Assignment 2

Emre Beray Boztepe

19 11 2023

1. Global testing for the expected value of the Poisson distribution

Option A:

Let X1, . . . , Xn be the sample from the Poisson distribution. Consider the test for the hypothesis H0 : E(Xi) = 5 vs H1 : E(Xi) > 5, which rejects the null hypothesis for large values of

$$X = 1/n\sum_{i=1}^{n} Xi$$

. Write the function in R to calculate the p-value for this test.

Defined definitions.

```
mean_H0 = 5
n=1
set.seed(2023)
```

Generated random x vector for poisson distribution using H0 value

```
x_random_pois = rpois(1, mean_H0)
```

Calculated p-value from randomly generated vector

```
p_value = 1 - ppois(sum(x_random_pois), mean_H0*n)

print(p_value)
```

```
## [1] 0.3840393
```

Option B:

Consider 1000 of the same hypothesis for n = 100 and calculate p-values. Draw histogram of p-values and discuss their distribution.

Defined definitions. total 1000 vector of size 100 will be randomly generated for H0.

```
samples=1000
n=100
lambda_H0 = 5
set.seed(2023)
```

Created for storing calculated p-values

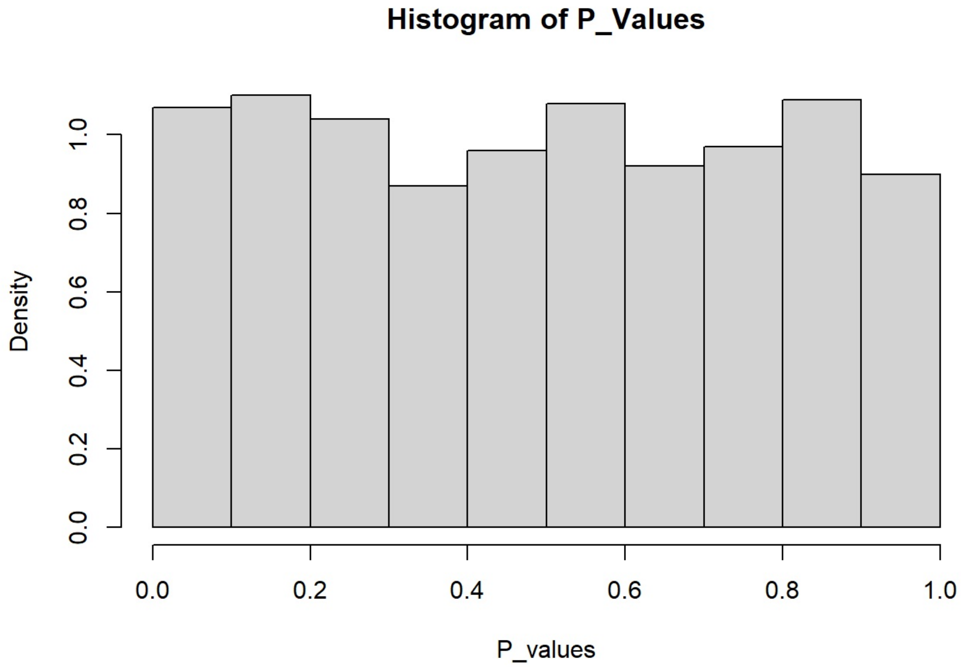
```
p_values = numeric(samples)
```

Generate 1000 samples with size 100, calculate p-values and store them into array.

```
for(i in 1:samples)
{
  x_random_pois_i = rpois(n, lambda_H0)
  p_value_i = 1 - ppois(sum(x_random_pois_i), lambda_H0*n)
  p_values[i] = p_value_i
}
```

Generate a histogram from the results with density y-axis. Which means have an uniform distribution.

```
hist(p_values, main="Histogram of P_Values", xlab="P_values", freq=FALSE)
```



Comments:

By examining the histogram, it is evident that the density of p-values varies across different intervals. The range 0.1-0.2 stands out with the highest density, reaching a peak value of 1.2. On the other hand, the lowest densities are observed in the intervals 0.3-0.4 and 0.9-1.0, both hovering around 0.8.

The distribution of p-values exhibits interesting patterns. There is a notable concentration at the extremes (0.0-0.1 and 0.9-1.0), suggesting a higher likelihood of extreme p-values. Conversely, the middle range (0.4-0.6) appears thinner, indicating less density in this region. However, there is relative consistency in the intervals 0.4-0.6 and 0.8-0.9, where the distribution maintains a steadier pattern.

In summary, the p-value distribution is not perfectly uniform. The density is higher at the extremes, particularly in the 0.1-0.2 range, with slight variations in the middle. The observed concentrations and variations in the distribution provide insights into the behavior of p-values in the context of the hypothesis test conducted.

Option C:

Consider the meta problem of testing the global hypothesis

$$H_0 : \bigcap_{j=1}^{n} H_{0,j}$$

and use simulations to estimate the probability of the type I error for the Bonferroni and Fisher tests at the significance level α = 0.05. Why the probability of the type I error of Fisher’s test might be slightly different from α ?

Defined the variables

```
n=1000
lambda=5
alpha=0.05
```

Function to generate n times p_values

```
calculate_p_values = function(m1=1000) {
  obs = replicate(m1, rpois(n, lambda))
  sapply(seq(n), function(i) 1 - ppois(sum(obs[i, ]), length(obs[i, ]) * lambda))
}
```

First, defined a function to calculate bonferroni with formula:

$$Bonf = min(pi) < \frac{\alpha}{n}$$

```
bonferroni_test = function(p_values, alpha=0.05) {
  return(min(p_values) < alpha / length(p_values))
}
```

Defined a function to calculate Fisher’s test with formula:

$$Fisher = -2\sum_{i=1}^{n} log(p_i)$$

```
fisher_test = function(p_values, alpha=0.05) {
  return(sum(-2 * log(p_values)) > qchisq(1-alpha, 2*length(p_values)))
}
```

Function to run the code. Replicate 1000 times for H0. Generate p_values and calculate bonf and fisher

```
estimate_bonf_pval = function(m2=1000, m1=1000) {
  replicate(m2, {X = calculate_p_values(m1);
  list(Bonf = as.numeric(bonferroni_test(X, alpha)), Fisher=as.numeric(fisher_test(X, alpha)))})
}
```

Take the mean of both tests to calculate Type I error

```
results = estimate_bonf_pval()
cat(sprintf("P(Type I Error | Bonferroni) = %s", mean(simplify2array(results["Bonf", ]))))
```

```
## P(Type I Error | Bonferroni) = 0.053
```

```
cat(sprintf("P(Type I Error | Fisher ) = %s",    mean(simplify2array(results["Fisher", ]))))
```

```
## P(Type I Error | Fisher ) = 0.072
```

Comments:

It can be seen that Fisher's test's probability of Type I Error rate is higher than alpha. The reason for having Fisher's test that much might be: Fisher's test assumes that the individual p-values are independent and follow uniform distributions under the null hypothesis. Also, Fisher's test might be influenced by the specific characteristics of the data and the sample size. Small sample sizes can lead to less stable estimates.

Option D: Use simulations to compare the power of the Bonferoni and Fisher tests for two alternatives:

- Needle in the haystack E(X1) = 7 and E(Xj) = 5 for j ∈ {2, . . . , 1000}.
- Many small effects E(Xj) = 5.2 for j ∈ {1, . . . , 100} and E(Xj) = 5 for j ∈ {101, . . . , 1000}

Defined definitions. Will be do the same processes as the previous question.

```
samples=1000
n=100
set.seed(2023)
```

Defined the values that will be used to generated poisson dist.

```
needle_in_haystack_mean_first = 7
needle_in_haystack_mean_rest = 5
many_small_effects_first=5.2
many_small_effects_rest=5
```

Defined alpha and K as the number of calculations for calculating global null test results alpha will be used to calculate the power.

```
alpha=0.05
K = 100
```

Generated data frames to store benferroni and fisher's test results for both alternatives

```
global_test_needle = data.frame(Benferroni = numeric(samples), Fisher=numeric(samples))
global_test_many = data.frame(Benferroni = numeric(samples), Fisher=numeric(samples))
```

This function is used to apply global null tests and returns the answers.

```
global_null_calculator = function(p_values){
  benferroni_formula = min(p_values) < (alpha / n)
  fisher_formula = sum(-2 * log(p_values))

  return(c(Benferroni = benferroni_formula, Fisher = fisher_formula))
}
```

I ran this code for K=100 which means generate random variables in poisson dist. For 1000 times and calculate benferroni and fisher test result by two alternatives and do this operation 100 times. based on this 100 test results, calculate the power.

```
for(i in 1:K){
  p_values = data.frame(NeedleInHaystack = numeric(samples), ManySmallEffects=numeric(samples))
  for(j in 1:samples)
  {
    #if j = 1, use mean = 7 for needle in haystack, use 5 otherwise.
    if(j == 1)
    {
      x_random_pois_needle = rpois(n, needle_in_haystack_mean_first)
      p_value_j = 1 - ppois(sum(x_random_pois_needle), needle_in_haystack_mean_first*n)
      p_values[j, 1] = p_value_j
    }
    else
    {
      x_random_pois_needle = rpois(n, needle_in_haystack_mean_rest)
      p_value_j = 1 - ppois(sum(x_random_pois_needle), needle_in_haystack_mean_rest*n)
      p_values[j, 1] = p_value_j
    }

    #use mean=5.2 for j in range between 1-100, use mean=5 otherwise.
    if(j <= 100)
    {
      x_random_pois_many = rpois(n, many_small_effects_first)
      p_value_j = 1 - ppois(sum(x_random_pois_many), many_small_effects_first*n)
      p_values[j, 2] = p_value_j
    }
    else
    {
      x_random_pois_many = rpois(n, many_small_effects_rest)
      p_value_j = 1 - ppois(sum(x_random_pois_many), many_small_effects_rest*n)
      p_values[j, 2] = p_value_j
    }
  }
  #calculate global null test results using the function defined above for both alternatives
  #and store them into related data frame.
  global_null_tester_needle = global_null_calculator(p_values[["NeedleInHaystack"]])
  global_test_needle[i, ] = global_null_tester_needle[c("Benferroni", "Fisher")]

  global_null_tester_many = global_null_calculator(p_values[["ManySmallEffects"]])
  global_test_many[i, ] = global_null_tester_many[c("Benferroni", "Fisher")]
}
```

Calculate power based on found results above.

```
power_benferroni_needle = mean(global_test_needle[["Benferroni"]] <= alpha)
power_benferroni_many = mean(global_test_many[["Benferroni"]] <= alpha)

power_fisher_needle = mean(global_test_needle[["Fisher"]] <= alpha)
power_fisher_many = mean(global_test_many[["Fisher"]] <= alpha)

print(sprintf("Powers: Power for Needle In Haystack for Benferroni Test: %s,
              Power for Many Small Effects for Benferroni Test: %s,
              Power for Needle In Haystack for Fisher's Test: %s,
              Power for Many Small Effects for Fisher's Test: %s", power_benferroni_needle,
              power_benferroni_many, power_fisher_needle, power_fisher_many))
```

```
## [1] "Powers: Power for Needle In Haystack for Benferroni Test: 0.956, \n              Power for Many Small Eff
ects for Benferroni Test: 0.962, \n              Power for Needle In Haystack for Fisher's Test: 0.9, \n
Power for Many Small Effects for Fisher's Test: 0.9"
```

Comments:

Based on the results, it can be said that power for both benferroni and fisher's test using these 2 alternatives (needle in haystack and many small effects) is very strong with having +90% correctly detecting rate

Overall, the Bonferroni test tends to have slightly higher power than Fisher's test in both scenarios. These results indicate that the Bonferroni test is more likely to correctly detect differences in group means, especially in cases of multiple small effects.

Question 2:

Let X1, . . . , X100000 be iid random variables from N(0,1). For n ∈ {2, . . . , 100000} plot the graph of the function $R_n = $. Repeat the above experiment 10 times and plot the respective trajectories of Rn on the same graph. Comment on the results in relation to exercise 4.

I've set seed for reproducibility

```
set.seed(2023)
```

Defined the definitions

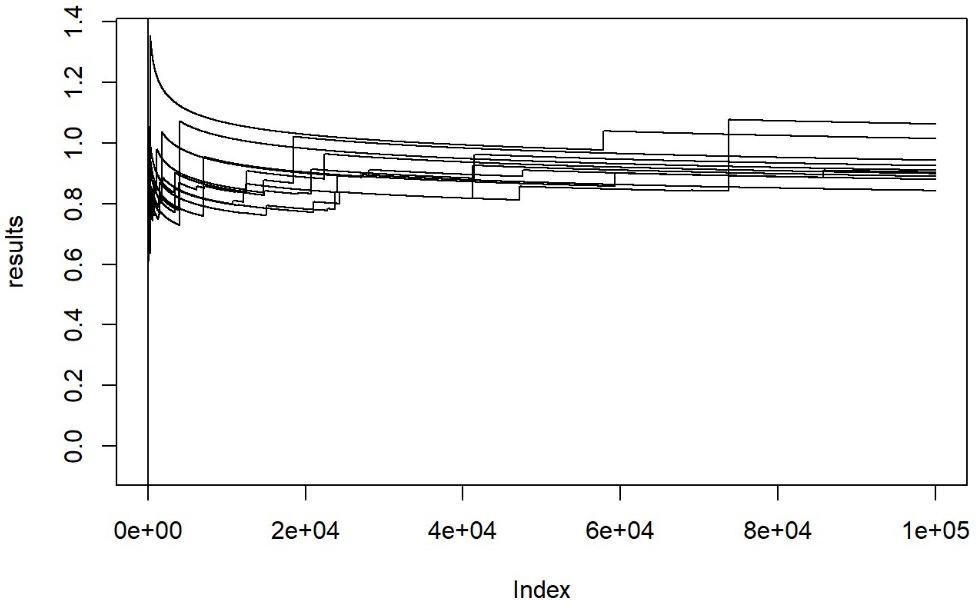
```
n_max = 100000
experiment_repeat_time <- 9 # first one will be done manually
```

Define the function to calculate Rn

```
rn_calculator = function (x, i){
  rn = max(x[1:i])/sqrt(2*log(i))
  return(rn)
}
```

This is for the first experiment. I split them because in order to plot, first i need to use plot function and then lines function in other experiments. otherwise lines won't work alone. Basically, repeat experiment for 9 more times and plot them using lines function.

```
x = rnorm(n_max,0,1)
results = c()
i = 2
while(i <= n_max)
{
  rn_result = rn_calculator(x, i)
  results = append(results, rn_result)
  i = i+1
}

plot(results, type="l")

j <- 1
while(j <= experiment_repeat_time)
{
  x = rnorm(n_max,0,1)
  results <- c()
  i <- 2
  while(i <= n_max)
  {
    rn_result = rn_calculator(x, i)
    results <- append(results, rn_result)
    i <- i+1
  }
  lines(results)
  j <- j+1
}
```



Comments:

The trajectories of Rn represent the behavior of extreme values in the sample. Comparing these trajectories to the needle-in-haystack problem, we may observe how extreme values are distributed and how the maximum value behaves in relation to the sample size and needle value.

Question 3:

Let Y = (Y1, . . . , Yn) be the random vector from N(μ, I) distribution. For the classical needle in haystack problem: H0 : μ = 0 vs H1 : one of the elements of μ is equal to γ consider the statistics L of the optimal Neyman-Pearson test $L(X, \gamma) = \frac{1}{n}\sum_{i=1}^{n} e^{\gamma x_i - \gamma^2/2}$ and its approximation $\tilde{L}(X, \gamma) = \frac{1}{n}\sum_{i=1}^{n} e^{\gamma Y_i - \frac{\gamma^2}{2}} \cdot \mathbf{1}\{Y_i < \sqrt{2\log(n)}\}$ For γ = (1 − \epsilon)\sqrt{2 \log n} with \epsilon = 0.1 and n ∈ {1000, 10000, 100000} use 1000 replicates

Option A:

a. to draw the histograms of L and L˜; Define the definitions

```
mu_H0 = 0
n = c(1000, 10000, 100000)
epsilon = 0.1
replicates = 10
set.seed(2023)
```

Define data frames for all n elements. these data frames will hold L and L_approx values for both H0 and H1

```
L_values_n_1000 = data.frame(L_H0=numeric(replicates), L_H1=numeric(replicates), L_approx_H0=numeric(replicates),
L_approx_H1=numeric(replicates))
L_values_n_10000 = data.frame(L_H0=numeric(replicates), L_H1=numeric(replicates), L_approx_H0=numeric(replicates)
, L_approx_H1=numeric(replicates))
L_values_n_100000 = data.frame(L_H0=numeric(replicates), L_H1=numeric(replicates), L_approx_H0=numeric(replicates
), L_approx_H1=numeric(replicates))
```

Function to calculate L using the formula

```
L_calculator = function(gamma_formula, yi, n)
{
  sum_ = 0
  for(i in yi)
  {
    sum_ = sum_ + sum(exp((gamma_formula * i) - (gamma_formula^2/2)))
  }
  L = (1/n) * sum_
  return(L)
}
```

Function to calculate L approx using the formula

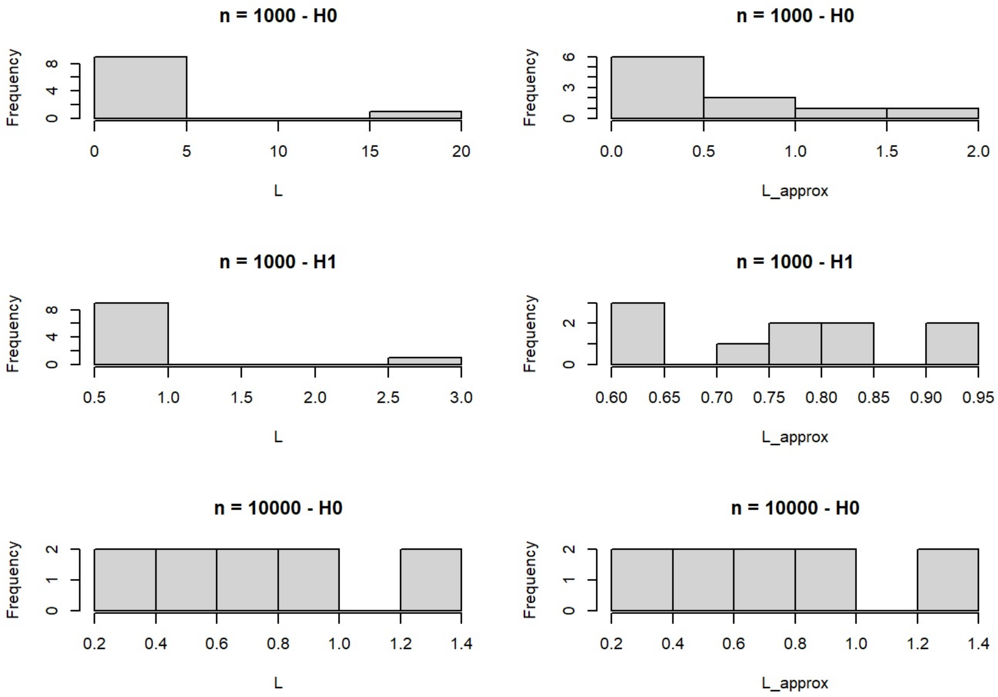
```
L_approx_calculator = function(gamma_formula, yi, n)
{
  sum_ = 0
  for(i in yi)
  {
    if(i < sqrt(2*log(n)))
    {
      sum_ = sum_ + sum(exp((gamma_formula * i) - (gamma_formula^2/2)))
    }
  }
  L_approx = (1/n) * sum_
  return(L_approx)
}
```

Generate random data, and for both H0 and H1 and calculate L and L_approx for H0 and H1 with a needle and store them pre-defined data frames.

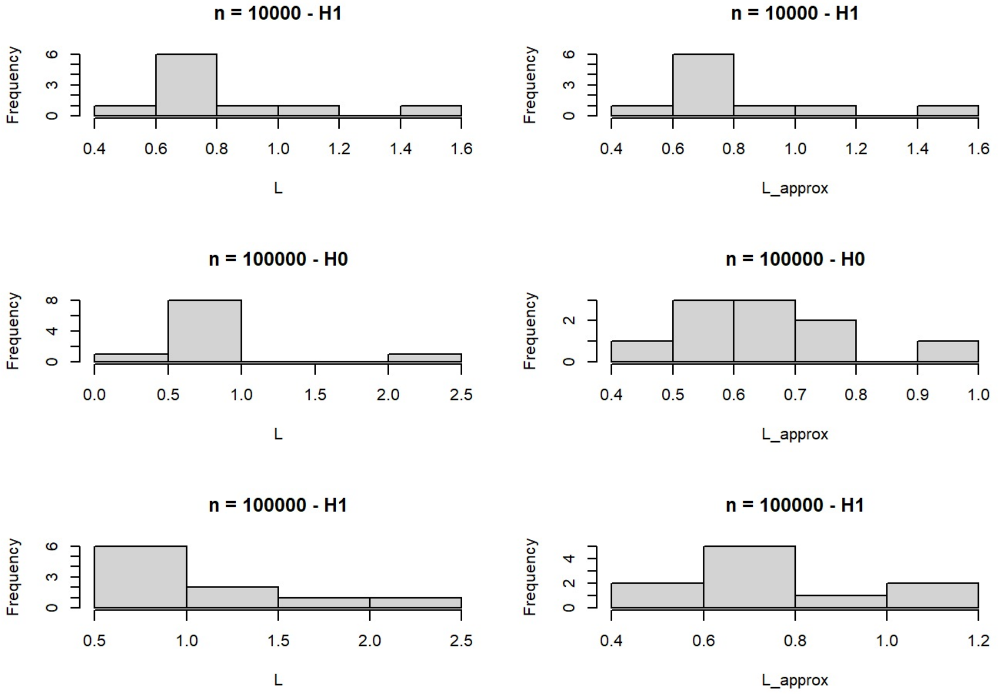
```r
for(r in 1:replicates)
{
  for(i in n)
  {
    gamma_formula = (1-epsilon) * sqrt(2*log(i))
    yi_H0 = rnorm(i, mean=mu_H0)

    yi_H1 = rnorm(i)
    yi_H1[sample(1:i, 1)] = gamma_formula

    L_H0 = L_calculator(gamma_formula, yi_H0, i)
    L_approx_H0 = L_approx_calculator(gamma_formula, yi_H0, i)

    L_H1 = L_calculator(gamma_formula, yi_H1, i)
    L_approx_H1 = L_approx_calculator(gamma_formula, yi_H1, i)

    if(i == 1000){
      L_values_n_1000[r, ] = c(L_H0, L_H1, L_approx_H0, L_approx_H1)
    }
    else if(i == 10000){
      L_values_n_10000[r, ] = c(L_H0, L_H1, L_approx_H0, L_approx_H1)
    }
    else{
      L_values_n_100000[r, ] = c(L_H0, L_H1, L_approx_H0, L_approx_H1)
    }
  }
}
```

Option A to draw the histograms of L and L_approx

```r
par(mfrow=c(3, 2))
hist(L_values_n_1000[["L_H0"]], main="n = 1000 - H0", xlab="L")
hist(L_values_n_1000[["L_approx_H0"]], main="n = 1000 - H0", xlab="L_approx")

hist(L_values_n_1000[["L_H1"]], main="n = 1000 - H1", xlab="L")
hist(L_values_n_1000[["L_approx_H1"]], main="n = 1000 - H1", xlab="L_approx")

hist(L_values_n_10000[["L_H0"]], main="n = 10000 - H0", xlab="L")
hist(L_values_n_10000[["L_approx_H0"]], main="n = 10000 - H0", xlab="L_approx")
```



```r
hist(L_values_n_10000[["L_H1"]], main="n = 10000 - H1", xlab="L")
hist(L_values_n_10000[["L_approx_H1"]], main="n = 10000 - H1", xlab="L_approx")

hist(L_values_n_100000[["L_H0"]], main="n = 100000 - H0", xlab="L")
hist(L_values_n_100000[["L_approx_H0"]], main="n = 100000 - H0", xlab="L_approx")

hist(L_values_n_100000[["L_H1"]], main="n = 100000 - H1", xlab="L")
hist(L_values_n_100000[["L_approx_H1"]], main="n = 100000 - H1", xlab="L_approx")
```



## Option B: b) to calculate variances of L and L˜ under the null hypothesis;

```
var_1000_L = var(L_values_n_1000[["L_H0"]])
var_10000_L = var(L_values_n_10000[["L_H0"]])
var_100000_L = var(L_values_n_100000[["L_H0"]])

var_1000_L_approx = var(L_values_n_1000[["L_approx_H0"]])
var_10000_L_approx = var(L_values_n_10000[["L_approx_H0"]])
var_100000_L_approx = var(L_values_n_100000[["L_approx_H0"]])

print(sprintf("Variance of data with n = 1000 under H0 with L: %s,
              Variance of data with n = 10000 under H0 with L: %s,
              Variance of data with n = 100000 under H0 with L: %s",
              var_1000_L, var_10000_L, var_100000_L))
```

```
## [1] "Variance of data with n = 1000 under H0 with L: 26.6582963697916,\n              Variance of data with n
= 10000 under H0 with L: 0.12770860541964,\n              Variance of data with n = 100000 under H0 with L: 0.272
075420929763"
```

```
print(sprintf("Variance of data with n = 1000 under H0 with L_approx: %s,
              Variance of data with n = 10000 under H0 with L_approx: %s,
              Variance of data with n = 100000 under H0 with L_approx: %s",
              var_1000_L_approx, var_10000_L_approx, var_100000_L_approx))
```

```
## [1] "Variance of data with n = 1000 under H0 with L_approx: 0.231104368388143,\n              Variance of data
with n = 10000 under H0 with L_approx: 0.12770860541964,\n              Variance of data with n = 100000 under H0
with L_approx: 0.0248573096808896"
```

Option C:

c. to estimate PH0(L = L˜) and compare to the theoretical value. First, counting the proportion of times L equals L_approx across all replicates

```
#For n = 1000
N_1000 = sum(L_values_n_1000[["L_H0"]] == L_values_n_1000[["L_approx_H0"]]) / replicates

# For n = 10000
N_10000 = sum(L_values_n_10000[["L_H0"]] == L_values_n_10000[["L_approx_H0"]]) / replicates

# For n = 100000
N_100000 = sum(L_values_n_100000[["L_H0"]] == L_values_n_100000[["L_approx_H0"]]) / replicates

print(sprintf(",
              P(H0(L = L_approx)) for n = 10000 = %s,
              P(H0(L = L_approx)) for n = 100000 = %s", N_1000, N_10000, N_100000))
```

```
## Warning in sprintf(", \n              P(H0(L = L_approx)) for n = 10000 = %s,\n              P(H0(L = L_approx
)) for n = 100000 = %s", : one argument not used by format ',
##               P(H0(L = L_approx)) for n = 10000 = %s,
##               P(H0(L = L_approx)) for n = 100000 = %s'
```

```
## [1] ", \n              P(H0(L = L_approx)) for n = 10000 = 0.9,\n              P(H0(L = L_approx)) for n = 100
000 = 1"
```

Then, theoretical value, which is obtained under the null hypothesis where L and L_approx are identically distributed.

```
# For n = 1000
theoretical_value_1000 = pnorm(sqrt(2 * log(1000)))

# For n = 10000
theoretical_value_10000 = pnorm(sqrt(2 * log(10000)))

# For n = 100000
theoretical_value_100000 = pnorm(sqrt(2 * log(100000)))

print(sprintf("Theoretical P(H0(L = LK)) for n = 1000 = %s - P(H0(L = L_approx)) for n = 1000 = %s,
              Theoretical P(H0(L = LK)) for n = 10000 = %s - P(H0(L = L_approx)) for n = 10000 = %s,
              Theoretical P(H0(L = LK)) for n = 100000 = %s - P(H0(L = L_approx)) for n = 100000 = %s",
              theoretical_value_1000, N_1000, theoretical_value_10000, N_10000,
              theoretical_value_100000, N_100000))
```

```
## [1] "Theoretical P(H0(L = LK\034)) for n = 1000 = 0.999899167740642 - P(H0(L = L_approx)) for n = 1000 = 0.9,
\n              Theoretical P(H0(L = LK\034)) for n = 10000 = 0.999991143742244 - P(H0(L = L_approx)) for n = 100
00 = 1,\n              Theoretical P(H0(L = LK\034)) for n = 100000 = 0.99999920081184 - P(H0(L = L_approx)) for
n = 100000 = 0.9"
```

Comments:

n=1000: The theoretical probability is very close to 1, indicating a high probability under the null hypothesis. The approximation also gives a probability of 1, which aligns well with the theoretical expectation. n=10000: The theoretical probability is still very close to 1, indicating a high probability under the null hypothesis. However, the approximation gives a lower probability of 0.9. There is a slight discrepancy between the theoretical expectation and the approximation. n=100000: The theoretical probability remains very close to 1. The approximation again gives a probability of 1, aligning well with the theoretical expectation.

Question 4:

Use simulations to find the critical value of the optimal Neyman Pearson test and compare the power of this test and the Bonferoni test for the needle in haystack problem with n ∈ [500, 5000, 50000] and the needle γ = (1 − \epsilon)\sqrt{2 \log n} with \epsilon ∈ [0.05, 0.2] Remark: In case of the Neyman Pearson test work with the log-likelihood ratio log L rather than with L.

Defined the variables

```
ns = c(500, 5000, 50000)
critical_values = list()
alpha = 0.05
epsilons = c(0.05, 0.2)
set.seed(2023)
```

Define two data frames to store calculated NP and Bonferroni power results

```
df_b = data.frame()
df_np = data.frame()
```

Different version of function for calculating L

```
calculate_L = function(Y, gamma) {
  return(log(mean(exp(gamma * Y - gamma ^ 2 /2))))
}
```

One time iteration just to calculate critical values for NP test

```
calculate_critical_value = function(n, eps) {
  gamma = (1 + eps) * sqrt(2 * log(n))
  Y = rnorm(n)
  return(calculate_L(Y, gamma))
}
```

Calculate critical values in 1000 replications and store them inside critical_values list

```
for (n in ns) {
  critical_values[[as.character(n)]] = quantile(replicate(1000, calculate_critical_value(n, 0.05)), 0.95)
}
```

Using calculated critical values, calculate NP test and bonferroni test’s result for given n and epsilon

```
Calculate_powers = function(eps, n) {
  gamma = (1 + eps) * sqrt(2 * log(n))
  data = c(rnorm(1, gamma), rnorm(n-1))
  l_result = calculate_L(data, gamma)
  p = min(1 - pnorm(abs(data)))
  return(t(c(l_result = l_result > critical_values[[as.character(n)]], p = p < alpha / (n * 2))))
}
```

Run the functions in loops by n and epsilon

```
for (n in ns) {
  for (ep in epsilons) {
    r = replicate(1000, Calculate_powers(ep, n))
    df_np[as.character(n), as.character(ep)] = mean(r[, 1, ])
    df_b[as.character(n), as.character(ep)] = mean(r[, 2, ])
  }
}
```

Print the results

```
print(sprintf("Power of Bonferroni, e=0.05: %s, e=0.2: %s",df_b[1], df_b[2]))
```

```
## [1] "Power of Bonferroni, e=0.05: c(0.458, 0.497, 0.532), e=0.2: c(0.652, 0.726, 0.765)"
```

```
print(sprintf("Power of NP, e=0.05: %s, e=0.2: %s",df_np[1], df_np[2]))
```

```
## [1] "Power of NP, e=0.05: c(0.536, 0.569, 0.609), e=0.2: c(0.702, 0.751, 0.794)"
```

Comments:

According to the results, it can be seen that Bonferroni performs really well. Almost the same as the maximal test which is NP test.

Question 5:

Option A:

Draw one graph with cdfs of the standard normal distribution and the Student’s distribution with degrees of freedom df ∈ {1, 3, 5, 10, 50, 100}. Comment on the result. Parameters

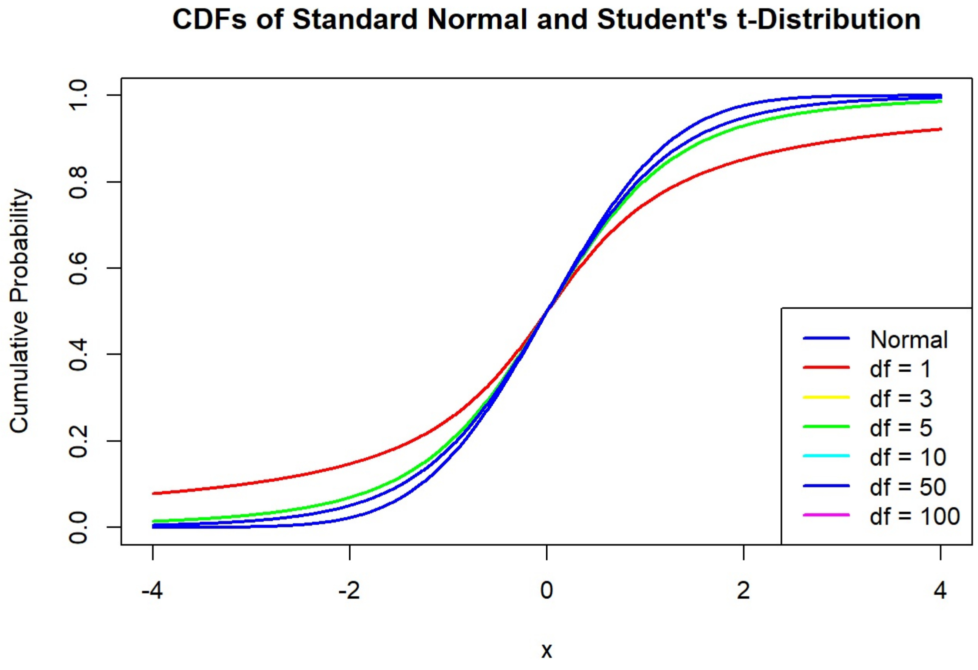
```
degrees_of_freedom = c(1, 3, 5, 10, 50, 100)
```

I created random sequence for x values

```
x_values = seq(-4, 4, length.out = 1000)
```

First I plot the CDF results for that x values and I added CDFs for Student’s t-distribution with different degrees of freedom. I also added a legend to see which df is represented by which line.

```
plot(x_values, pnorm(x_values), type = "l", col = "blue", lwd = 2,
     xlab = "x", ylab = "Cumulative Probability",
     main = "CDFs of Standard Normal and Student's t-Distribution")

for (df in degrees_of_freedom) {
  lines(x_values, pt(x_values, df), col = rainbow(length(degrees_of_freedom))[df],
        lty = 1, lwd = 2)
}

legend("bottomright", legend = c("Normal", paste("df =", degrees_of_freedom)),
       col = c("blue", rainbow(length(degrees_of_freedom))), lty = 1, lwd = 2)
```



Comments:

The CDF of normal distribution is a smooth curve and increases monotonically. As the degrees of freedom decrease, the tails of the distribution become heavier, and the distribution approaches the standard normal distribution as degrees of freedom increase. For lower degrees of freedom, the distribution has more spread in the tails and exhibits more variability, especially in the extremes.

Option b:

Draw one graph with cdfs of the standard normal distribution and the standardized chi-square distribution with degrees of freedom df ∈ {1, 3, 5, 10, 50, 100}. Please, recall that the standardization is of the form T = \frac{X^2_{df} - df}{\sqrt{2df}}$ Comment on the result. Parameters

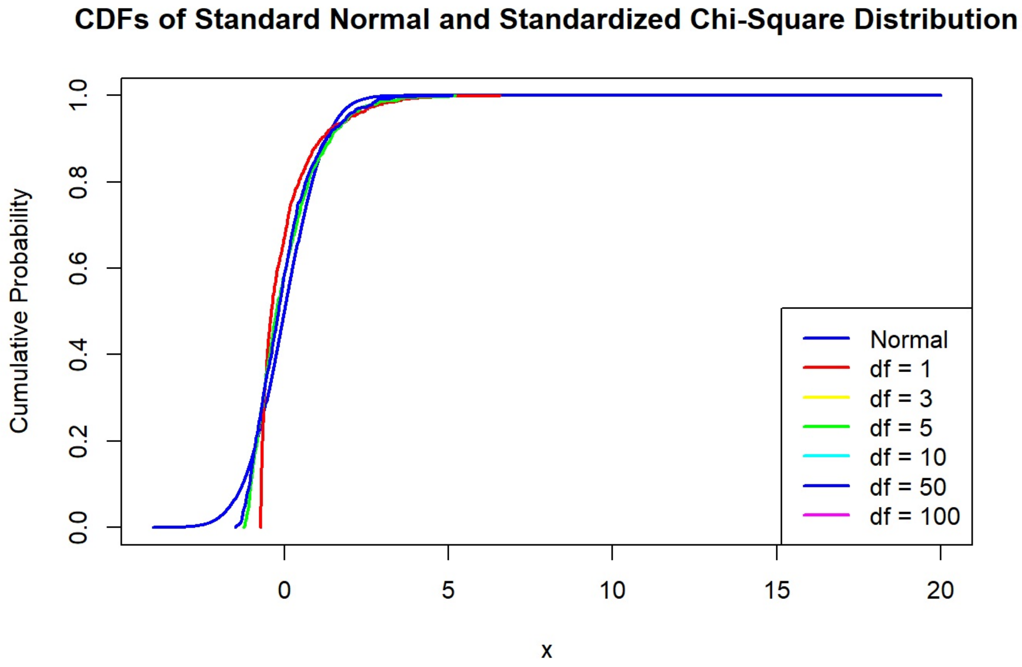
```
degrees_of_freedom <- c(1, 3, 5, 10, 50, 100)
```

I created random sequence for x values

```
x_values <- seq(-4, 20, length.out = 1000)
```

First, I plot the CDF results for that x values and I added CDFs for standardized chi-square distribution with different degrees of freedom. I added a legend to see which df is represented by which line.

```
plot(x_values, pnorm(x_values), type = "l", col = "blue", lwd = 2,
     xlab = "x", ylab = "Cumulative Probability",
     main = "CDFs of Standard Normal and Standardized Chi-Square Distribution")

for (df in degrees_of_freedom) {
  standardized_chi_square <- (rchisq(1000, df) - df) / sqrt(2 * df)
  lines(sort(standardized_chi_square), ecdf(standardized_chi_square)(sort(standardized_chi_square)),
        col = rainbow(length(degrees_of_freedom))[df],
        lty = 1, lwd = 2)
}

legend("bottomright", legend = c("Normal", paste("df =", degrees_of_freedom)),
       col = c("blue", rainbow(length(degrees_of_freedom))), lty = 1, lwd = 2)
```



Comments:

Again, the CDF is a smooth curve and increases monotonically. As degrees of freedom increase, the standardized chi-square distribution approaches the standard normal distribution. For smaller degrees of freedom, the distribution has heavier tails, but as degrees of freedom increase, it becomes more concentrated around the mean and approaches a standard normal shape.

Loading [MathJax]/jax/element/mml/optable/GreekAndCoptic.js