STAT 5701: Statistical Computing

Homework 2

Jingxiang Li

October 8, 2015

Problem 1.a

R Code Chunk

```r
#' compute the confidence interval of sigma^2 given i.i.d. observations
#'
#' @param x vector of i.i.d. observations
#' @param alph this is 1 - confidence level
#'
#' @return a vector (left, right), (1 - alpha) confidence interval of sigma^2
#' @export
confint_sigma_sq = function(x, alpha = 0.05) {
    n = length(x)
    var_sample = var(x)
    left = (n - 1) * var_sample / qchisq(1 - alpha / 2, df = n - 1)
    right = (n - 1) * var_sample / qchisq(alpha / 2, df = n - 1)
    c(left, right)
}

#' simulate the confidence interval of sigma^2 for i.i.d. normal random
#' observations
#'
#' @param reps number of replications for the simulation study
#' @param n sample size of normal realization
#' @param mu true mean of the normal distribution
#' @param sigma true standard deviation of the normal distribution
#' @param alpha alpha level used for constructing confidence interval of sigma^2
#' @param score_conf confidence level for constructing the confidence interval
#'   of the coverage probability
#'
#' @return a vector (left, right), score_conf level confidence interval for the
#'   coverage probability
#' @export
sim_confint_sigma_sq_norm = function(reps, n, mu, sigma, alpha, score_conf) {
    x_sim = rnorm(n = n * reps, mean = mu, sd = sigma)
    x_sim = matrix(x_sim, nrow = reps, ncol = n)
    int_sim = apply(x_sim, 1, confint_sigma_sq, alpha = alpha)
    indicator_sim = apply(int_sim, 2, function(x) {
        x[1] < sigma ^ 2 && x[2] > sigma ^ 2
    })
    test_result = prop.test(
        sum(indicator_sim), n = reps,
        p = 1 - alpha, conf.level = score_conf
    )
    test_result$conf.int
}

reps = 10000
n = 10
mu = 68
sigma = 3
alpha = 0.05
score_conf = 0.99
sim_confint_sigma_sq_norm(reps, n, mu, sigma, alpha, score_conf)
```

```
## [1] 0.9399294 0.9516740
## attr(,"conf.level")
## [1] 0.99
```

Here we could see the simulated 99% confidence interval for the coverage probability is (0.9399294, 0.9516740), which is very close to 0.95.

Problem 1.b

R Code Chunk

```
#' simulate the confidence interval of sigma^2 for i.i.d. Exp(1) random
#' observations
#'
#' @param number of replications for the simulation study
#' @param n sample size of normal realization
#' @param alpha alpha level used for constructing confidence interval of sigma^2
#' @param score_conf confidence level for constructing the confidence interval
#'   of the coverage probability
#'
#' @return a vector (left, right), score_conf level confidence interval for the
#'   coverage probability
#' @export
sim_confint_sigma_sq_exp = function(reps, n, alpha, score_conf) {
    x_sim = rexp(n = n * reps)
    x_sim = matrix(x_sim, nrow = reps, ncol = n)
    int_sim = apply(x_sim, 1, confint_sigma_sq, alpha = alpha)
    indicator_sim = apply(int_sim, 2, function(x) {
        x[1] < 1 && x[2] > 1
    })
    test_result = prop.test(
        sum(indicator_sim), n = reps,
        p = 1 - alpha, conf.level = score_conf
    )
    test_result$conf.int
}

reps = 10000
n_vec = c(10, 50, 500)
alpha_vec = c(0.01, 0.05)
score_conf = 0.99
result = matrix(0, length(n_vec) * length(alpha_vec), 4)
colnames(result) = c("n", "alpha", "left", "right")
k = 1
for (n in n_vec)
    for (alpha in alpha_vec) {
        result[k, 1] = n
        result[k, 2] = alpha
        result[k, c(3, 4)] = sim_confint_sigma_sq_exp(reps, n, alpha, score_conf)
        k = k + 1
    }
result
```

```
##        n alpha      left     right
## [1,]  10  0.01 0.8768867 0.8933991
## [2,]  10  0.05 0.7563930 0.7782506
## [3,]  50  0.01 0.8225082 0.8418488
## [4,]  50  0.05 0.6951928 0.7187314
## [5,] 500  0.01 0.7931160 0.8136794
## [6,] 500  0.05 0.6626715 0.6868956
```

Here we could see the resulting confidence interval for the coverage probability is far away from $1 - \alpha$, which suggests that this is not a proper way to construct confidence interval of σ^2 for i.i.d. exponential random variables.

Problem 1.c

R Code Chunk

```
#' simulate the confidence interval of sigma^2 for multivariate random
#' observations, with mean = rep(mu, n), covariance matrix var_mat, where
#' var_mat[i, j] = sigma ^ 2 * rho ^ abs(i - j)
#'
#' @param reps number of replications for the simulation study
#' @param n sample size of normal realization
#' @param mu true mean of the normal distribution
#' @param sigma true marginal standard deviation of the normal distribution
#' @param rho decay parameter used for constructing the covariace matrix
#' @param alpha alpha level used for constructing confidence interval of sigma^2
#' @param score_conf confidence level for constructing the confidence interval
#'   of the coverage probability
#'
#' @return a vector (left, right), score_conf level confidence interval for the
#'   coverage probability
#' @export
sim_confint_sigma_sq_multinorm = function(reps, n, mu, sigma, rho, alpha, score_conf) {
    mean_vec = rep(mu, n)
    var_mat = matrix(0, n, n)
    for (i in 1:n)
        for (j in 1:n) {
            var_mat[i, j] = sigma ^ 2 * rho ^ abs(i - j)
        }
    x_sim = mvrnorm(reps, mean_vec, var_mat)
    int_sim = apply(x_sim, 1, confint_sigma_sq, alpha = alpha)
    indicator_sim = apply(int_sim, 2, function(x) {
        x[1] < sigma ^ 2 && x[2] > sigma ^ 2
    })
    test_result = prop.test(
        sum(indicator_sim), n = reps,
        p = 1 - alpha, conf.level = score_conf
    )
    test_result$conf.int
}

reps = 10000
```

```
n_vec = c(10, 50, 500)
alpha_vec = c(0.01, 0.05)
score_conf = 0.99
mu = 68
sigma = 3
rho = 0.7
result = matrix(0, length(n_vec) * length(alpha_vec), 4)
colnames(result) = c("n", "alpha", "left", "right")
k = 1
for (n in n_vec)
    for (alpha in alpha_vec) {
        result[k, 1] = n
        result[k, 2] = alpha
        result[k, c(3, 4)] =
            sim_confint_sigma_sq_multinorm(reps, n, mu, sigma, rho, alpha, score_conf)
        k = k + 1
    }
result
```

```
##        n alpha      left     right
## [1,]  10  0.01 0.9311270 0.9436879
## [2,]  10  0.05 0.7870039 0.8077995
## [3,]  50  0.01 0.8685464 0.8855503
## [4,]  50  0.05 0.7373139 0.7597548
## [5,] 500  0.01 0.8570336 0.8746781
## [6,] 500  0.05 0.7357929 0.7582778
```

Here we could see the resulting confidence interval for the coverage probability is far away from $1-\alpha$, which suggests that this is not a proper way to construct confidence interval of σ^2 for dependent normal random variables.

Problem 2.a

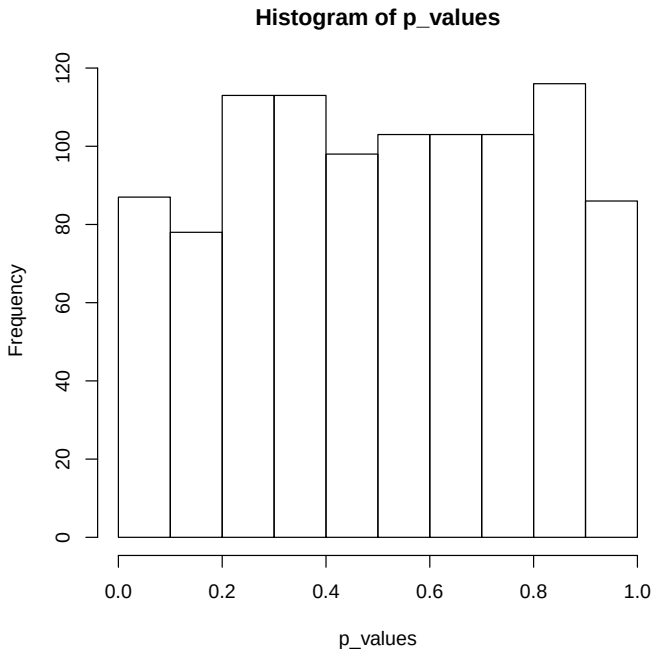
R Code Chunk

```r
#' simulate p-values from a t-test scenario where two samples are drawn from
#' normal distribution
#'
#' @param reps number of replications for this simulation
#' @param n1 sample size of sample #1
#' @param n2 sample size of sample #2
#' @param mu1 mean value of sample #1
#' @param mu2 mean value of sample #2
#' @param sigma1 standard deviation of sample #1
#' @param sigma2 standard deviation of sample #2
#'
#' @return a vector of p-values calculated from a series of t-tests
sim_pval_ttest = function(reps, n1, n2, mu1, mu2, sigma1, sigma2) {
    x1 = rnorm(reps * n1, mu1, sigma1)
    x1 = matrix(x1, reps, n1)
    x2 = rnorm(reps * n2, mu2, sigma2)
    x2 = matrix(x2, reps, n2)
    p_values = numeric(reps)
    for (r in 1:reps) {
        test_result = t.test(x1[r,], x2[r,], var.equal = TRUE)
        p_values[r] = test_result$p.value
    }
    p_values
}

#' calculate the power of a test given a simulated p-value vector
#'
#' @param vector of simulated p-value based on the test
#' @param significance level to determine the rejection region
#'
#' @return the power of the test
calc_power = function(pvals, alpha) {
    sum(pvals > (1 - alpha / 2) | pvals < (alpha / 2)) / length(pvals)
}

n1 = n2 = 50
mu1 = mu2 = 68
sigma1 = sigma2 = 3
reps = 1000

p_values = sim_pval_ttest(reps, n1, n2, mu1, mu2, sigma1, sigma2)
hist(p_values)
```



The actual distribution of the random p-value is Unif$(0, 1)$.

Proof. Note that if $u \sim \text{Unif}(0, 1)$, $\text{P}(u < \alpha) = \alpha,\ \forall \alpha \in [0, 1]$.
Let p be the p-value, under H_0 we have

$$\begin{aligned} & \text{P}(p < \alpha) \\ = \ & \text{P}(t > T_{1-\alpha/2} \text{ or } t < T_{\alpha/2}) \\ = \ & \text{P}(t > T_{1-\alpha/2}) + \text{P}(t < T_{\alpha/2}) \\ = \ & \alpha/2 + \alpha/2 \\ = \ & \alpha \end{aligned}$$

i.e., $\forall \alpha \in [0, 1]\ \text{P}(p < \alpha) = \alpha$, which implies that p-value has a uniform distribution under H_0. Q.E.D.

Problem 2.b

R Code Chunk

```
n1 = n2 = 50
mu1 = 68
sigma1 = sigma2 = 3
alpha = 0.05
reps = 1000

length_mu2 = 5
mu2_vec = seq(from = mu1, by = 1, length.out = length_mu2)
power_result = numeric(length_mu2)
for (i in 1:length_mu2) {
    p_values = sim_pval_ttest(reps, n1, n2, mu1, mu2_vec[i], sigma1, sigma2)
    power_result[i] = calc_power(p_values, alpha)
}
power_result[power_result < 1]
```

```
## [1] 0.050 0.295 0.855 0.999
```

```
mu2_vec[power_result < 1]
```

```
## [1] 68 69 70 71
```

The sequence of values for μ_2 we select is 68, 69, 70, 71, and the corresponding power for these values are 0.050, 0.295, 0.855, 0.999.

Problem 2.c

R Code Chunk

```
mu1 = 68
mu2 = 68.5
sigma1 = sigma2 = 3
alpha = 0.05
reps = 1000

length_n = 10
n_vec = seq(from = 1050, by = 20, length.out = length_n)
power_result = numeric(length_n)
for (i in 1:length_n) {
    p_values = sim_pval_ttest(reps, n_vec[i], n_vec[i], mu1, mu2, sigma1, sigma2)
    power_result[i] = calc_power(p_values, alpha)
}

power_result[which.min(abs(power_result - 0.95))]
```

```
## [1] 0.953
```

```
n_vec[which.min(abs(power_result - 0.95))]
```

```
## [1] 1110
```

The sample size we find is 1110, with corresponding power 0.953.

Problem 2.d

R Code Chunk

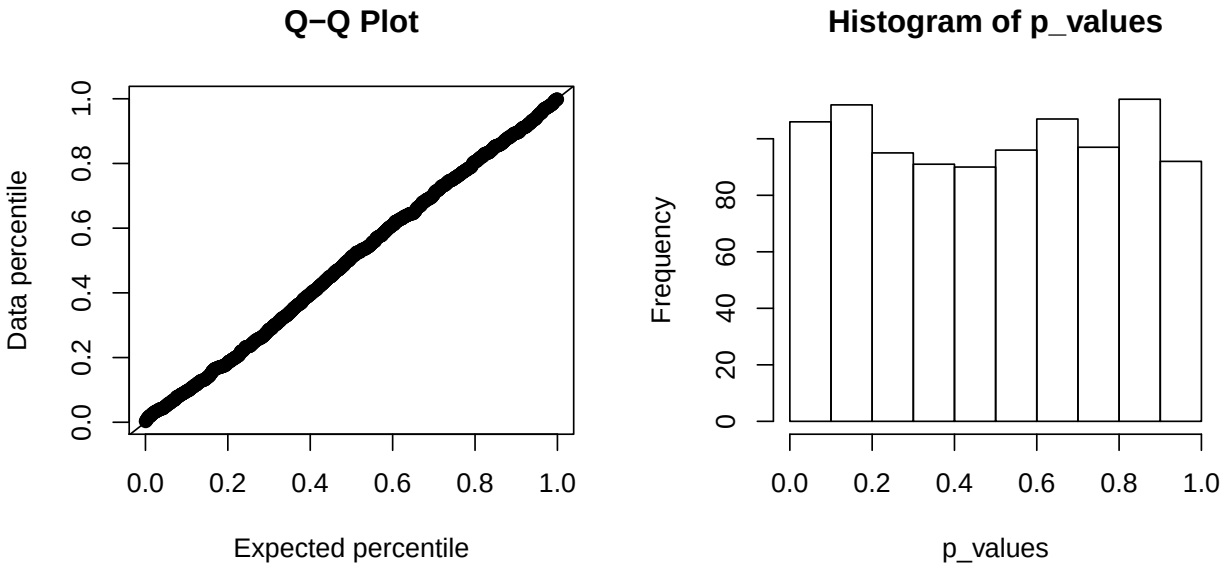
```
#' Draw qqplot (modified from Adam's code)
#'
#' @param x.list target sample
#' @param quant.func quantile function for the target distribution
#' @param ... parameters to be passed to the quant.func()
#'
#' @return no return value
adam.qqplot = function (x.list, quant.func, ...) {
```

```
    n <- length(x.list)
    probs <- ppoints(n)
    plot(
        quant.func(probs, ...), quantile(x.list, probs),
        xlab = "Expected percentile", ylab = "Data percentile",
        main = "Q-Q Plot"
    )
    abline(0, 1)
}

n1 = n2 = 100
mu1 = mu2 = 68
sigma1 = 3
sigma2 = 6
reps = 1000

p_values = sim_pval_ttest(reps, n1, n2, mu1, mu2, sigma1, sigma2)
par(mfrow = c(1, 2))
adam.qqplot(p_values, qunif)
hist(p_values)
quantile(p_values, c(0.01, 0.05))
```

```
##         1%         5%
## 0.01870995 0.05100711
```



In this scenario, $n1$ is equal to $n2$. The QQ-plot shows that the distribution of simulated p-values is perfectly uniform, and the true 0.01, 0.05 data quantiles are very close to 0.01 and 0.05, suggesting that with equal sample size, the difference in variance is not a problem.

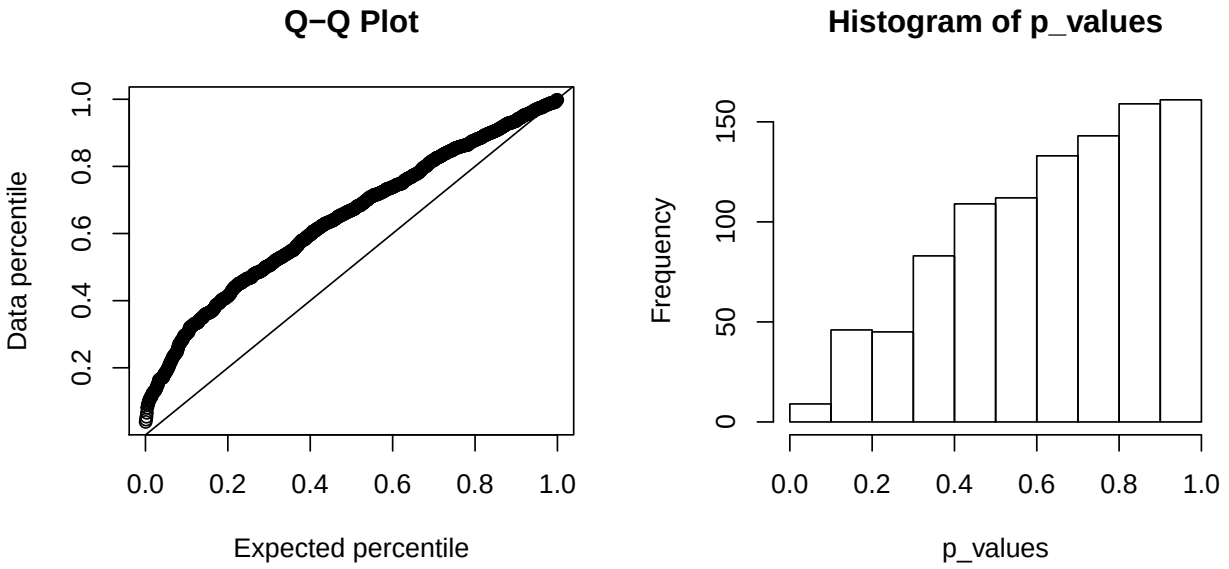
R Code Chunk

```
n1 = 20
n2 = 100
mu1 = mu2 = 68
sigma1 = 3
sigma2 = 6
```

```
reps = 1000

p_values = sim_pval_ttest(reps, n1, n2, mu1, mu2, sigma1, sigma2)
par(mfrow = c(1, 2))
adam.qqplot(p_values, qunif)
hist(p_values)
quantile(p_values, c(0.01, 0.05))
```

```
##        1%        5%
## 0.1054931 0.1870478
```



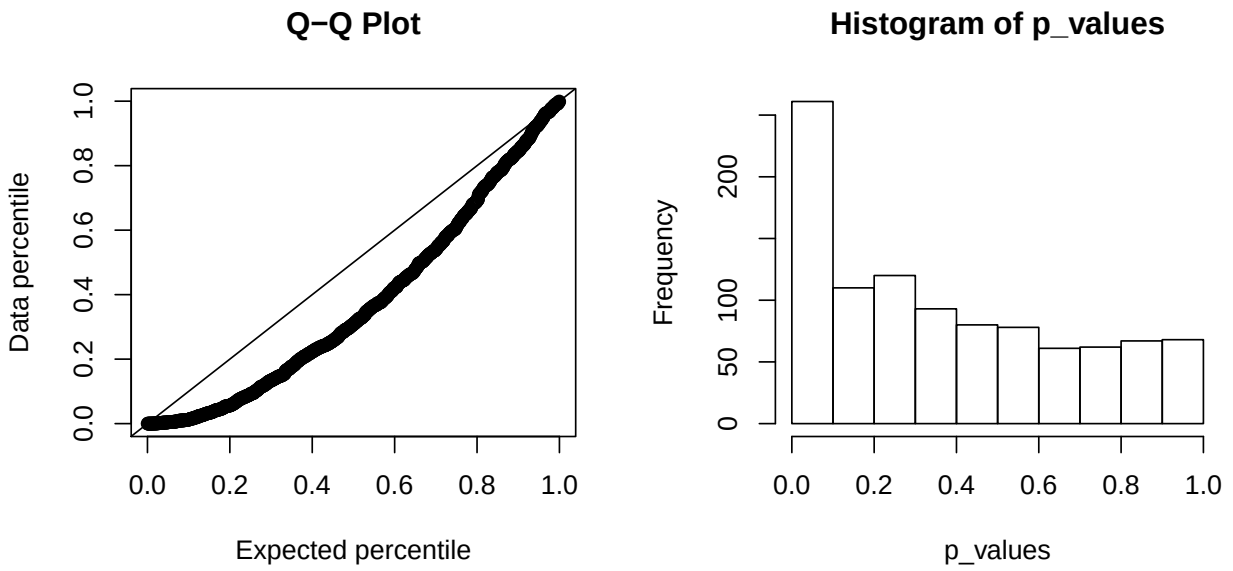
In this scenario $n2 > n1$, which means sample with lower variance has smaller sample size, and sample with higher variance has larger sample size. The QQ-plot shows that the distribution is not uniform, it's a left-skewed distribution. The true 0.01, 0.05 data quantiles are significantly larger than 0.01 and 0.05, which suggests that using 0.01 and 0.05 as cut values is too conservative.

R Code Chunk

```
n1 = 100
n2 = 20
mu1 = mu2 = 68
sigma1 = 3
sigma2 = 6
reps = 1000

p_values = sim_pval_ttest(reps, n1, n2, mu1, mu2, sigma1, sigma2)
par(mfrow = c(1, 2))
adam.qqplot(p_values, qunif)
hist(p_values)
quantile(p_values, c(0.01, 0.05))
```

```
##           1%           5%
## 0.0001202363 0.0040105301
```



In this scenario $n2 < n1$, which means sample with lower variance has larger sample size, and sample with higher variance has smaller sample size. The QQ-plot shows that the distribution is not uniform, it's a right-skewed distribution. The true 0.01, 0.05 data quantiles are significantly smaller than 0.01 and 0.05, which suggests that using 0.01 and 0.05 as cut values failed to protect the corresponding type I error.

Problem 2.e

R Code Chunk

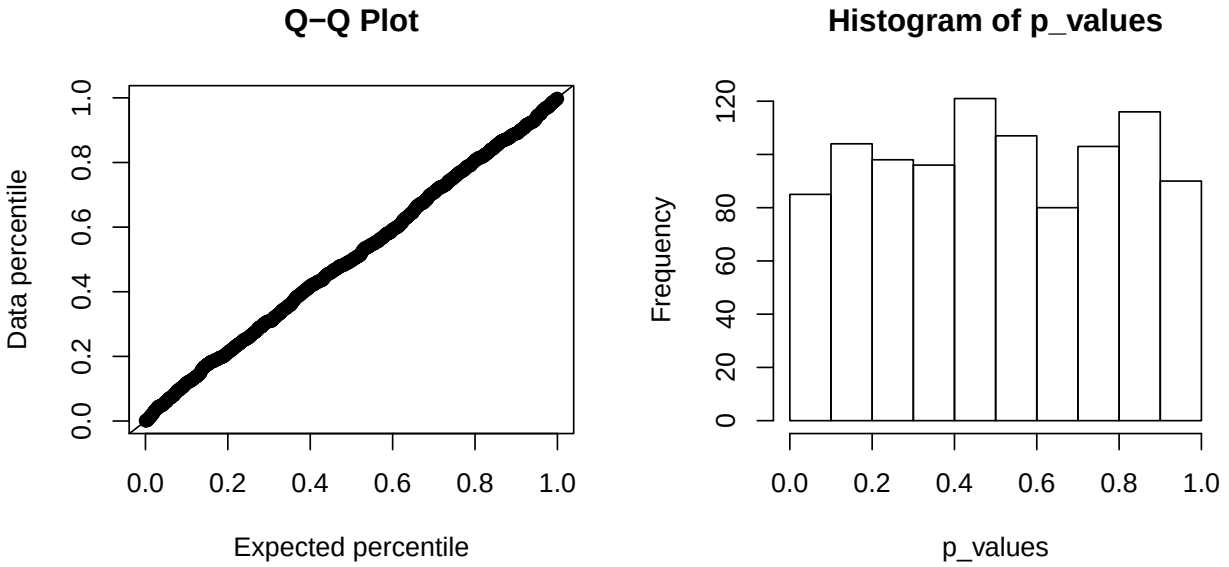
```
sim_pval_ttest_multinorm = function(reps, n, mu1, mu2, sigma, rho) {
    mu_vec = c(mu1, mu2)
    var_mat = matrix(0, 2, 2)
    var_mat[1, 1] = var_mat[2, 2] = sigma ^ 2
    var_mat[1, 2] = var_mat[2, 1] = sigma ^ 2 * rho
    p_values = numeric(reps)
    for (r in 1:reps) {
        data_xy = mvrnorm(n, mu_vec, var_mat)
        test_result = t.test(data_xy[, 1], data_xy[, 2], var.equal = TRUE)
        p_values[r] = test_result$p.value
    }
    p_values
}

n = 20
mu1 = mu2 = 68
sigma = 3
rho = 0.01
reps = 1000

p_values = sim_pval_ttest_multinorm(reps, n, mu1, mu2, sigma, rho)
par(mfrow = c(1, 2))
adam.qqplot(p_values, qunif)
hist(p_values)
```

```
quantile(p_values, c(0.01, 0.05))
```

```
##         1%         5%
## 0.01133713 0.05917701
```



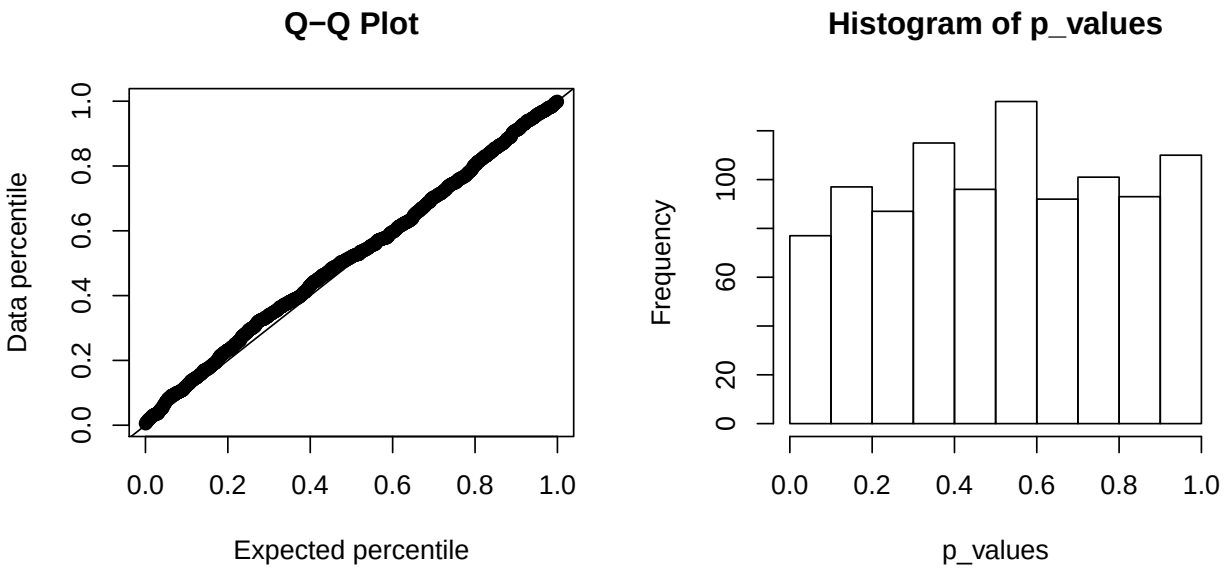
In this case $\rho = 0.01$, which suggests very low correlation among random variables. Then the QQ-plot shows that the distribution of resulting p-values is very close to uniform, and the true 0.01, 0.05 data quantiles are very close to 0.01 and 0.05, suggesting that low correlation among observations is not a problem for t-test.

R Code Chunk

```
n = 20
mu1 = mu2 = 68
sigma = 3
rho = 0.1
reps = 1000

p_values = sim_pval_ttest_multinorm(reps, n, mu1, mu2, sigma, rho)
par(mfrow = c(1, 2))
adam.qqplot(p_values, qunif)
hist(p_values)
quantile(p_values, c(0.01, 0.05))
```

```
##         1%         5%
## 0.01940264 0.07276316
```



In this case $\rho = 0.1$, which is still a relatively low correlation among random variables. Then the QQ-plot shows that the distribution of resulting p-values is very close to uniform. However, the true 0.01, 0.05 data quantiles are slightly larger than 0.01 and 0.05, which suggests that using 0.01 and 0.05 as cut values is conservative for this kind of scenario.

Problem 3.a

Proof.

$$l(x_i|\mu) = \log L(x_i|\mu) = -\log \mu - \frac{x}{\mu}$$
$$\Rightarrow l(x|\mu) = \sum_{i=1}^{n} l(x_i|\mu) = -n\log \mu - n\frac{\bar{x}}{\mu}$$

Then set

$$\frac{\partial}{\partial \mu} l(x|\mu) = 0$$
$$\Rightarrow -\frac{1}{\mu} + \frac{\bar{x}}{\mu^2} = 0$$
$$\Rightarrow \hat{\mu} = \bar{x}$$

Q.E.D.

Problem 3.b

$$\begin{aligned}
& L(a) = \mathrm{E}\{(a\bar{x} - \mu)^2\} \\
\Rightarrow \quad & L(a) = \mathrm{E}\{(a(\bar{x} - E(\bar{x})) + a\mathrm{E}(\bar{x}) - \mu)^2\} \\
\Rightarrow \quad & L(a) = \mathrm{E}\{a^2(\bar{x} - E(\bar{x}))^2\} + \mathrm{E}\{(a\mathrm{E}(\bar{x}) - mu)^2\} \\
\Rightarrow \quad & L(a) = a^2\mathrm{Var}(\bar{x}) + a^2\mu^2 - 2a\mu^2 + \mu^2 \\
\Rightarrow \quad & L(a) = a^2\frac{\mu^2}{n} + a^2\mu^2 - 2a\mu^2 + \mu^2
\end{aligned}$$

Then set

$$\begin{aligned}
& \frac{\partial}{\partial a} L(a) = 0 \\
\Rightarrow \quad & 2a\frac{\mu^2}{n} + 2a\mu^2 - 2\mu^2 = 0 \\
\Rightarrow \quad & \hat{a} = \frac{\mu^2}{\frac{\mu^2}{n} + \mu^2} = \frac{n}{n+1}
\end{aligned}$$

Problem 3.c

$$\mathrm{E}\{(\bar{x} - \mu)^2\} = \mathrm{Var}(\bar{x}) = \frac{\mu^2}{n}$$
$$\mathrm{E}\{(a\bar{x} - \mu)^2\} = a^2\mathrm{Var}(\bar{x}) + (a\mu - \mu)^2 = \frac{1}{n+1}\mu^2$$

The shrinkage estimator has the smaller mean-squared error.

Problem 3.d

R Code Chunk

```
#' simulate the mean square errors for likelihood estimator and shrinkage
#' estimator for the mean of a Exponential distribution
#'
#' @param reps number of replications for this simulation
#' @param n sample size
#' @param mu mean of the Exponential distribution
#'
#' @return a vector containing simulated mean square errors for the two estimators
#' (likelihood, shrinkage)
sim_diff_lkh_shk = function(reps, n, mu) {
    x = rexp(reps * n, 1 / mu)
    x = matrix(x, reps, n)
    a = n / (n + 1)
    erros = apply(x, 1, function(x, mu) {
        c((mean(x) - mu) ^ 2, (a * mean(x) - mu) ^ 2)
    }, mu)
    apply(erros, 1, mean)
}

n_vec = c(5, 10, 50)
mu_vec = c(0.5, 1, 10)
reps = 1000
result = matrix(0, length(n_vec) * length(mu_vec), 6)
colnames(result) = c("n", "mu", "sim_lkh_err", "sim_shk_est",
                     "true_lkh_err", "true_shk_est")
k = 1
for (n in n_vec)
    for (mu in mu_vec) {
        result[k, 1] = n
        result[k, 2] = mu
        result[k, c(3, 4)] = sim_diff_lkh_shk(reps, n, mu)
        result[k, 5] = mu ^ 2 / n
        result[k, 6] = mu ^ 2 / (n + 1)
        k = k + 1
    }
round(result, 3)
```

```
##        n   mu sim_lkh_err sim_shk_est true_lkh_err true_shk_est
##  [1,]  5  0.5       0.048       0.040        0.050        0.042
##  [2,]  5  1.0       0.224       0.174        0.200        0.167
##  [3,]  5 10.0      21.670      17.764       20.000       16.667
##  [4,] 10  0.5       0.025       0.023        0.025        0.023
##  [5,] 10  1.0       0.099       0.088        0.100        0.091
##  [6,] 10 10.0       9.764       9.016       10.000        9.091
##  [7,] 50  0.5       0.006       0.006        0.005        0.005
##  [8,] 50  1.0       0.020       0.020        0.020        0.020
##  [9,] 50 10.0       1.896       1.861        2.000        1.961
```

From the resulting table we could see that the simulated estimates are very close the results derived

in part 3c.