matrix:", round(mean_mse_scm, 4), "\n")
cat("NMSE of PCA with single factor:", round(mean_mse_1factor, 4), "\n")
cat("Improvement ratio:", round(mean_mse_scm / mean_mse_1factor, 4), "\n")
cat("Standard deviation of SCM NMSE:", round(sd(mse_scm_values), 4), "\n")
cat("Standard deviation of 1-factor NMSE:", round(sd(mse_1factor_values), 4), "\n")
```

```
Results from 1000 experiments with T = 20 :

Mean Normalized Mean Squared Error (NMSE):
NMSE of Sample Covariance matrix: 0.4064
NMSE of PCA with single factor: 0.2532
Improvement ratio: 1.605
Standard deviation of SCM NMSE: 0.1519
Standard deviation of 1-factor NMSE: 0.1467
```

```r
library(ggplot2)
# Create boxplot comparison
mse_data <- data.frame(
  MSE = c(mse_scm_values, mse_1factor_values),
  Method = rep(c("Sample Covariance", "Single-Factor PCA"), each = num_experiments)
)

ggplot(mse_data, aes(x = Method, y = MSE, fill = Method)) +
  geom_boxplot(alpha = 0.7) +
  scale_fill_manual(values = c("Sample Covariance" = "lightblue",
                               "Single-Factor PCA" = "lightcoral")) +
  labs(title = paste("MSE Distribution Comparison (T =", T, ")"),
       x = "Estimation Method",
       y = "Normalized Mean Squared Error") +
  theme_minimal() +
  theme(legend.position = "none")
```



c. Multiple experiments as a function of T:

```r
# Function to run experiments for different T values
run_experiments_varying_T <- function(T_values, N, beta, num_experiments = 500) {
  results_df <- data.frame()

  for (T in T_values) {
    cat("Running experiments for T =", T, "\n")

    # Run experiments for current T
    results <- replicate(num_experiments, run_single_experiment(N, T, beta), simplify = FALSE)

    # Extract MSE values
    mse_scm_values <- sapply(results, function(x) x$mse_scm)
    mse_1factor_values <- sapply(results, function(x) x$mse_1factor)

    # Compute statistics
    mean_mse_scm <- mean(mse_scm_values)
    mean_mse_1factor <- mean(mse_1factor_values)
    se_mse_scm <- sd(mse_scm_values) / sqrt(num_experiments)
    se_mse_1factor <- sd(mse_1factor_values) / sqrt(num_experiments)

    # Store results
    temp_df <- data.frame(
      T = T,
      Method = c("Sample Covariance", "Single-Factor PCA"),
      Mean_MSE = c(mean_mse_scm, mean_mse_1factor),
      SE_MSE = c(se_mse_scm, se_mse_1factor)
    )

    results_df <- rbind(results_df, temp_df)
  }

  return(results_df)
}
```

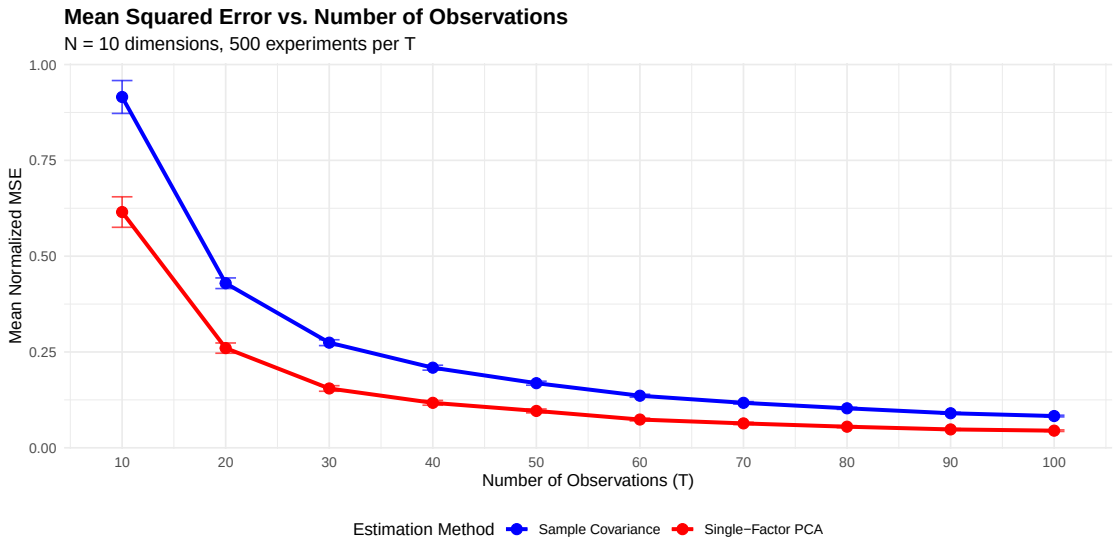
```r
# Parameters
N <- 10
T_values <- seq(10, 100, by = 10)
beta <- runif(N)
num_experiments <- 500

# Run experiments
set.seed(123)
results_varying_T <- run_experiments_varying_T(T_values, N, beta, num_experiments)
```

```
Running experiments for T = 10
Running experiments for T = 20
```

```
Running experiments for T = 30
Running experiments for T = 40
Running experiments for T = 50
Running experiments for T = 60
Running experiments for T = 70
Running experiments for T = 80
Running experiments for T = 90
Running experiments for T = 100
```

```r
# Create the plot
ggplot(results_varying_T, aes(x = T, y = Mean_MSE, color = Method)) +
  geom_line(size = 1.2) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = Mean_MSE - 1.96 * SE_MSE,
                    ymax = Mean_MSE + 1.96 * SE_MSE),
                width = 2, alpha = 0.7) +
  scale_color_manual(values = c("Sample Covariance" = "blue",
                                "Single-Factor PCA" = "red")) +
  labs(title = "Mean Squared Error vs. Number of Observations",
       subtitle = paste("N =", N, "dimensions,", num_experiments, "experiments per T"),
       x = "Number of Observations (T)",
       y = "Mean Normalized MSE",
       color = "Estimation Method") +
  theme_minimal() +
  theme(legend.position = "bottom",
        plot.title = element_text(size = 14, face = "bold"),
        plot.subtitle = element_text(size = 12)) +
  scale_x_continuous(breaks = T_values)
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```




```
# Display summary statistics
cat("\nSummary of results:\n")
```

```
Summary of results:
```

```
print(results_varying_T)
```

```
     T            Method   Mean_MSE      SE_MSE
1   10 Sample Covariance 0.91538616 0.021887117
2   10 Single-Factor PCA 0.61515693 0.020309448
3   20 Sample Covariance 0.42935896 0.007077762
4   20 Single-Factor PCA 0.26011617 0.006827174
5   30 Sample Covariance 0.27426260 0.003880434
6   30 Single-Factor PCA 0.15476176 0.003707439
7   40 Sample Covariance 0.20901161 0.003268171
8   40 Single-Factor PCA 0.11731800 0.003177076
9   50 Sample Covariance 0.16856657 0.002677914
10  50 Single-Factor PCA 0.09616690 0.002595141
11  60 Sample Covariance 0.13583662 0.002019541
12  60 Single-Factor PCA 0.07370669 0.001862286
13  70 Sample Covariance 0.11729081 0.001619109
14  70 Single-Factor PCA 0.06362166 0.001542134
15  80 Sample Covariance 0.10301533 0.001418531
16  80 Single-Factor PCA 0.05501044 0.001340530
17  90 Sample Covariance 0.09011639 0.001245239
```

```
18  90 Single-Factor PCA 0.04792290 0.001166863
19 100 Sample Covariance 0.08283160 0.001125983
20 100 Single-Factor PCA 0.04470006 0.001055740
```

```r
# Compute improvement ratios
improvement_ratios <- results_varying_T[results_varying_T$Method == "Sample Covariance", "Mean_MSE"
                      results_varying_T[results_varying_T$Method == "Single-Factor PCA", "Mean_MSE"]

improvement_df <- data.frame(
  T = T_values,
  Improvement_Ratio = improvement_ratios
)

# Plot improvement ratio
ggplot(improvement_df, aes(x = T, y = Improvement_Ratio)) +
  geom_line(color = "darkgreen", size = 1.2) +
  geom_point(color = "darkgreen", size = 3) +
  geom_hline(yintercept = 1, linetype = "dashed", color = "red", alpha = 0.7) +
  labs(title = "Performance Improvement of Single-Factor PCA over Sample Covariance",
       x = "Number of Observations (T)",
       y = "Improvement Ratio (SCM MSE / PCA MSE)") +
  theme_minimal() +
  scale_x_continuous(breaks = T_values) +
  annotate("text", x = max(T_values) * 0.8, y = 1.1,
           label = "No improvement", color = "red", alpha = 0.7)
```