Project 1 - AMCS241/STAT250 Stochastic Processes

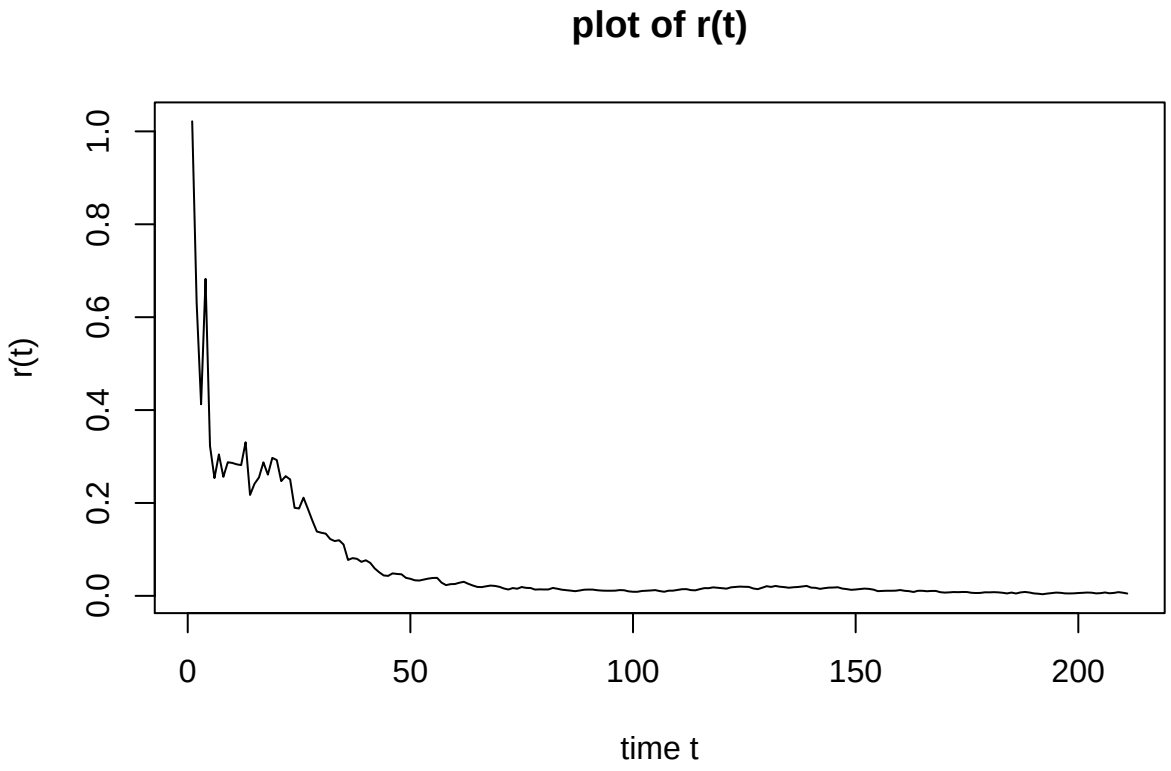
André Victor Ribeiro Amaral & Pratik Nag

November 25, 2020

Problem 1 (Estimation of the mean and covariance function)

a)

```
# loading the Rdata
load("data.RData")

# create the variable I(t)

I = covid[,2][9:220]
T = length(I)
r = rep(0,T)
for (t in 2:length(I)){
  r[t] = log(I[t]/I[t-1])
}

# plot r(t)
plot(r[2:T],type = "l",main="plot of r(t)",
     xlab = "time t",ylab = "r(t)")
```



Answer:

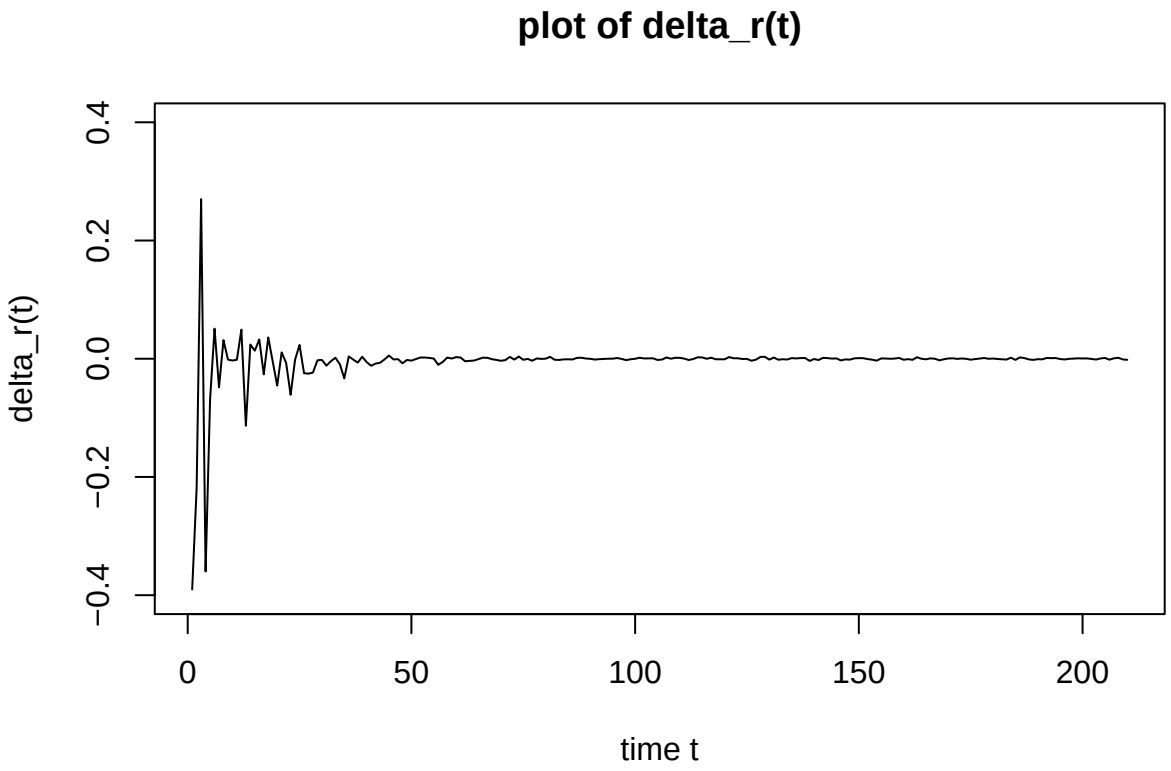
The `covid` dataset has two columns. Column 1 contains the time stamps whereas column 2 contains the covariate values, which we will denote by $I(t)$. Here, we have calculated the value of $r(t)$, which is given by

$$r(t) = log\left(\frac{I(t)}{I(t-1)}\right).$$

In the code $I(t)$ and $r(t)$ are stored in `I` and `r` respectively. Hence from the plot it is evident that r(t) is not at all stationary.

b)

```
# create the difference variable

del_r = rep(0,T)
for(t in 3:T){
  del_r[t] = r[t] - r[t-1]
}

# plot del_r(t)
plot(del_r[3:T],type = "l",main="plot of delta_r(t)",
     xlab = "time t",ylab = "delta_r(t)", ylim=c(-0.4, 0.4))
```



Answer:

In order to deal with the non-stationarity of the process, we have taken the difference, defined as $\Delta r(t)$, which is equal to

$$\Delta r(t) = r(t) - r(t-1).$$

In the above code, $\Delta r(t)$ is stored in `del_r` variable. From the plot we can say that the process is more stationary than the previous one.

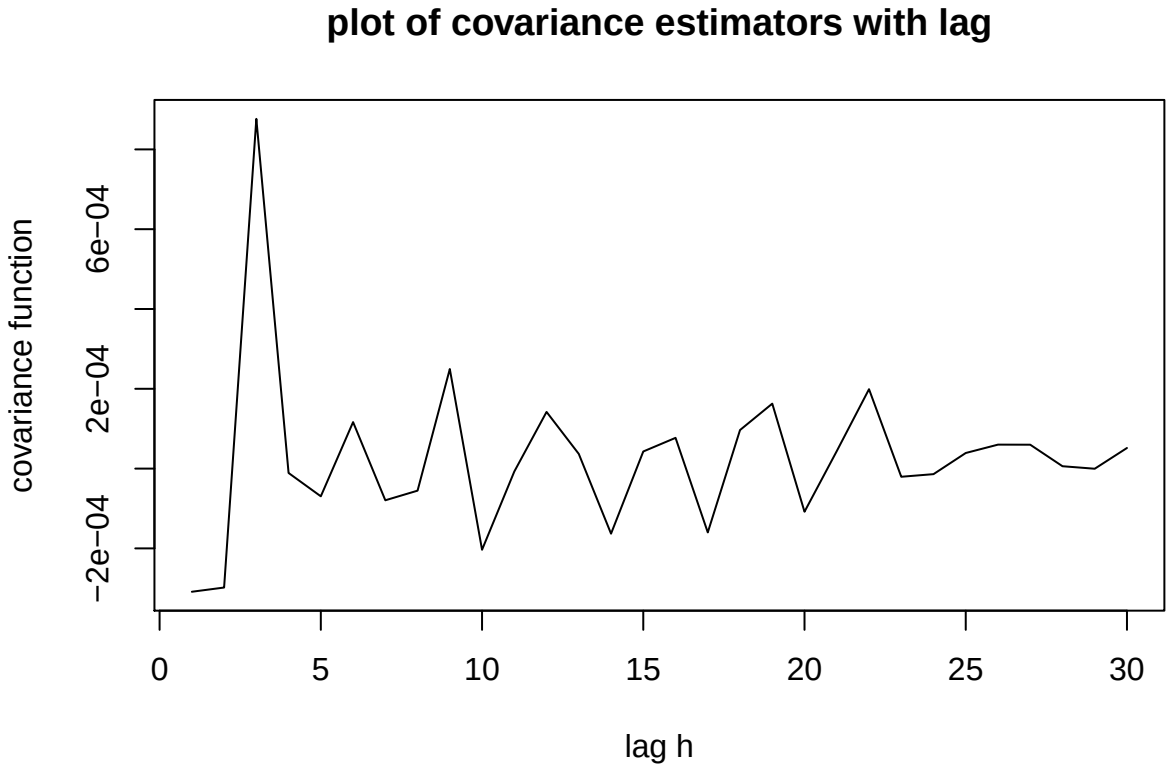
c)

```
# functions for estimating the mean and covariance functions

mean_estimator = function(stoc.process,timepoints){
  return(sum(stoc.process)/timepoints)
}

cov_estimator = function(stoc.process, timepoints, h){
```

```
  if (h >= T){
    print("Give value of h less then total number of time points.")
  }else{
    total_sum = 0
    for (t in 1:(timepoints - h)){
      within_sum_expression = (stoc.process[t]-mean_estimator(stoc.process, timepoints)) *
                              (stoc.process[t + h] -
                               mean_estimator(stoc.process, timepoints))
      total_sum = total_sum + within_sum_expression
    }
    return(total_sum / timepoints)
  }
}

print(paste0("Estimated mean: ", mean_estimator(del_r[3:T], T - 2), "."))
```

```
## [1] "Estimated mean: -0.00484121327175172."
```

```
# plot of the data for h from 1 to 30
cov_lag_h = rep(NA,T - 3)
for (h in 1:(T - 3)){
  cov_lag_h[h] = cov_estimator(del_r[3:T], T - 2, h)
}
plot(cov_lag_h[1:30],type = "l", main="plot of covariance estimators with lag",
     xlab = "lag h", ylab = "covariance function")
```



Answer:

After considering the above difference process $\Delta r(t)$ as weakly stationary, we have created two functions which will give us the *mean function estimate* and *covariance function estimate* at a given lag h. The following are the mathematical expressions of each estimate, where $X(t)$ is the weakly stationary process.

Mean estimate

$$\hat{m}_n = \frac{1}{n}\sum_{k=1}^{n} X(k).$$

Covariance estimate

$$\hat{r}_n(h) = \frac{1}{n}\sum_{k=1}^{n-h}(X(k)-m)(X(k+h)-m),$$

which is an asymptotically unbiased estimator for any weakly stationary process $X(t)$. Notice that the actual mean value m is not known, so we can estimate it with $\hat{m}_n$.

In the above code the functions `mean_estimator` and `cov_estimator` are the functions created for finding the estimated values for the mean and covariance at lag h. The function `mean_estimator` takes the list of stochastic process variables and the total number of time points as inputs, and the function `cov_estimator`, additionally to these already mentioned quantities, also takes the lag h as an input value. Here, as mean was unknown, we have estimated the mean (≈ -0.0048412) value with the value obtained from `mean_estimator`.

From the plot for lag $h \in \{1, 2, \cdots, 30\}$, we can observe that the covariance decreases as the lag increases, as one would expect from a time dependent process.

d)

```
# confidence interval

estimated_mean = mean_estimator(del_r[3:T], T - 2)
estimated_variance = ((T - 2) * cov_estimator(del_r[3:T], T - 2, 0) +
                      (2 * sum(cov_lag_h))) / (T - 2)^2
factor = qnorm(0.975, 0, 1) * sqrt(estimated_variance)
print(paste0("The 95% confidence interval is W = (",
             (estimated_mean - factor), ", ", (estimated_mean + factor),")."))
```

```
## [1] "The 95% confidence interval is W = (-0.0110046986135886, 0.00132227207008515)."
```

Answer:

Here, we have estimated the mean value with the value of $\hat{m}_n$. Recall that `estimated_mean` in code has stored this value.

If $\hat{r}_n(h)$ is the estimated covariance function, then we know that, from the definition of $\mathbb{V}(\hat{m}_n)$,

$$\mathbb{V}(\hat{m}_n) = \frac{1}{n^2}\left[n\,\hat{r}_n(0) + 2\cdot\sum_{h=1}^{n-1}\hat{r}_n(h)\right].$$

In our case, k will go from -219 to 219. The `estimated_variance` variable is storing the above variance estimate. Hence, the confidence interval for m can be written in the following way:

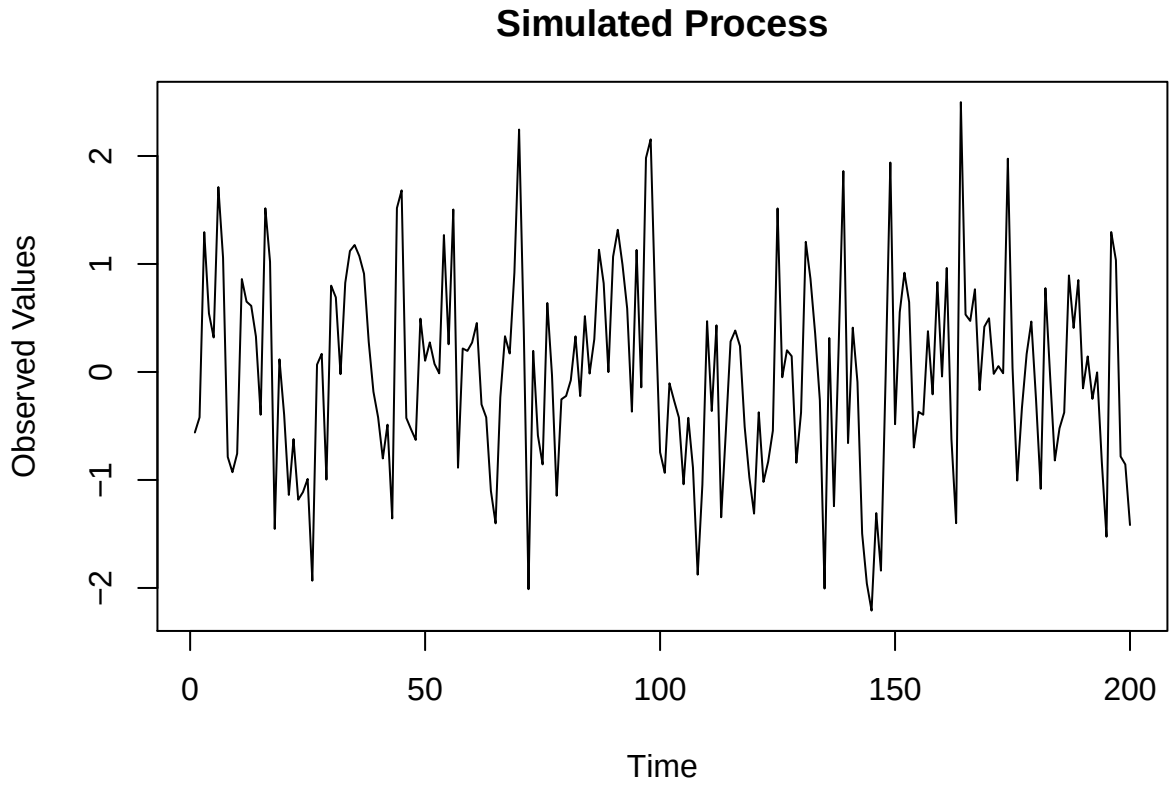
$$w_m = \left[\hat{m}_n \pm \tau_{\frac{\alpha}{2}}\sqrt{\mathbb{V}(\hat{m}_n)}\right],$$

where $\tau_{\frac{\alpha}{2}}$ is the upper $\alpha/2$ point of $N(0,1)$ distribution. From above computations, we obtained $w_m = (-0.0110046986135886, 0.00132227207008515)$.

Problem 2 (Gaussian Processes)

a)

```r
set.seed(123)

# load 'mnormt' library
library(mnormt)

n_observations = 200

mean_vector = matrix(0, nrow = 1, ncol = n_observations)

covariance_matrix = matrix(0, nrow = n_observations, ncol = n_observations)

for(i in 1:n_observations) {
    for(j in 1:n_observations) {
        covariance_matrix[i, j] = exp(-abs(i - j))
    }
}

# Generate a random random vector with multivariate Normal distribution
generated_process = rmnorm(n = 1, mean = mean_vector, varcov = covariance_matrix)

# Plot the generated process
par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(generated_process, type = 'l', main = 'Simulated Process', xlab = 'Time',
     ylab = 'Observed Values')
```



b)

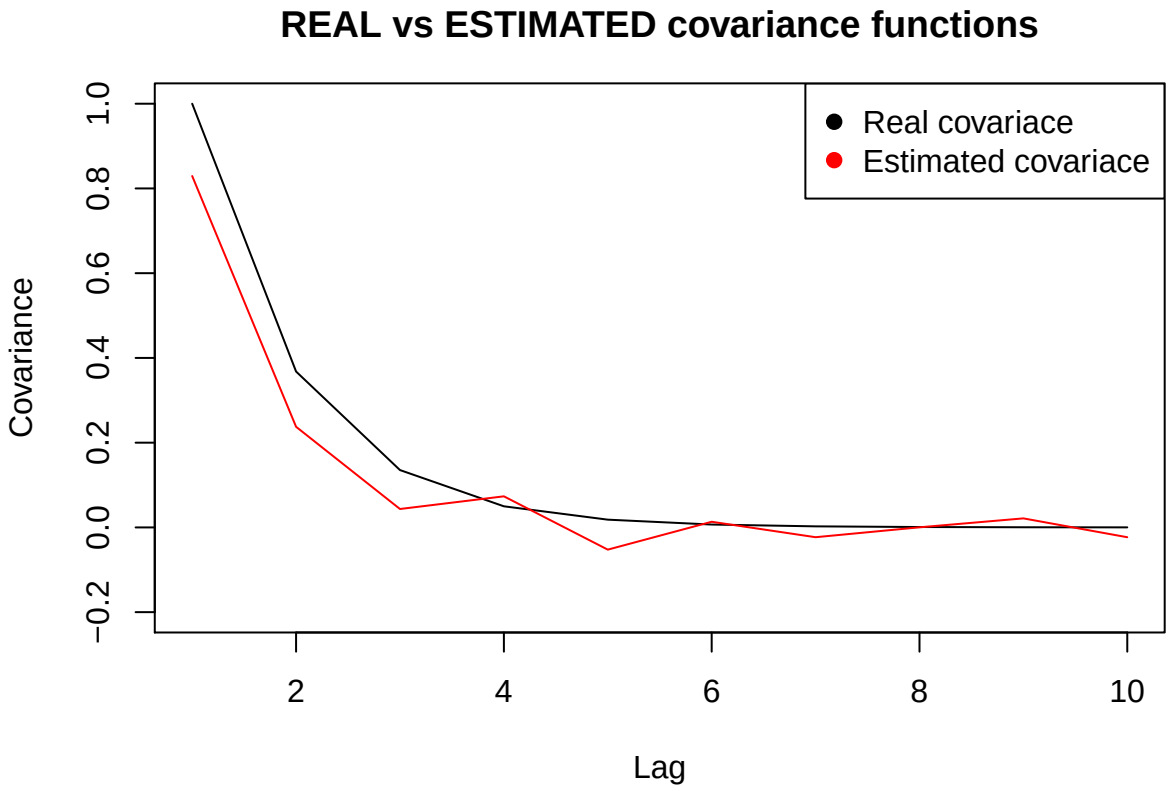
```
mean_estimator = function(sequence) {
    return(mean(sequence))
}

covariance_estimator = function(sequence, lag, est_mean) {
    sum = 0
    for(t in 1:(length(sequence) - lag)) {
        sum = sum + ((sequence[t] - est_mean) * (sequence[t + lag] - est_mean))
    }
    return(sum / length(sequence))
}
```

```
est_mean = mean_estimator(generated_process)
print(paste0('The true mean is \'0\' and the estimated mean is \'', est_mean, '\'.'))
```

```
## [1] "The true mean is '0' and the estimated mean is '-0.00879406217819189'."
```

```
tru_covariance = matrix(0, nrow = 1, ncol = n_observations)
est_covariance = matrix(0, nrow = 1, ncol = n_observations)

for(i in 0:(n_observations - 1)) {
    tru_covariance[1, i + 1] = exp(-abs(i))
    est_covariance[1, i + 1] = covariance_estimator(generated_process, i, est_mean)
}
```

```
# Plot the real and estimated covariance functions
par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(tru_covariance[1, 0:10], type = 'l', main = 'REAL vs ESTIMATED covariance functions',
     xlab = 'Lag', ylab = 'Covariance', ylim = c(-0.20, 1))
lines(est_covariance[1, 0:10], col = 'red')
legend('topright', legend = c('Real covariace', 'Estimated covariace'),
       col = c('black', 'red'), pch = 19)
```



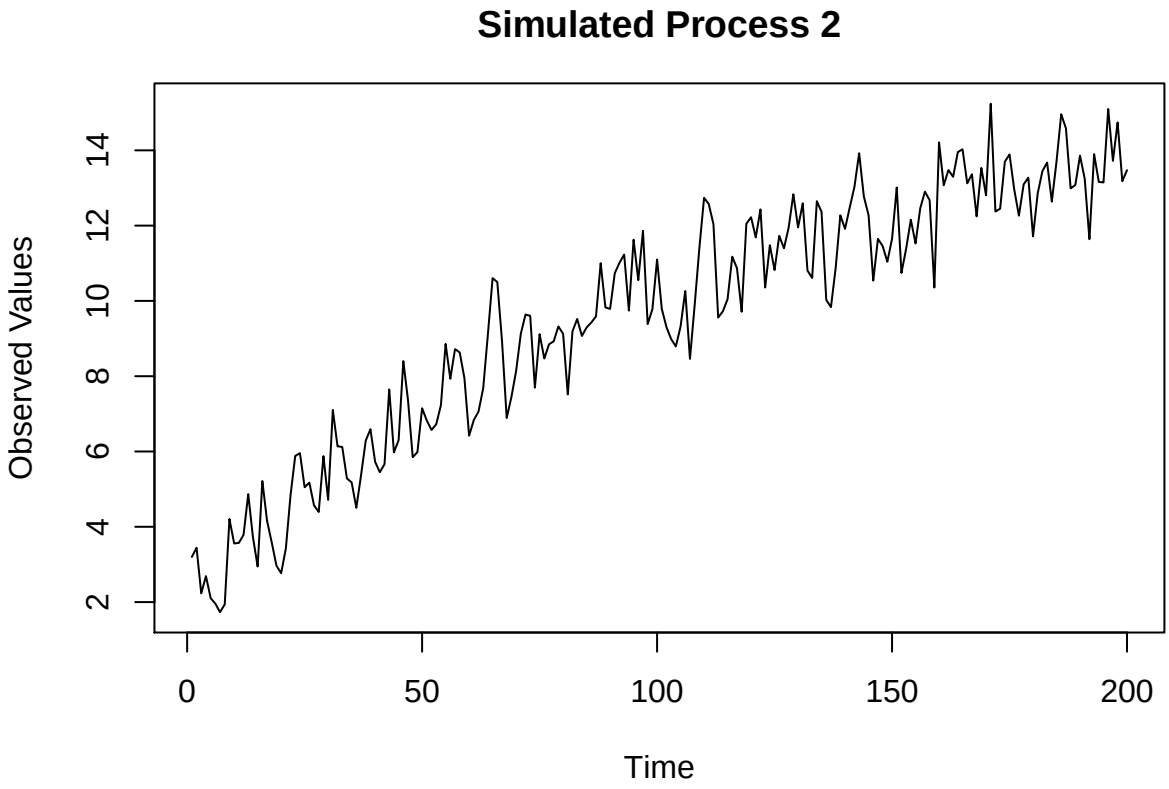
In the above code, notice that the `mean_estimator()` and the `covariance_estimator()` functions are the same as the `mean_estimator()` and `cov_estimator()` functions from the previous problem; we have just written them in a slightly different way.

Answer:

As we have a Gaussian process that is (weakly) stationary (recall that we have constant *mean* and *covariance functions*, for some fixed lag h), our estimates for both the mean and convariance functions work very well, and we get estimates very close to the real values – as one can see comparing $m = 0$ to $\hat{m}_n = -0.00993975536855208$ and the curves of $r(h)$ and $\hat{r}_n(h)$ in the above plot.

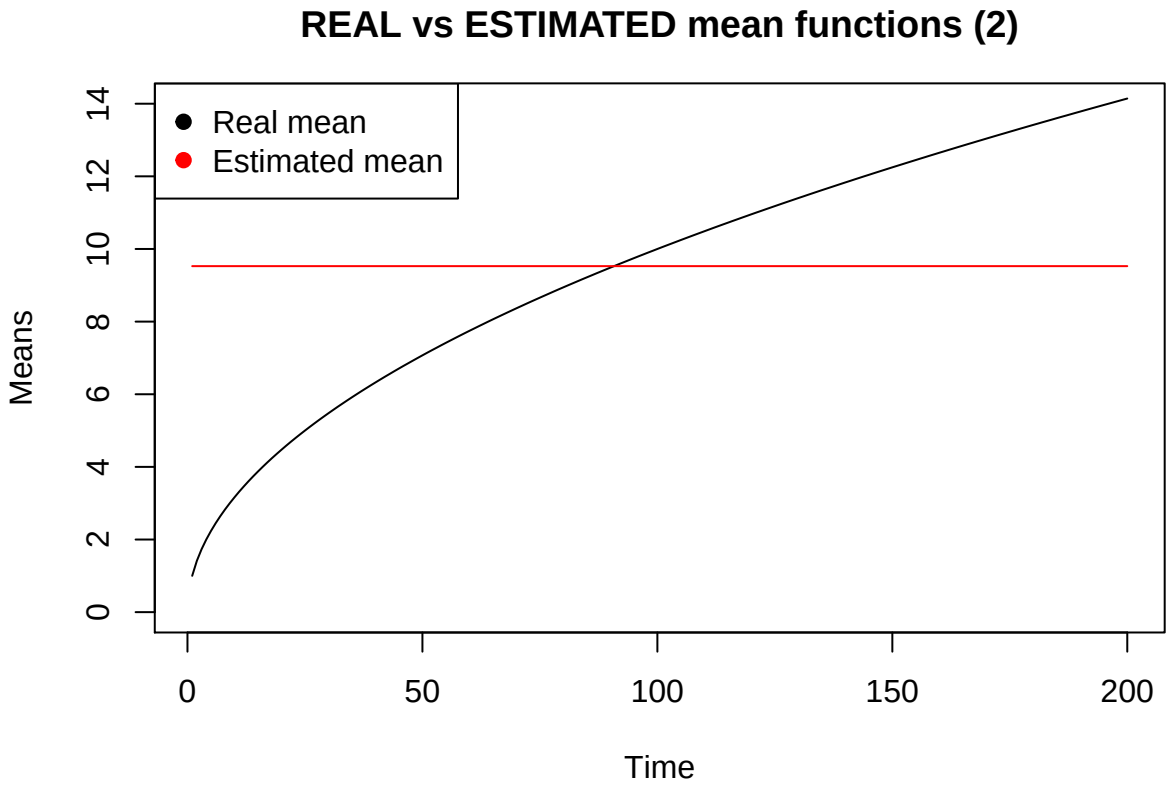
c)

```
mean_vector2 = matrix(0, nrow = 1, ncol = n_observations)

for(i in 1:n_observations) {
    mean_vector2[1, i] = sqrt(i)
}

generated_process2 = rmnorm(n = 1, mean = mean_vector2, varcov = covariance_matrix)

# Plot the generated process
par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(generated_process2, type = 'l', main = 'Simulated Process 2',
     xlab = 'Time', ylab = 'Observed Values')
```

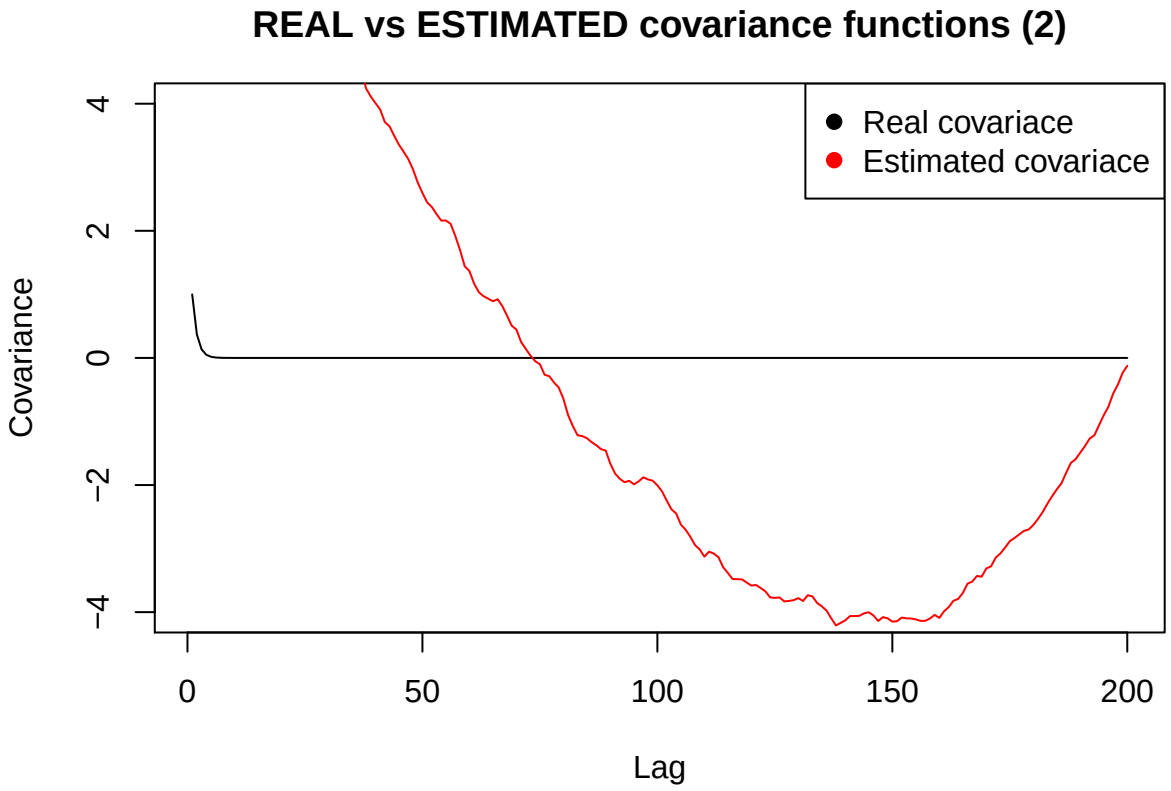


d)

```
est_mean2 = mean_estimator(generated_process2)
est_mean2 = matrix(est_mean2, nrow = 1, ncol = n_observations)

par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(mean_vector2[1, ], type = 'l', main = 'REAL vs ESTIMATED mean functions (2)',
     xlab = 'Time', ylab = 'Means', ylim = c(0, 14))
lines(est_mean2[1, ], col = 'red')
legend('topleft', legend = c('Real mean', 'Estimated mean'),
       col = c('black', 'red'), pch = 19)
```




```
tru_covariance2 = matrix(0, nrow = 1, ncol = n_observations)
est_covariance2 = matrix(0, nrow = 1, ncol = n_observations)

for(i in 0:(n_observations - 1)) {
    tru_covariance2[1, i + 1] = exp(-abs(i))
    est_covariance2[1, i + 1] = covariance_estimator(generated_process2, i,
                                                     est_mean2[1, 1])
}

par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(tru_covariance2[1, ], type = 'l',
     main = 'REAL vs ESTIMATED covariance functions (2)',
     xlab = 'Lag', ylab = 'Covariance', ylim = c(-4, 4))
lines(est_covariance2[1, ], col = 'red')
legend('topright', legend = c('Real covariace', 'Estimated covariace'),
       col = c('black', 'red'), pch = 19)
```



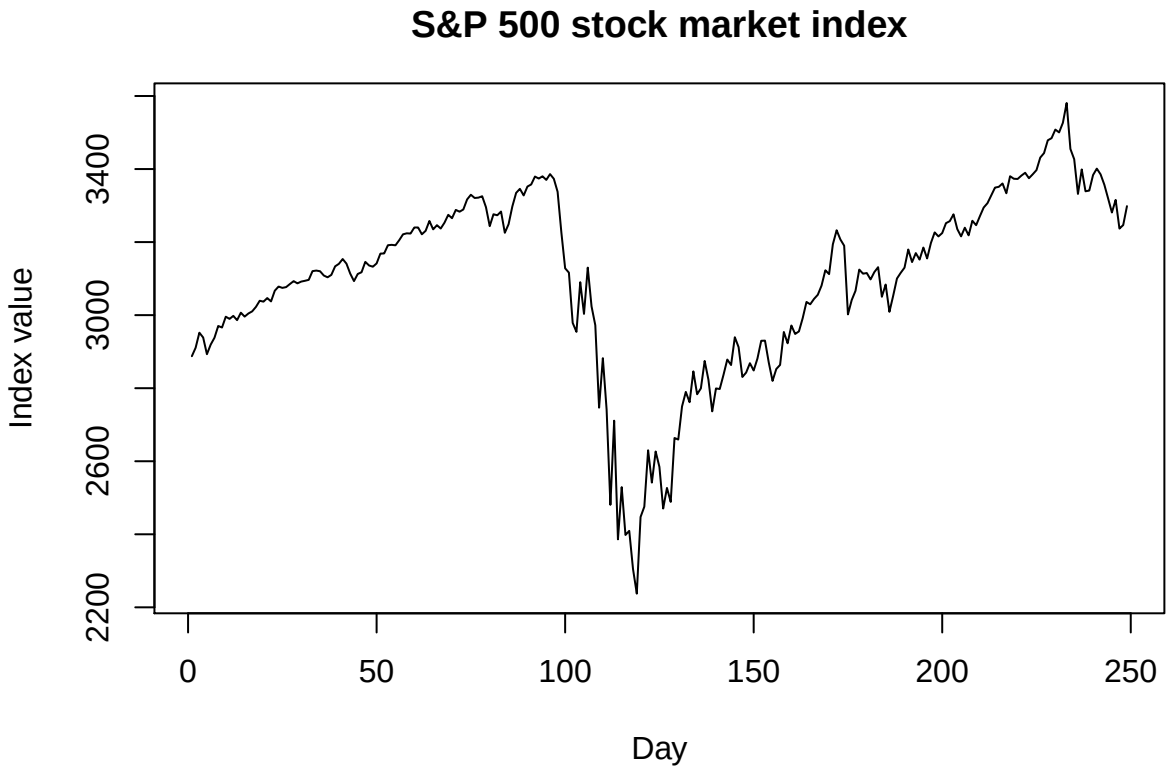
Remark: In order to see the difference between the magnitude of the two curves in the "`REAL vs ESTIMATED covariance functions (2)`" graph, we plotted the covariance functions for all lags $h \in \{0, 1, \cdots, 199\}$, instead of just for $h \in \{0, 1, \cdots, 10\}$. However if one wants to see only the first 11 entries of the `tru_covariance2` and `est_covariance` vectors, it suffices to use the following two lines of code: `plot(tru_covariance2[1, 1:11])` and `lines(est_covariance2[1, 1:11])`.

Answer:

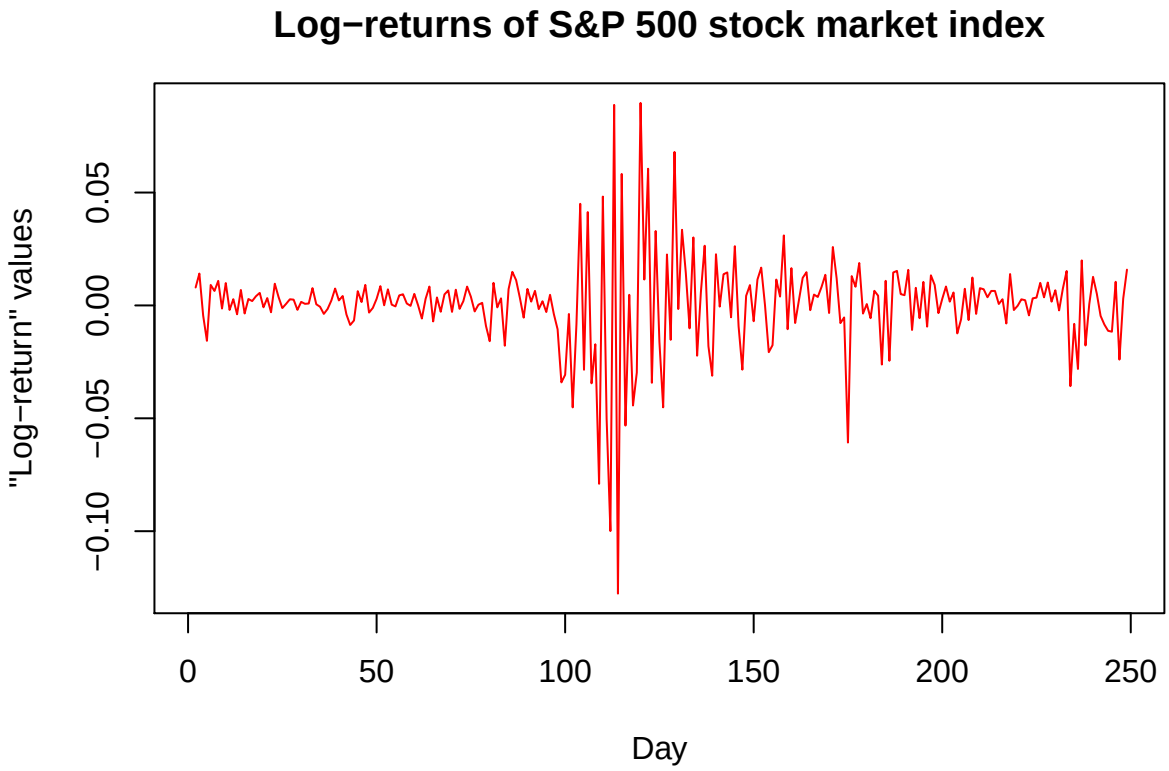
In this case, notice that, since our process is, by construction, **not** (weakly) stationary, our estimates (computed using the `mean_estimator` and `covariance_estimator` functions) did not work. We have defined the mean function as $m(t) = \sqrt{t}$, which is **not** constant for any t. Therefore, the theorem that gives us a formula to compute $\hat{m}_n$ and $\hat{r}_n(h)$ for some lag h **no longer holds**.

e)

```r
log_return = matrix(NA, nrow = 1, ncol = length(sp))

for(i in 2:length(sp)) {
    log_return[1, i] = log(sp[i] / sp[i - 1])
}

# Plot the original data
par(mar = c(5.6, 4.1, 3.1, 2.1))
plot(sp, type = 'l', main = 'S&P 500 stock market index',
     xlab = 'Day', ylab = 'Index value')
```




```
# Plot the transformed data
plot(log_return[1, ], type = 'l',
     main = 'Log-returns of S&P 500 stock market index',
     xlab = 'Day', ylab = '"Log-return" values', col = 'red')
```



Answer:

As one can see from the two plots above, the original process is clearly not (weakly) stationary. On the other hand, one can argue that the `log-returns` process can be viewed as a (weakly) stationary one (recall that, under the assumption of normality for all $X(t)$, "weakly stationary" and "stationary" can be used interchangeably, since the former implies the latter). And that is why we will choose to work with the `log-returns` process, instead of with the original data.

f)

```
# Predicting the three next values of our process
new_length = length(sp) + 2

mean_vector_sp = matrix(0, nrow = 1, ncol = new_length)
covariance_matrix_sp = matrix(0, nrow = new_length, ncol = new_length)

sigma_2 = (0.02**2)
a = 0.4

for(i in 1:new_length) {
    for(j in 1:new_length) {
        covariance_matrix_sp[i, j] = (sigma_2 / (1 - a**2)) * ((-a)**(abs(i - j)))
    }
}

# Compute partial quantities of interest
x_a = log_return[1, 2:length(log_return[1, ])]

mean_a = mean_vector_sp[1, 1:(length(sp) - 1)]
mean_b = mean_vector_sp[1, length(sp):length(mean_vector_sp[1, ])]

sigma_aa = covariance_matrix_sp[1:(length(sp) - 1), 1:(length(sp) - 1)]
sigma_ab = covariance_matrix_sp[1:(length(sp) - 1),
                                length(sp):length(covariance_matrix_sp[1, ])]
sigma_ba = covariance_matrix_sp[length(sp):length(covariance_matrix_sp[, 1]),
                                1:(length(sp) - 1)]
sigma_bb = covariance_matrix_sp[length(sp):length(covariance_matrix_sp[, 1]),
                                length(sp):length(covariance_matrix_sp[1, ])]


# Compute the conditional mean convariance values
mean_b_given_a  = mean_b + ((sigma_ba %*% solve(sigma_aa)) %*% (x_a - mean_a))
sigma_b_given_a = sigma_bb - (sigma_ba %*% solve(sigma_aa) %*% sigma_ab)

# Print final answer
for(i in 1:(new_length - (length(sp) - 1))) {
    print(paste0('R_', length(sp) + i, ' has mean \'',
                 format(mean_b_given_a[i], width = 10, digits = 5),
                 '\' and variance \'', format(sigma_b_given_a[i, i],
                                              width = 9, digits = 4), '\'.'))
}
```

```
## [1] "R_250 has mean '-0.0063402' and variance '    4e-04'."
## [1] "R_251 has mean ' 0.0025361' and variance ' 0.000464'."
## [1] "R_252 has mean '-0.0010144' and variance '0.0004742'."
```

Answer:

In this case, we are predicting the values of $R_{250},\ R_{251}$ and R_{252} using the following mean and covariance functions:

$$m(t) = 0 \quad \text{and} \quad r(h) = \frac{\sigma^2}{(1-a^2)} \cdot (-a)^{|h|},$$

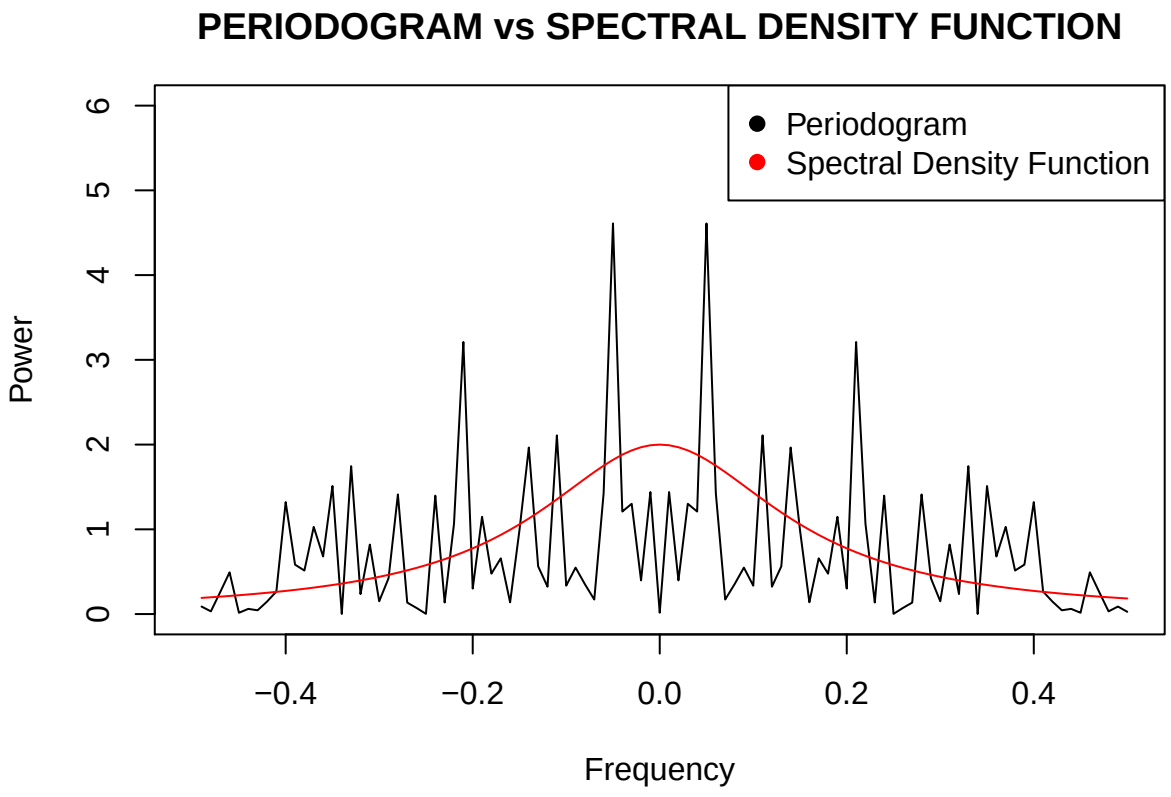
with $\hat{\sigma} = 0.02$ and $\hat{a} = 0.4$. Then, as showed in the last lines of the above code:

```
'R_{250}' has mean '-0.0063402' and variance '   0.0004',
'R_{251}' has mean ' 0.0025361' and variance ' 0.000464' and
'R_{252}' has mean '-0.0010144' and variance '0.0004742'.
```

Problem 3 (Periodogram and filtering)

a)

```
compute_fourier_transform = function(process, freq) {
  return(sum(c(process) * exp(-1 * complex(real = 0, imaginary = 1) * 2 * pi * freq
                              * c(0:(length(c(process)) - 1)))))
}

w = seq(from = -0.49, to = 0.50, by = 0.01)
periodogram = spectral_df = matrix(0, nrow = 1, ncol = length(w))

for(i in 1:(length(w))) {
  periodogram[, i] = (1 / length(generated_process)) *
                     (Mod(compute_fourier_transform(generated_process, w[i])) ** 2)
  spectral_df[, i] = 2 / (1 + (2 * pi * w[i]) ** 2)
}

par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(w, periodogram[1, ], type = 'l', main = 'PERIODOGRAM vs SPECTRAL DENSITY FUNCTION',
     xlab = 'Frequency', ylab = 'Power', xlim = c(-0.5, 0.5), ylim = c(0, 6))
lines(w, spectral_df[1, ], col = 'red')
legend('topright', legend = c('Periodogram', 'Spectral Density Function'),
       col = c('black', 'red'), pch = 19)
```

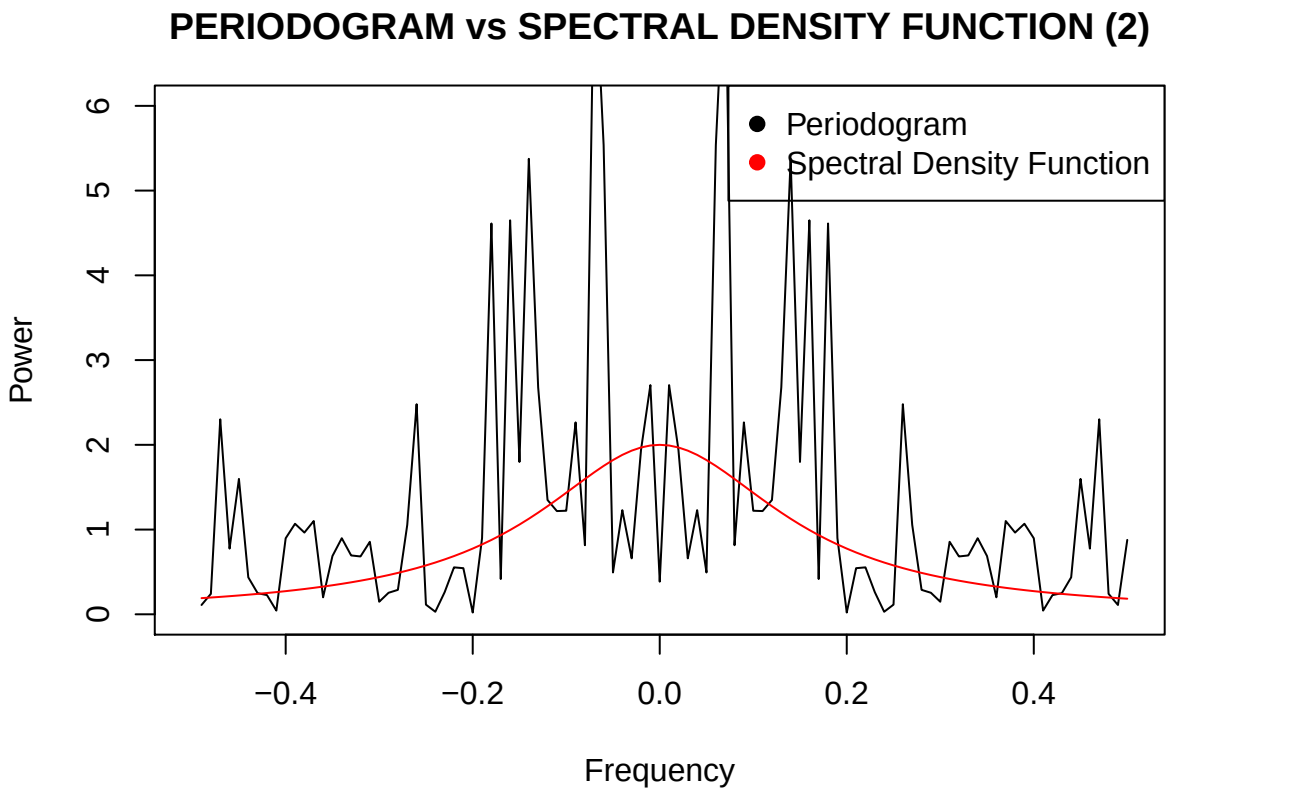


Answer:

As one can see from the above plot, the $\hat{R}(\omega)$ does not approximate well the real the Spectral Density Function $R(\omega) = \frac{2}{1+(2\pi\omega)^2}$. We will see a different estimator for the Spectral Density Function on item (d).

b)

```
n_observations = 1000

mean_vector = matrix(0, nrow = 1, ncol = n_observations)
covariance_matrix = matrix(0, nrow = n_observations, ncol = n_observations)

for(i in 1:n_observations) {
  for(j in 1:n_observations) {
    covariance_matrix[i, j] = exp(-abs(i - j))
  }
}
generated_process3 = rmnorm(n = 1, mean = mean_vector, varcov = covariance_matrix)

periodogram3 = matrix(0, nrow = 1, ncol = length(w))

for(i in 1:(length(w))) {
  periodogram3[, i] = (1 / length(generated_process3)) *
                      (Mod(compute_fourier_transform(generated_process3, w[i])) ** 2)
}

par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(w, periodogram3[1, ], type = 'l', main = 'PERIODOGRAM vs SPECTRAL DENSITY FUNCTION (2)',
     xlab = 'Frequency', ylab = 'Power', xlim = c(-0.5, 0.5), ylim = c(0, 6))
lines(w, spectral_df[1, ], col = 'red')
legend('topright', legend = c('Periodogram', 'Spectral Density Function'),
       col = c('black', 'red'), pch = 19)
```

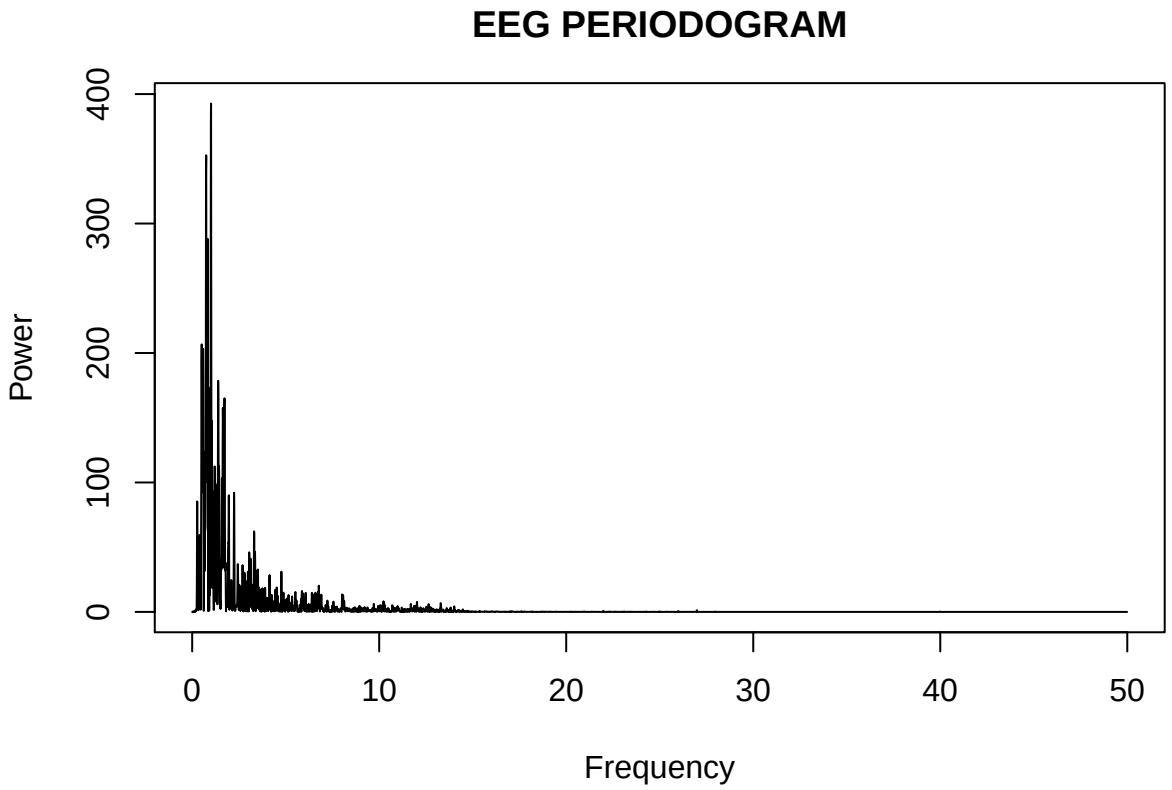


Answer:

In this case, we did not have observed any improvement in the estimation of $R(\omega)$, even though we have increased the number of observations from $n = 200$ to $n = 1000$. This phenomenon can be justified by the fact that $\mathbb{V}(\hat{R}(\omega))$ does not goes to 0 as $n \to +\infty$; i.e. $\hat{R}(\omega)$ is not a consistent estimator for $R(\omega)$.

c)

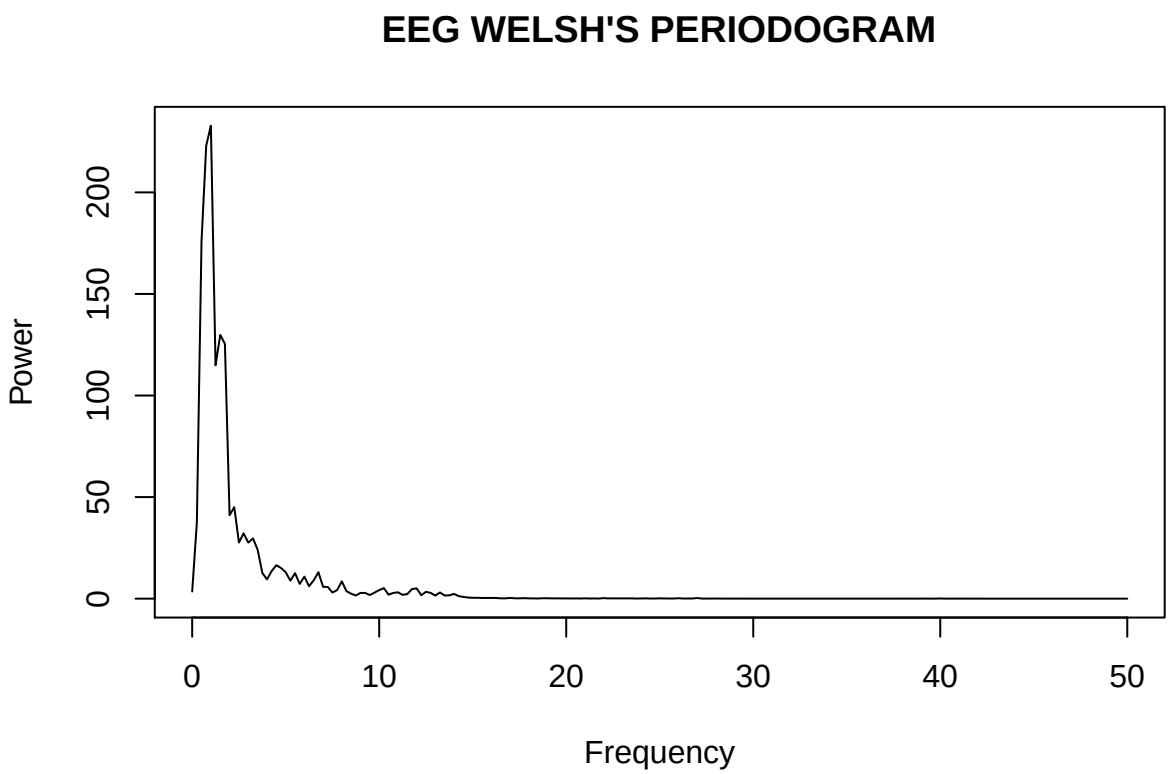
```
compute_fourier_transform_v2 = function(process, freq, d) {
  return(sum(c(process) * exp(-1 * complex(real = 0, imaginary = 1) * 2 * pi * freq
                              * (d * c(0:(length(c(process)) - 1))))))
}

w2 = seq(from = 0, to = 49.99, by = 0.01)

periodogram_eeg = matrix(0, nrow = 1, ncol = length(w2))

for(i in 1:(length(w2))) {
  periodogram_eeg[, i] = (1/100) * (1 / length(eeg)) *
                      (Mod(compute_fourier_transform_v2(eeg, w2[i], 1/100)) ** 2)
}

par(mar = c(5.1, 4.1, 3.1, 2.1))
plot(w2, periodogram_eeg[1, ], type = 'l', main = 'EEG PERIODOGRAM',
     xlab = 'Frequency', ylab = 'Power', xlim = c(0, 50))
```



d)

```
library(bspec)
welsh = welchPSD(ts(eeg, frequency = 100), seglength = 4)
```

```r
plot(welsh$frequency, welsh$power, type = 'l', main = 'EEG WELSH\'S PERIODOGRAM',
     xlab = 'Frequency', ylab = 'Power', xlim = c(0, 50))
```




```r
compute_approx_area = function(low, upper) {
  low_position   = match(low,   welsh$frequency)
  upper_position = match(upper, welsh$frequency)
  area = 0
  for(i in (low_position + 1):upper_position) {
    area = area + ((welsh$frequency[i] - welsh$frequency[i - 1]) *
                   welsh$power[i])
  }
  return(area)
}
print(paste0('The average band power of the \'delta\' band was ',
             round(compute_approx_area(0.5, 4), 3), '.'))
```

```
## [1] "The average band power of the 'delta' band was 268.832."
```

```r
print(paste0('The average band power of the \'theta\' band was ',
             round(compute_approx_area(4,   8), 3), '.'))
```

```
## [1] "The average band power of the 'theta' band was 38.272."
```

```r
print(paste0('The average band power of the \'alpha\' band was ',
             round(compute_approx_area(8,  12), 3), '.'))
```

```
## [1] "The average band power of the 'alpha' band was 12.269."
```

```r
print(paste0('The average band power of the \'beta\'  band was ',
             round(compute_approx_area(12, 30), 3), '.'))
```

```
## [1] "The average band power of the 'beta'  band was 7.107."
```

Answer:

As showed above, the average band power of the delta, theta, alpha and beta bands were 268.832, 38.272, 12.269 and 7.107, respectively. Based on this, one may say that the person was deep sleeping, since we have a predominance of slow-waves with frequency range comprised between 0.5 and 4 Hz.

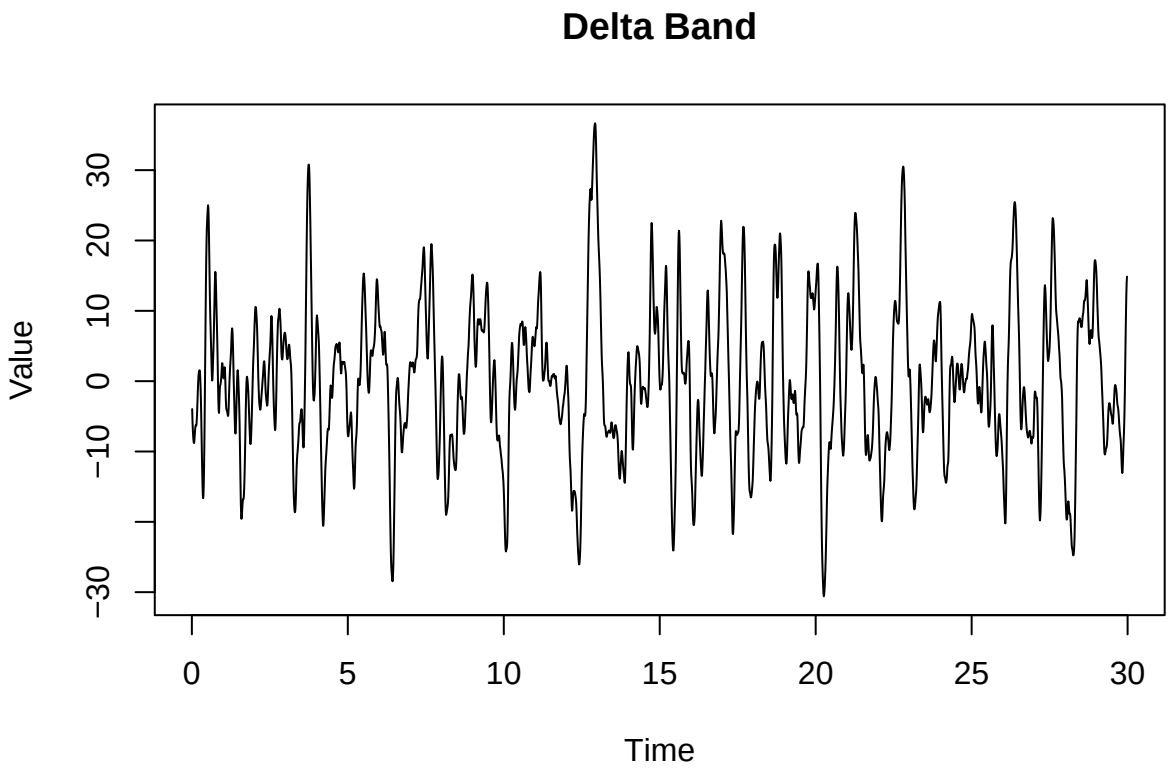
e)

Answer:

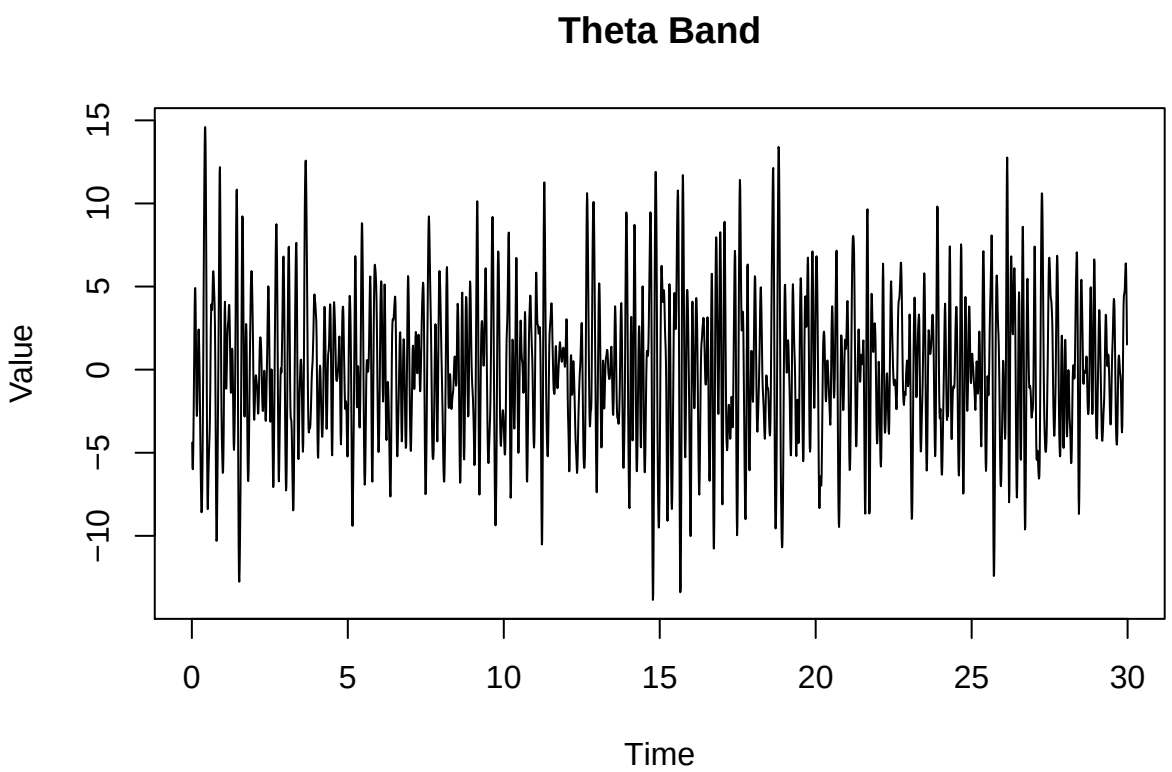
$$
\begin{aligned}
h(t) = \int \exp(i2\pi\omega t)\mathbb{I}_{a<|\omega|<b}(\omega)d\omega &= \int_{-b}^{-a} \exp(i2\pi\omega t)d\omega + \int_{a}^{b} \exp(i2\pi\omega t)d\omega \\
&= \frac{\exp(-i2\pi at) - \exp(-i2\pi bt)}{i2\pi t} + \frac{\exp(i2\pi bt) - \exp(i2\pi at)}{i2\pi t} \\
&= \frac{\exp(i2\pi bt) - \exp(-i2\pi bt)}{i2\pi t} - \frac{\exp(i2\pi at) - \exp(-i2\pi at)}{i2\pi t} \\
&= \frac{sin(2\pi bt)}{\pi t} - \frac{sin(2\pi at)}{\pi t} = \frac{\sin(2\pi tb) - \sin(2\pi ta)}{\pi t} \qquad \blacksquare.
\end{aligned}
$$

f)

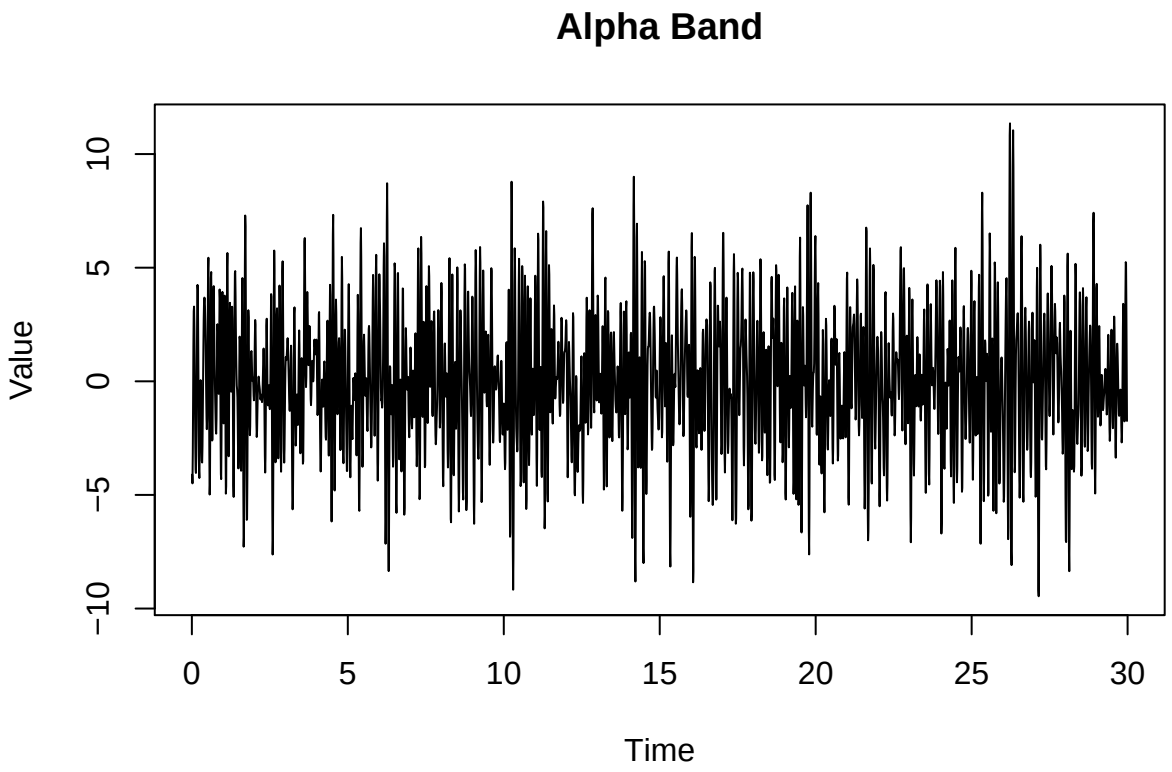
```
impulse_response = function(a, b, t) {
  return((sin(2 * pi * t * b) - sin(2 * pi * t * a)) / (pi * t))
}

frequency_filter = function(process, d, a, b) {
  new_process = rep(0, length(process))

  for(t in 1:(length(process))) {
    sum = 0
    for(i in 0:(length(process) - 1)) {
      if((t - i + 1) > 0) {
        if(i == 0) {
          sum = sum + (2 * (b - a) * process[t - i + 1] * d)
        } else {
          sum = sum + impulse_response(a, b, d * i) * process[t - i + 1] * d
        }
      }
    }
    new_process[t] = sum
  }

  return(new_process)
}

delta = frequency_filter(eeg, 1/100, 0.5, 4)
theta = frequency_filter(eeg, 1/100, 4,   8)
alpha = frequency_filter(eeg, 1/100, 8,  12)
beta  = frequency_filter(eeg, 1/100, 12, 30)

x_axis = 1:(length(eeg)) * (1/100)
plot(x_axis, delta, type = 'l', main = 'Delta Band',
   xlab = 'Time', ylab = 'Value')
```



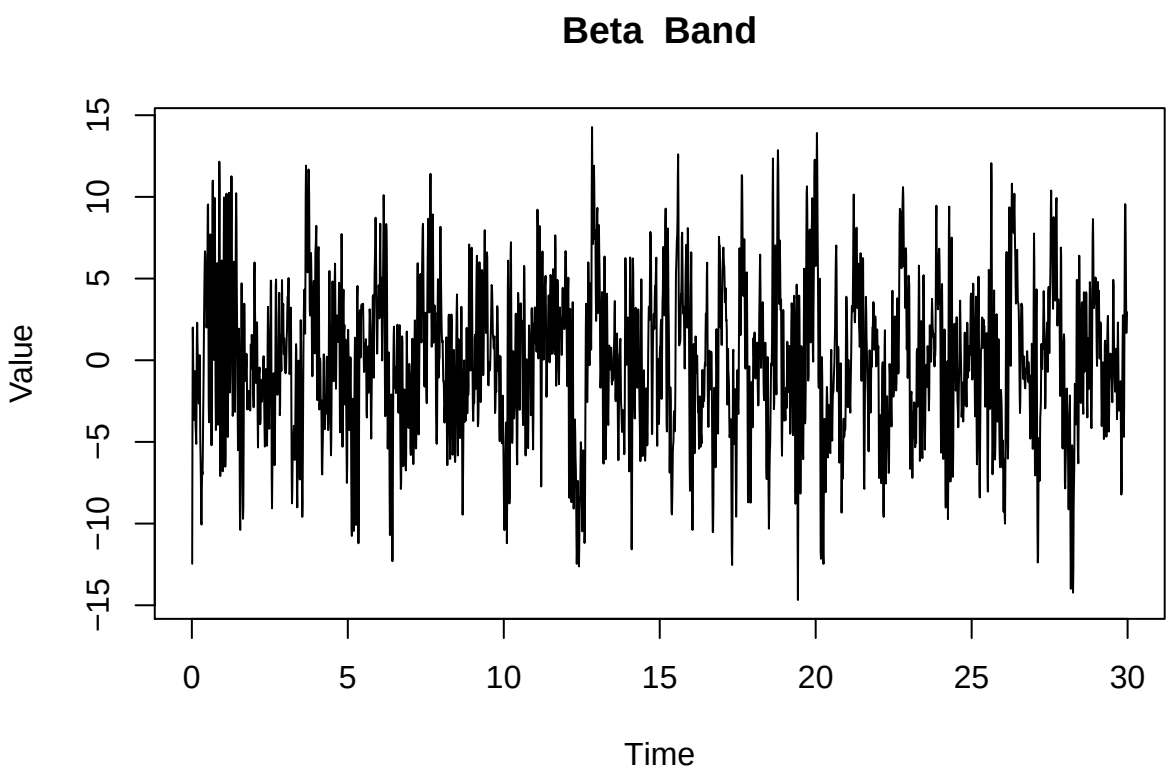

```
plot(x_axis, theta, type = 'l', main = 'Theta Band',
   xlab = 'Time', ylab = 'Value')
```




```
plot(x_axis, alpha, type = 'l', main = 'Alpha Band',
   xlab = 'Time', ylab = 'Value')
```




```
plot(x_axis, beta,  type = 'l', main = 'Beta  Band',
   xlab = 'Time', ylab = 'Value')
```



Problem 4 (Simulation of Markov Chain)

a)

```
customer = samba$customer
processing_time = samba$processing_time
T = length(customer)
interarrival_time = rep(NA,T)
interarrival_time[1] = customer[1]

for (i in 2:T){
  interarrival_time[i] = customer[i]-customer[i-1]
}

# we know inter arrival time follows exponential

est_lambda = 1 / (mean(interarrival_time))

# processing time also follows exponential
est_mu = 1 / (mean(processing_time))

# estimates of lambda and mu

print(paste0("The estimates of mu and lambda are: ",
             est_lambda, " and ", est_mu, ", respectively."))
```

```
## [1] "The estimates of mu and lambda are: 0.498671505608878 and 0.608041645561374, respectively."
```

Answer:

In the above problem we have estimated the rate of the Poisson process (λ) in the following way:

We know that the inter arrival times between occurrences of a Poisson process follows Exponential with mean $1/\lambda$, independently. Hence, we first calculated the inter arrival times for the data `customer`; then calculated the mean, which is an estimate of $1/\lambda$. Hence, the estimate of λ is reciprocal of the calculated mean. Similarly, an estimate of μ will be the reciprocal of the mean of the column `processing_time`. From the calculations, we can say that the estimates for λ and μ are

$$\hat{\lambda} = 0.498671505608878$$
$$\hat{\mu} = 0.608041645561374.$$

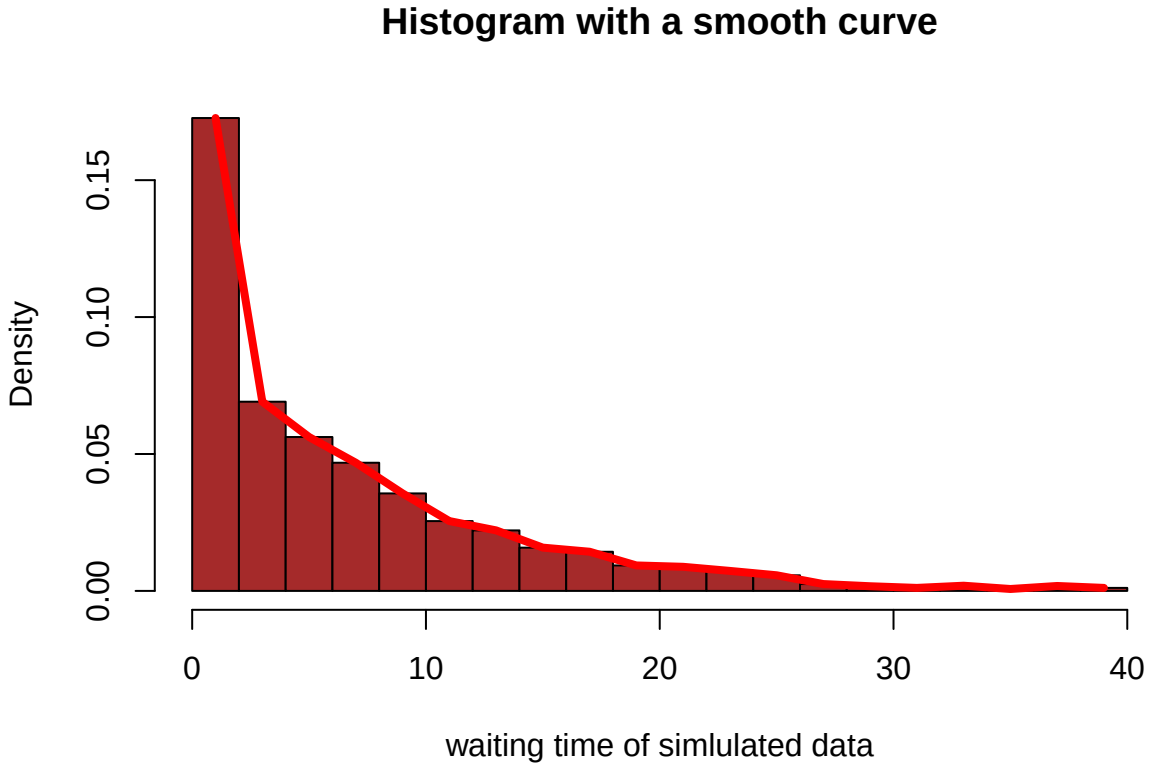
b)

```
# simulate 5000 points

n = 5000
set.seed(233784)
interarrival_time_simulated = rexp(n , rate = est_lambda)
processing_time_simulated = rexp(n , rate = est_mu)

customer_simulated = rep(NA,n)
customer_simulated[1] = interarrival_time_simulated[1]

for(i in 2:n){
```

```
  customer_simulated[i] = customer_simulated[i-1] + interarrival_time_simulated[i]
}
waiting_time = rep(NA,n)
waiting_time[1] = 0

for(i in 2:n){
  waiting_time[i] = max(0,
                        (customer_simulated[i-1]+
                           waiting_time[i-1] +
                          processing_time_simulated[i-1]-
                          customer_simulated[i]))
}
average_waiting_time = mean(waiting_time)
print(paste("Average waiting time from simulated data is : ", average_waiting_time))
```

```
## [1] "Average waiting time from simulated data is :  6.54525818624759"
```

```
h <- hist(waiting_time, breaks = 20 , col="brown", probability = TRUE,
          xlab="waiting time of simlulated data",
          main="Histogram with a smooth curve")
lines(h$mids,h$density,col="red",lwd=4)
```



Answer:

Here, we have simulated 5000 points using $\hat{\lambda}$ and $\hat{\mu}$. For simulation of the customer dataset (`customer_simulated`), we have simulated the inter arrival times (`interarrival_time_simulated`) from the following distribution:

$$T \sim \text{Exp}\left(\frac{1}{\hat{\lambda}}\right)$$

Then, the `customer` data is given by:

$$\begin{aligned}\text{Customer Simulated}[i] &= \sum_{j=1}^{i} \text{Simulated Interarrival}[j] \\ &= \text{Customer Simulated}[i-1] + \text{Simulated Interarrival}[i].\end{aligned}$$

The processing time has been simulated from the distribution,

$$P \sim \text{Exp}\left(\frac{1}{\hat{\mu}}\right)$$

The calculation of the waiting times have been explained through the following table presented in Figure 1.

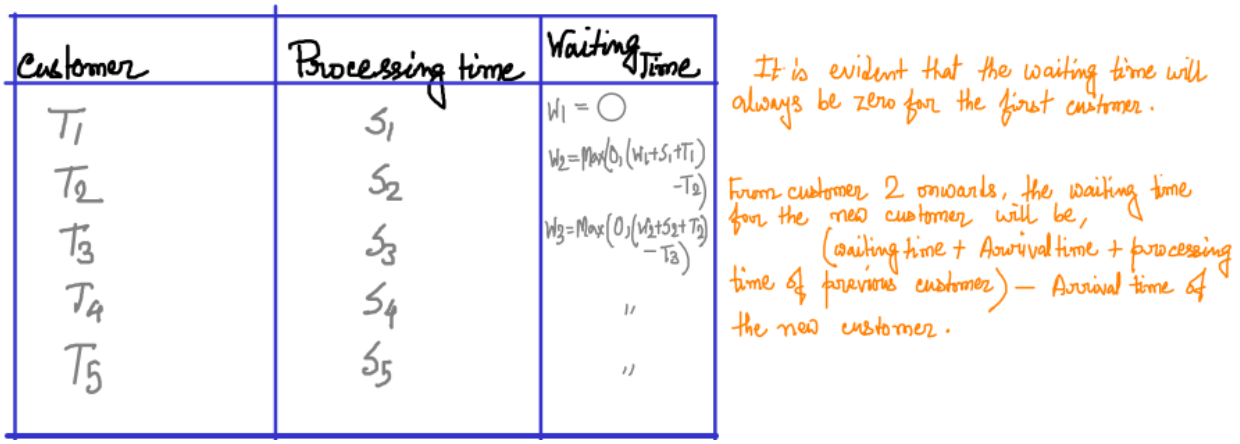


Figure 1: Part 4-b) solution.

From the above calculations the average waiting time is 6.54525818624759.

From the histogram plot we can say that the waiting times follow an exponential distribution.

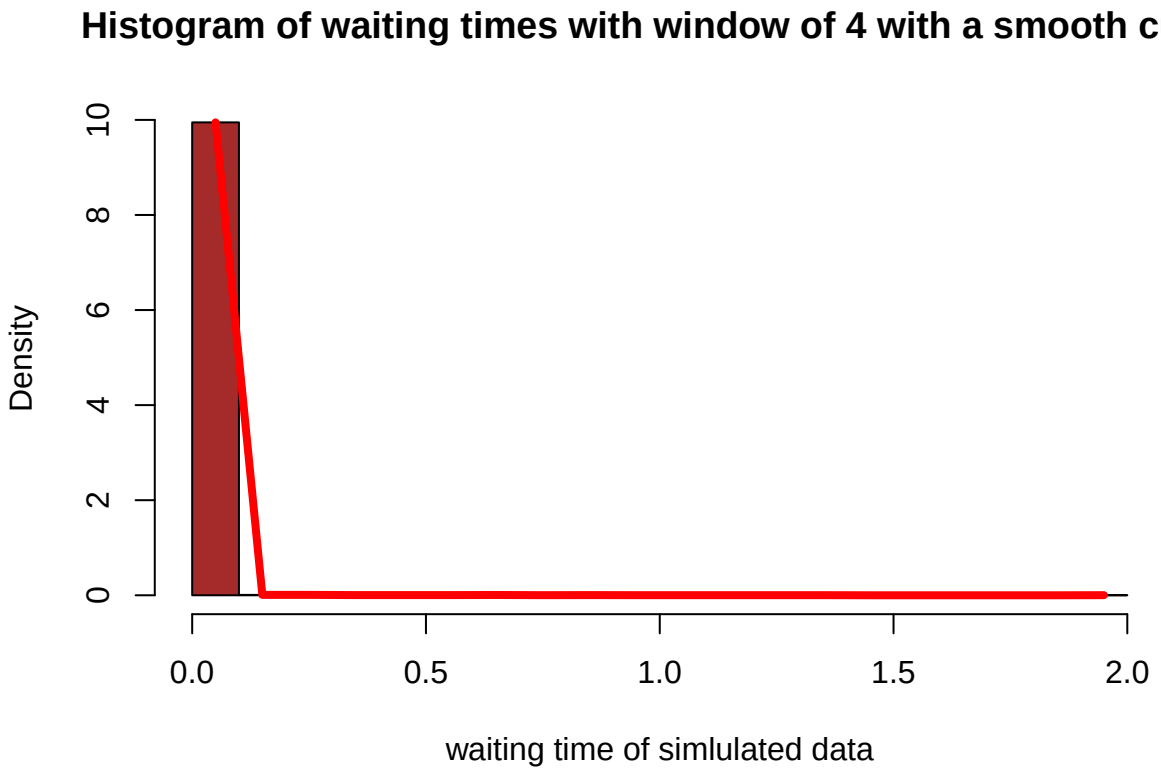
c)

```
# Waiting time for a window of 4
waiting_time_revised = rep(NA,n)
waiting_time_revised[1] = 0
waiting_time_revised[2] = 0
waiting_time_revised[3] = 0
waiting_time_revised[4] = 0

index = c(1,2,3,4)

for(i in 5:n){
  t1 = index[1]
  t2 = index[2]
  t3 = index[3]
  t4 = index[4]
  k1 = customer_simulated[t1] + processing_time_simulated[t1] + waiting_time_revised[t1]
  k2 = customer_simulated[t2] + processing_time_simulated[t2] + waiting_time_revised[t2]
```

```
  k3 = customer_simulated[t3] + processing_time_simulated[t3] + waiting_time_revised[t3]
  k4 = customer_simulated[t4] + processing_time_simulated[t4] + waiting_time_revised[t4]
  f = c(k1,k2,k3,k4)
  m = which.min(f)
  waiting_time_revised[i] = max(0,(min(f) - customer_simulated[i]))
  index[m] = i
}

average_waiting_time_revised = mean(waiting_time_revised)
print(paste("Average waiting time from simulated data is : ",
            average_waiting_time_revised))
```

```
## [1] "Average waiting time from simulated data is :  0.00336876906505022"
```

```
h <- hist(waiting_time_revised, breaks = 20 , col="brown", probability = TRUE,
          xlab="waiting time of simlulated data",
          main="Histogram of waiting times with window of 4 with a smooth curve")
lines(h$mids,h$density,col="red",lwd=4)
```



Answer:

This question is almost the same as previous with the change that now 4 counters are open in the bank. Here, the calculations of the waiting times has been also explained through the table in Figure 2.

From the above calculations, it is found that the average waiting time in this case is 0.00336876906505022, which is very close to zero.

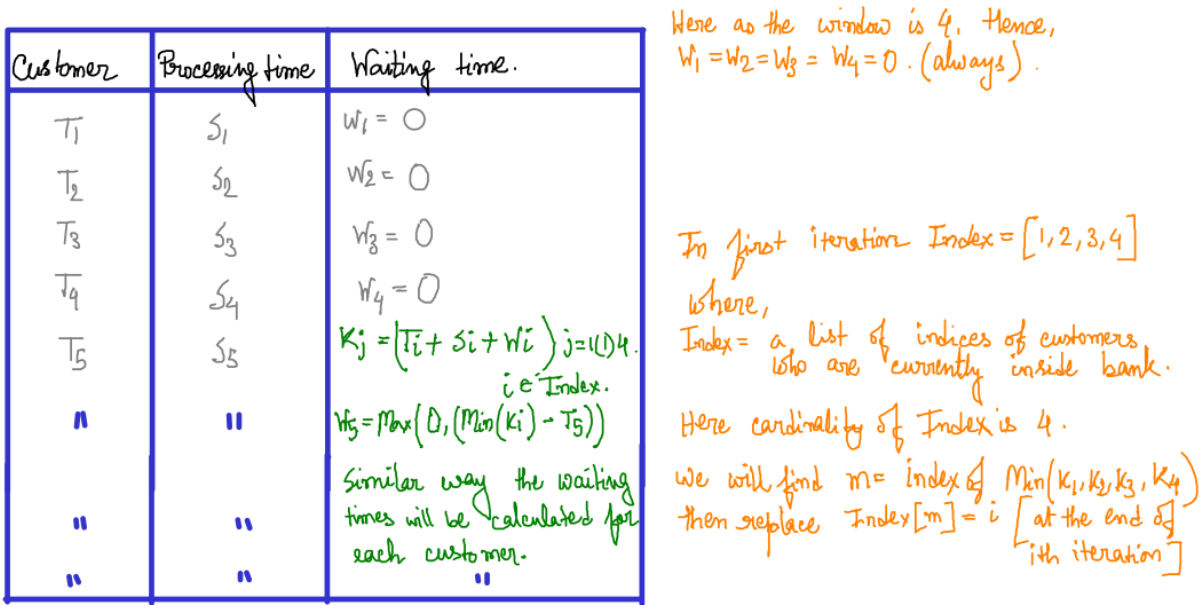


Figure 2: Part 4-c) solution.