Homework 03 - STAT440

Joseph Sepich (jps6444)

09/11/2020

```
set.seed(42)
```

Problem 1

In this problem we will use simulation to explore the accuracy of confidence intervals. Consider a sample of n i.i.d. normal random variables with mean μ and standard deviation σ. Assume μ is unknown, but σ is known.

Part a

Generate such a sample $\mu = 1$ and $\sigma = 0.5$ for the cases $n = \{10, 100, 1000\}$. In reality this step would be done by nature, but here we can simulate it ourselves.

```
n <- c(10, 100, 1000)
samples <- vector(mode = "list", length = length(n))

mu <- 1
sigma <- 0.5

for (i in 1:length(n)) {
  samples[[i]] <- rnorm(n[i], mean = mu, sd = sigma)
}
```

Part b

Build a 95% confidence interval for each of these samples. Remember σ is assumed to be known but not μ.

Recall that when σ is known we can create our confidence interval for μ by using a transformation to the standard normal variable resulting in:

$$[\overline{X} \pm z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}]$$

Here we use $\alpha = 0.05$, so $\frac{\alpha}{2} = 0.025$. $z_{0.025} = 1.96$.

```
z_multi <- 1.96
intervals <- vector("list", length(n))
```

```
for (i in 1:length(n)) {
  x_bar <- mean(samples[[i]])
  diff <- z_multi * sigma / sqrt(n[i])
  intervals[[i]] <- c(x_bar - diff, x_bar + diff)
}
```

Part c

```
num <- 1000
n <- 100
diff <- z_multi * sigma / sqrt(n)
count_in <- 0

for (i in 1:num) {
  sample <- rnorm(n, mean = mu, sd = sigma)
  x_bar <- mean(sample)
  if(x_bar - diff < mu & x_bar + diff > mu) {
    count_in <- count_in + 1
  }
}
print(count_in)
```

```
## [1] 952
```

Out of 1000 simulations of sampling 100 values from the normal distribution with $\mu = 1$ and $\sigma = 0.5$, we get μ in our 95% confidence interval 952 times, or 95.2% of the time. This makes perfect sense from the definition of a confidence interval, which states that for a given confidence level, the level represents the frequency with which a confidence interval at that level should contain the given parameter. In plain English this means that for a confidence interval at x% confidence, we should expect x% of the confidence intervals constructed at that level to contain the parameter.

Part d

If the parameter σ was not known, then we would have to use the student's t distribution to calculate the confidence interval.

Problem 2

For this problem we will use the matrix D from the previous homework.

```
mat_v <- matrix(c(1, 2, 3, 2, 1, 0, 4, -4, 1, -1, 1, -1, 0, 3, 5), nrow = 3, ncol=5)

l2_norm <- function(vec) {
  sqrt(sum(vec^2))
}

num_cols <- dim(mat_v)[2]
mat_d <- matrix(1:25, nrow = num_cols, ncol = num_cols)
```

```r
for (i in 1:num_cols) {
  for (j in 1:num_cols) {
    mat_d[i, j] <- l2_norm(mat_v[,i] - mat_v[,j])
  }
}
mat_d
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 0.000000 3.316625 7.000000 4.582576 2.449490
## [2,] 3.316625 0.000000 5.477226 3.162278 5.744563
## [3,] 7.000000 5.477226 0.000000 7.348469 9.000000
## [4,] 4.582576 3.162278 7.348469 0.000000 6.403124
## [5,] 2.449490 5.744563 9.000000 6.403124 0.000000
```

Part a

Create a matrix Σ whose ij^{th} element is $exp(-\tau D_{ij}^2)$ for $\tau = \frac{1}{20}$.

```r
tau <- 1 / 20
mat_sig <- matrix(1:25, nrow = num_cols, ncol = num_cols)
mat_d_square <- mat_d %*% mat_d

for (i in 1:num_cols) {
  for (j in 1:num_cols) {
    mat_sig[i, j] <- exp(-1 * tau * mat_d_square[i, j])
  }
}
mat_sig
```

```
##              [,1]          [,2]         [,3]         [,4]          [,5]
## [1,] 0.012906813 0.035254569 2.486457e-02 0.020638686 0.0038113645
## [2,] 0.035254569 0.014995577 7.388970e-03 0.009936377 0.0205812201
## [3,] 0.024864572 0.007388970 2.254494e-05 0.004741827 0.0083692354
## [4,] 0.020638686 0.009936377 4.741827e-03 0.001836305 0.0084266593
## [5,] 0.003811364 0.020581220 8.369235e-03 0.008426659 0.0003191019
```
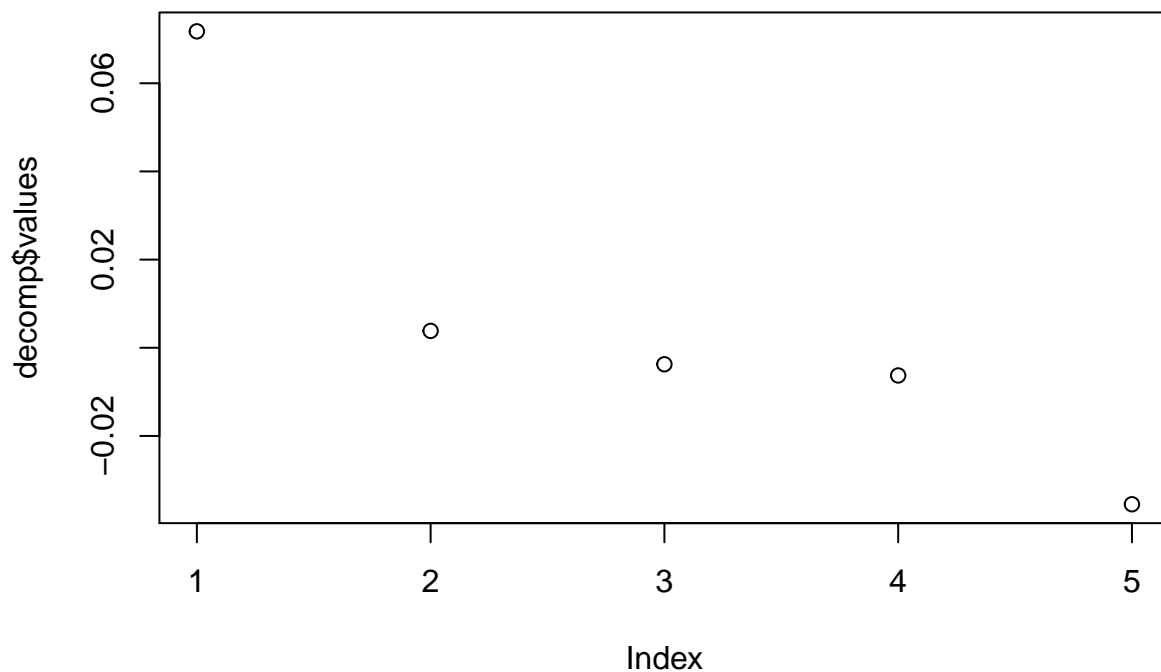
Part b

Compute the eigen-decomposition of Σ and plot the eigen values (sorted from largest to smallest).

```r
decomp <- eigen(mat_sig)

plot(decomp$values)
```

Part c

```r
frob_norm <- function(mat_input) {
  decomp <- eigen(mat_input)
  return(sqrt(sum(decomp$values^2)))
}
```

```r
epsilon <- 0.5

# the eigen values are the diagonal entries of lambda
for (l in 1:length(decomp$values)) {
  diag_entries <- numeric(length(decomp$values))
  diag_entries[1:l] <- decomp$values[1:l]
  sig_l <- decomp$vectors %*% diag(diag_entries) %*% t(decomp$vectors)
  mat_diff <- mat_sig - sig_l
  err_val <- frob_norm(mat_diff) / frob_norm(mat_sig)
  print(err_val)
}
```

```
## [1] 0.452698
## [1] 0.4501939
## [1] 0.4477838
## [1] 0.440927
## [1] 3.144088e-15
```

The minimum l, so that $\frac{||\Sigma - \Sigma^l||_F}{||\Sigma||_F} < \epsilon$ is just 1. This would imply that a majority of the variance/information contained within the original matrix Σ is stored within the first eigen value/vector. Basically when you start with l = 1, you will receive a matrix that is very close to the values in Σ, and as you increase l, the matrix becomes closer to Σ itself, but the larger values contribute more information.

Part d

If $l << L$, then the using M^l would require much less memory and involve less computation time, because you would only need to store the first l eigen vectors/eigen values to calculate the matrix. This is the entire idea behind principal component analysis and dimension reduction. If our error from the Frobenius norm is small enough this replacement is justified, because we will still maintain a majority of the information/variance from the original matrix.

Problem 3

Consider the density:

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2}$$

Part a

To show that this is a valid density we must recall the two properties that make this a valid density.

1. $f(x) \geq 0$
2. $\int_0^\infty f(x)dx = 1$

We are already given that the first property is satisfied, since $x \geq 0$, the density also has to be. The numerator of the fraction coefficient is x and the other place it apears is as an exponent, which negative values will merely give a number less than one, but greater than zero.

For the second property let's take the integral:

$$\int_0^\infty f(x)dx = \int_0^\infty \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} dx$$

To do this let's use u substitution where $u = \frac{-x^2}{2\sigma^2}; du = \frac{-x}{\sigma^2}dx \rightarrow dx = -\frac{-\sigma^2}{x}du$

$$\int_{-\infty}^0 e^u du = (e^u)|_{-\infty}^0 = 1 - 0 = 1$$

We have now satisfied both properties for a valid probability density function.

Part b

Compute the CDF and its inverse. To compute the CDF of a density function, we merely integrate the function from its start to some value a:

$$\int_0^a \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} dx$$

We can use the same u sub where $u = \frac{-x^2}{2\sigma^2}; du = \frac{-x}{\sigma^2}dx \rightarrow dx = -\frac{-\sigma^2}{x}du$.

$$\int_{-\infty}^{\frac{-a^2}{2\sigma^2}} e^u du = (e^u)|_{-\infty}^{\frac{-a^2}{2\sigma^2}} = e^{\frac{-a^2}{2\sigma^2}} - 0 = e^{\frac{-a^2}{2\sigma^2}}$$

$$F_x(x) = e^{\frac{-x^2}{2\sigma^2}}$$

Now let's calculate the inverse.

$$y = e^{\frac{-x^2}{2\sigma^2}}$$

$$ln(y) = \frac{-x^2}{2\sigma^2}$$

$$2ln(y)\sigma^2 = -x^2$$

$$-2ln(y)\sigma^2 = x^2$$

$$\sqrt{-2ln(y)\sigma^2} = x$$

This gives us our inverse CDF:

$$F_x^{-1}(y) = \sqrt{-2ln(y)\sigma^2}$$

Problem 4

Simulate 100 i.i.d. random variables from the density f of the previous problem using the inverse-CDF method. Assume $\sigma^2 = 2$. Plot a histogram of the resulting realizations together with a curve of the density function f(z).

```
inverse_cdf <- function(x, sigma) {
  sqrt(-2 * sigma^2 * log(x))
}


f_dens <- function(z, sigma) {
  (z / sigma^2) * exp(-1 * z^2 / (2 * sigma^2))
}


n <- 1000
sigma <- sqrt(2)
samples <- runif(n)
samples <- inverse_cdf(samples, sigma)
hist(samples, freq=FALSE)
x <- seq(0, 10, 0.1)
y <- f_dens(x, sigma)
lines(x,y)
```

Histogram of samples
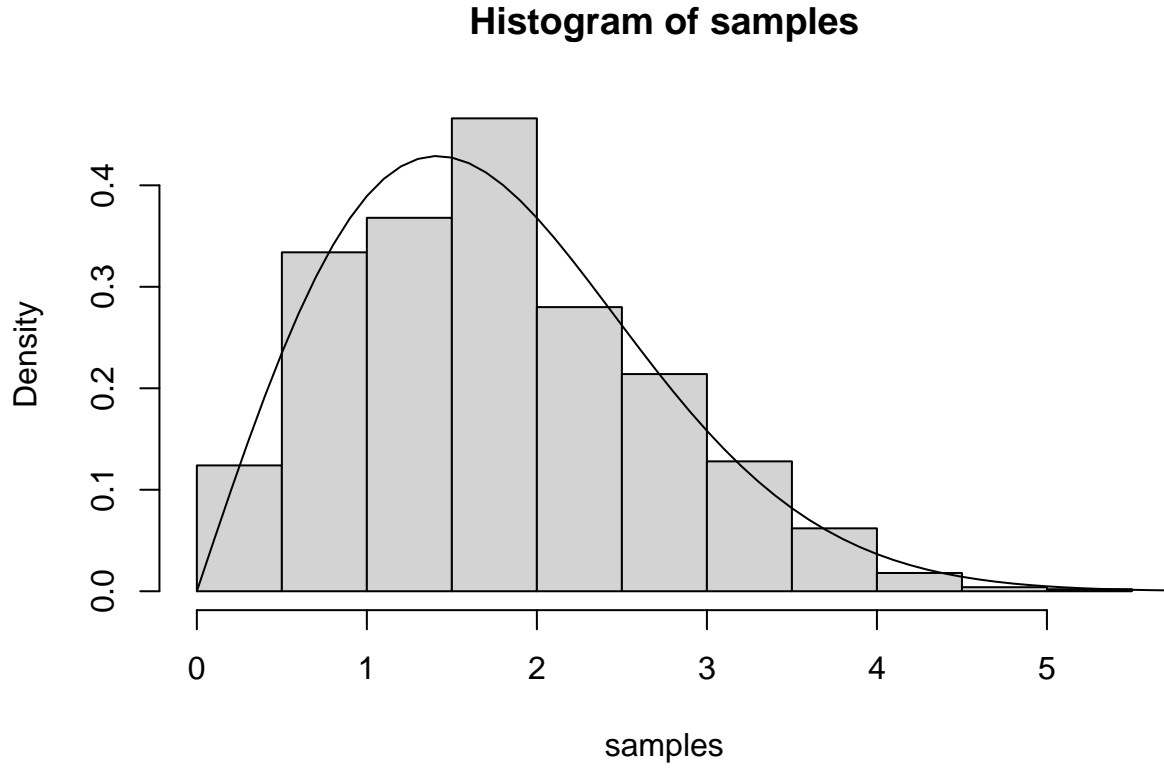
Problem 5

Part a

We must find a value c such that $\frac{f(x)}{g(x)} \leq c < \infty$ where f is the density from the previous two problems, and g is an exponential random variables with parameter λ. Recall the pdf of an exponential R.V:

$$g(x) = \lambda e^{-\lambda x}$$

Let's now define $h(x) = \frac{f(x)}{g(x)}$.

$$h(x) = \frac{f(x)}{g(x)} = \frac{\frac{x}{\sigma^2} e^{-x^2/2\sigma^2}}{\lambda e^{-\lambda x}} = \frac{x}{\lambda \sigma^2} e^{\lambda x - x^2/2\sigma^2}$$

The support here ranges from 0 to ∞, so our value c, in order to minimize its value, must be the max of $h(x)$ between 0 and ∞. To find the max we can take the first derivative:

$$h'(x) = \frac{1}{\lambda \sigma^2} e^{\lambda x - x^2/2\sigma^2} + (\lambda - \frac{x}{\sigma^2}) \frac{x}{\lambda \sigma^2} e^{\lambda x - x^2/2\sigma^2}$$

$$0 = \frac{1}{\lambda \sigma^2} e^{\lambda x - x^2/2\sigma^2} + (\lambda - \frac{x}{\sigma^2}) \frac{x}{\lambda \sigma^2} e^{\lambda x - x^2/2\sigma^2}$$

$$0 = 1 + (\lambda - \frac{x}{\sigma^2})x$$

$$\frac{x^2}{\sigma^2} - \lambda x - 1 = 0$$

We can use the quadratic formula to solve:

$$x = \frac{\sigma(\sigma\lambda + \sqrt{\sigma^2\lambda^2 + 4})}{2}$$

We can plug this value of x back in to get our value of c:

$$c = h(\frac{\sigma(\sigma\lambda + \sqrt{\sigma^2\lambda^2 + 4})}{2})$$

Part b

```r
n <- 1000
counter <- 0
found_count <- 0
found_samps <- numeric(n)

lambda <- 1
sigma <- sqrt(2)

h_dens <- function(x) {
  return(f_dens(x, sigma) / dexp(x, lambda))
}
c <- h_dens((sigma * (sigma * lambda + sqrt(sigma^2 * lambda^2 + 4))) / 2)
print(c)
```

```
## [1] 3.247647
```

```r
while (found_count < n) {
 g_samp <- rexp(1)
 g_yval <- runif(1, 0, c *dexp(g_samp))
 if (g_yval <= f_dens(g_samp, sigma)) {
   found_count <- found_count + 1
   found_samps[found_count] <- g_samp
 }
 counter <- counter + 1
}
# number of iterations for 1000 samples
print(counter)
```

```
## [1] 3157
```

```r
hist(found_samps, freq = FALSE)
x <- seq(0, 10, 0.1)
y <- f_dens(x, sigma)
g <- c * dexp(x, lambda)
lines(x,y)
lines(x,g)
```

Histogram of found_samps