Advanced Statistical Analysis Assignment 1

Minjoo Kim (Sungshin Women's University)

2025-10-21

Problem 1

Problem 1 - (a)

- 위 데이터를 data/ 경로에 csv로 저장하는 코드를 작성하시오.

```
#
set.seed(1)
n = 200
x = seq(0, 1, length.out = n)
y = sin(2*pi*x) + rnorm(n, sd = 0.15)

# (a) data/    csv
data1 <- data.frame(x = x, y = y)
getwd()
```

```
## [1] "C:/Users/  /Desktop/Advanced_Statistical_Analysis/assignment1/R"
```

```
write.csv(data1, "../../assignment1/data/problem1_data.csv")
```

Problem 1 - (b)

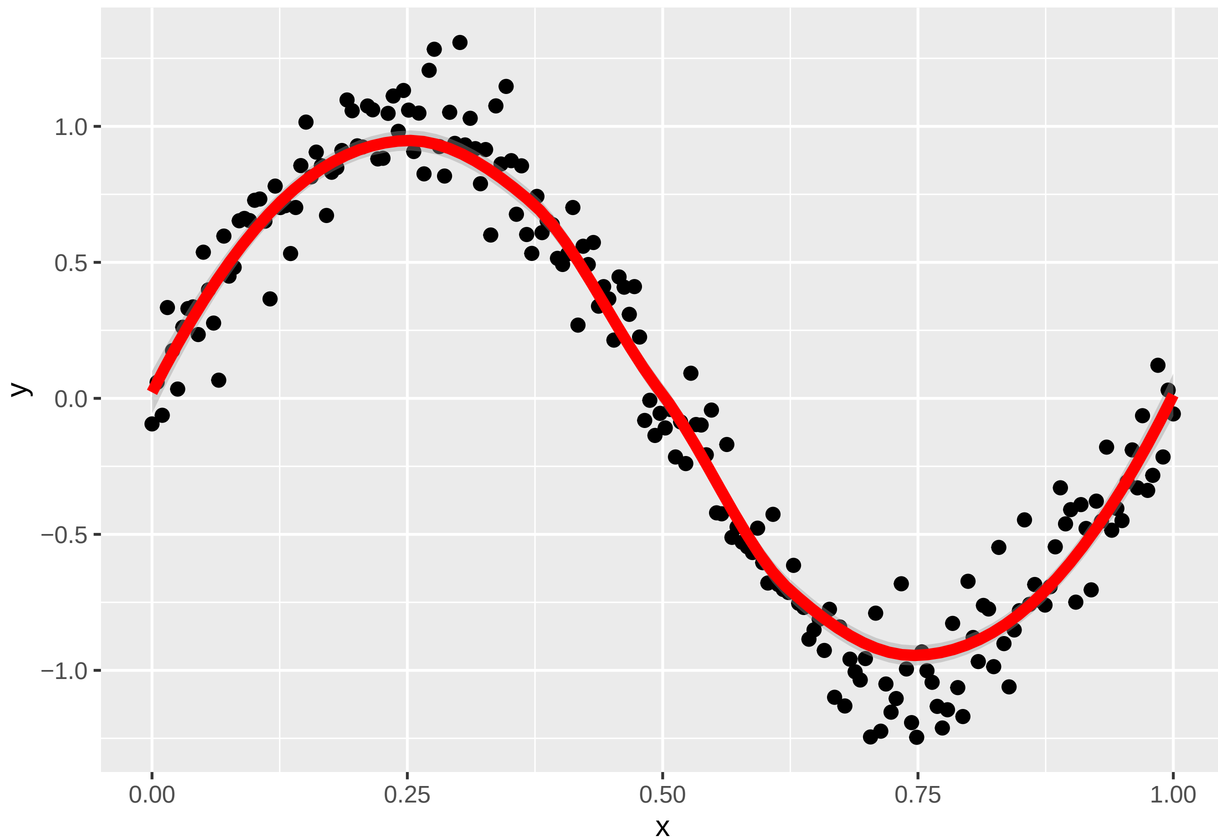
- R/ 폴더 아래 R 스크립트를 작성하여, 위에서 저장한 데이터를 불러오고, ggplot2를 바탕으로 산점도와 회귀곡선 적합을 시각화하시오 (ggplot2의 geom_smooth()에서 적절한method 선택)

```
data2 <- read.csv("../../assignment1/data/problem1_data.csv")

library(ggplot2)
plot1<- ggplot(data=data2, aes(x=x, y=y))+
  geom_point(size=2)+
  geom_smooth(color='red',size=2)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
plot1
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Problem 1 - (c)~(d)

- ggsave() 함수를 이용하여 작성한 플랏을 plots/폴더에 저장하는 코드를 작성하시오

```
ggsave(path = "../../assignment1/plots", filename = "problem1_plot.png")
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

- R에서 Git 버전 컨트롤을 사용하여 commit하고 본인의 GitHub에 올린 후 GitHub저장소 링크를 같이 제출하시오.
- https://github.com/minjoo0760/Advanced_Statistical_Analysis

Problem 2

Problem 2 - (a)

- 버블정렬을 R함수로 구현하고, 아래 테스트 케이스를 사용하여 결과를 확인하시오. 오름차순과 내림차순을 옵션으로 받을 수 있게 하여 두 경우 모두 결과를 제시하시오.

```r
bubble_sort <- function(x, decreasing = FALSE)
{
  n <- length(x)
  x_sorted <- x

  for (i in 1:(n-1)) {
    swapped <- FALSE
    for (j in 1:(n-i)) {
      if (decreasing) {
        condition <- x_sorted[j] < x_sorted[j+1] #
      } else {
        condition <- x_sorted[j] > x_sorted[j+1] #
      }
      if (condition) {
        temp <- x_sorted[j]
        x_sorted[j] <- x_sorted[j+1]
        x_sorted[j+1] <- temp
        swapped <- TRUE
      }
    }
    if (!swapped) break
  }
  return(x_sorted)
}

set.seed(1)
x <- runif(10)
print(x)
```

```
##  [1] 0.26550866 0.37212390 0.57285336 0.90820779 0.20168193 0.89838968
##  [7] 0.94467527 0.66079779 0.62911404 0.06178627
```

```r
#
print(bubble_sort(x, decreasing = FALSE))
```

```
##  [1] 0.06178627 0.20168193 0.26550866 0.37212390 0.57285336 0.62911404
##  [7] 0.66079779 0.89838968 0.90820779 0.94467527
```

```r
#
print(bubble_sort(x, decreasing = TRUE))
```

```
##  [1] 0.94467527 0.90820779 0.89838968 0.66079779 0.62911404 0.57285336
##  [7] 0.37212390 0.26550866 0.20168193 0.06178627
```

Problem 2 - (b)

- 퀵정렬을 R함수로 구현하고, 아래 테스트 케이스를 사용하여 결과를 확인하시오. 오름차순과 내림차순을 옵션으로 받을 수 있게 하여 두 경우 모두 결과를 제시하시오

```
quick_sort <- function(x, decreasing = FALSE)
{
  if (length(x) <= 1) {
    return(x)
  }
  pivot_idx <- ceiling(length(x) / 2)
  pivot <- x[pivot_idx]
  rest <- x[-pivot_idx]
  if (decreasing) {
    left <- rest[rest >= pivot] #
    right <- rest[rest < pivot]
  } else {
    left <- rest[rest < pivot] #
    right <- rest[rest >= pivot]
  }
  return(c(quick_sort(left, decreasing), pivot, quick_sort(right, decreasing)))
}

set.seed(1)
x <- runif(10)
print(x)
```

```
##  [1] 0.26550866 0.37212390 0.57285336 0.90820779 0.20168193 0.89838968
##  [7] 0.94467527 0.66079779 0.62911404 0.06178627
```

```
#
print(quick_sort(x, decreasing = FALSE))
```

```
##  [1] 0.06178627 0.20168193 0.26550866 0.37212390 0.57285336 0.62911404
##  [7] 0.66079779 0.89838968 0.90820779 0.94467527
```

```
#
print(quick_sort(x, decreasing = TRUE))
```

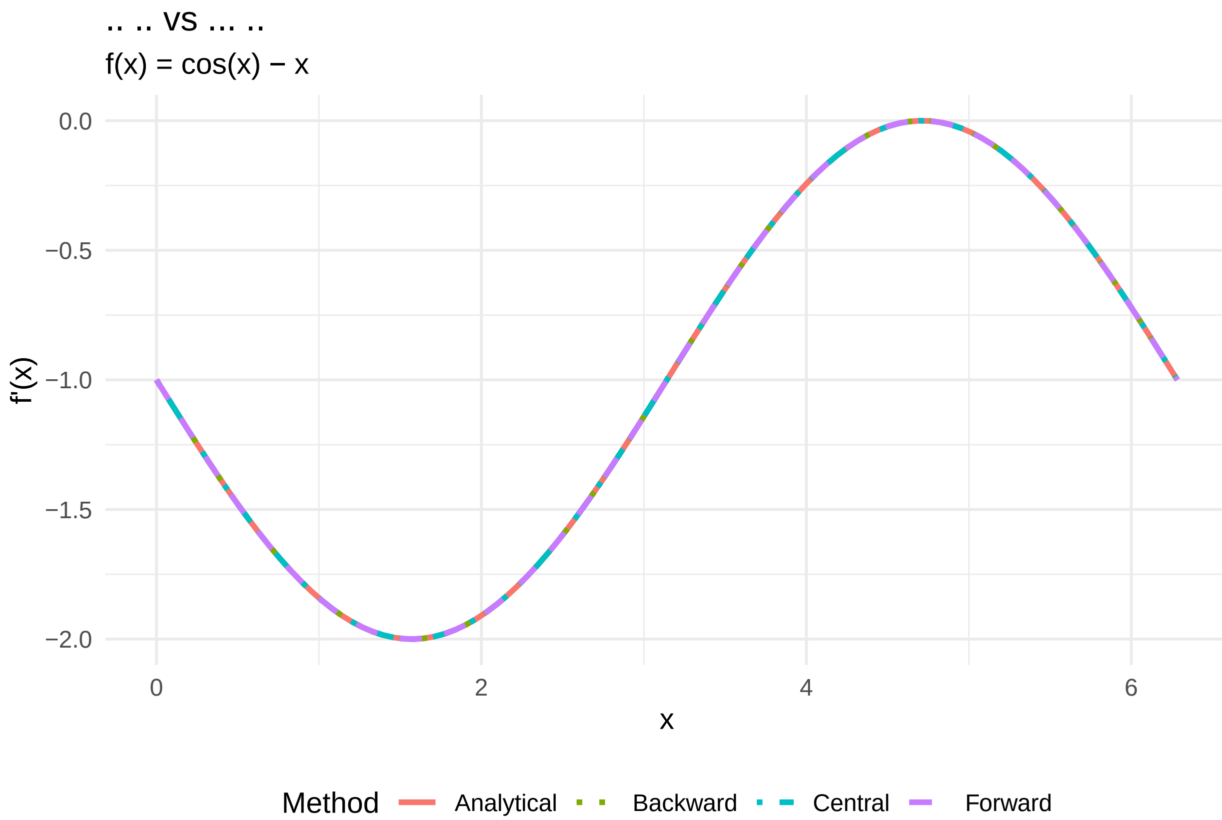
```
##  [1] 0.94467527 0.90820779 0.89838968 0.66079779 0.62911404 0.57285336
##  [7] 0.37212390 0.26550866 0.20168193 0.06178627
```

Problem 3

Problem 3 - (a)

- 수치 미분 함수 구현

```r
numerical_derivative <- function(f, x, h = 1e-6, method = "central")
{
  if (method == "forward") {
    #
    return((f(x + h) - f(x)) / h)
  } else if (method == "backward") {
    #
    return((f(x) - f(x - h)) / h)
  } else if (method == "central") {
    #
    return((f(x + h) - f(x - h)) / (2 * h))
  } else {
    stop('method "forward", "backward", "central"    ')
  }
}

f <- function(x) cos(x) - x
f_prime <- function(x) -sin(x) - 1

x_vals <- seq(0, 2*pi, length.out = 100)

deriv_forward <- sapply(x_vals, function(x) numerical_derivative(f, x, method = "forward"))
deriv_backward <- sapply(x_vals, function(x) numerical_derivative(f, x, method = "backward"))
deriv_central <- sapply(x_vals, function(x) numerical_derivative(f, x, method = "central"))
deriv_analytical <- f_prime(x_vals)

#
library(ggplot2)
df <- data.frame(
  x = rep(x_vals, 4),
  derivative = c(deriv_analytical, deriv_forward, deriv_backward, deriv_central),
  method = rep(c("Analytical", "Forward", "Backward", "Central"), each = length(x_vals))
)
ggplot(df, aes(x = x, y = derivative, color = method, linetype = method)) +
  geom_line(linewidth = 1) +
  scale_linetype_manual(values = c("Analytical" = "solid", "Forward" = "dashed",
                                   "Backward" = "dotted", "Central" = "dotdash")) +
  labs(title = "    vs     ",
       subtitle = "f(x) = cos(x) - x",
       x = "x", y = "f'(x)",
       color = "Method", linetype = "Method") +
  theme_minimal() +
  theme(legend.position = "bottom")
```




```r
cat("Forward  :", mean(abs(deriv_forward - deriv_analytical)), "\n")
```

```
## Forward  : 3.200912e-07
```

```r
cat("Backward  :", mean(abs(deriv_backward - deriv_analytical)), "\n")
```

```
## Backward  : 3.201633e-07
```

```r
cat("Central  :", mean(abs(deriv_central - deriv_analytical)), "\n")
```

```
## Central  : 1.418127e-10
```

Problem 3 - (b)

- Newton-Raphson 방법 구현

```r
newton_raphson <- function(f, x0, fprime = NULL, maxiter = 100,
                           h = 1e-6, epsilon = 1e-10)
{
  x <- x0
  for (i in 1:maxiter) {
    #
```

```r
    if (is.null(fprime)) {
      #
      fp <- numerical_derivative(f, x, h = h, method = "central")
    } else {
      fp <- fprime(x)
    }

    x_new <- x - f(x) / fp

    #
    if (abs(x_new - x) < epsilon) {
      cat("    ,      :", i, "\n")
      cat(" :", x_new, "\n")
      cat("f(x):", f(x_new), "\n")
      return(list(root = x_new, iterations = i, f_value = f(x_new)))
    }
    x <- x_new
  }
  return(list(root = x, iterations = maxiter, f_value = f(x)))
}
```

Problem 3 - (c)

```r
f <- function(x) cos(x) - x
fprime <- function(x) -sin(x) - 1
x0 <- 0.5

#
newton_raphson(f, x0, fprime = NULL, h = 1e-6, epsilon = 1e-10)
```

```
##    ,     : 5
##  : 0.7390851
## f(x): 0
```

```
## $root
## [1] 0.7390851
##
## $iterations
## [1] 5
##
## $f_value
## [1] 0
```

```r
#
newton_raphson(f, x0, fprime = fprime, epsilon = 1e-10)
```

```
##    ,     : 5
##  : 0.7390851
## f(x): 0
```

```
## $root
## [1] 0.7390851
##
## $iterations
## [1] 5
##
## $f_value
## [1] 0
```

Problem 4

Problem 4 - (a)

- Left Rectangle 방식을 R코드로 구현하시오. 함수의 인자로 적분 대상 함수 f와 적분 구간 a, b, 그리고 n을 입력받는다.

```r
integrate_left_rectangle <- function(f, a, b, n)
{
  h <- (b - a) / n
  x <- seq(a, b - h, length.out = n)
  f_vals <- f(x)
  return(h * sum(f_vals))
}
```

Problem 4 - (b)

- Trapezoid 방식을 R코드로 구현하시오. 함수의 인자는 위와 같다.

```r
integrate_trapezoid <- function(f, a, b, n)
{
  h <- (b - a) / n
  x <- seq(a, b, length.out = n + 1)
  f_vals <- f(x)
  result <- h / 2 * (f_vals[1] + 2 * sum(f_vals[2:n]) + f_vals[n + 1])
  return(result)
}
```

Problem 4 - (c)

- Simpson 방식을 R코드로 구현하시오. 함수의 인자는 위와 같다.

```r
integrate_simpson <- function(f, a, b, n)
{
  if (n %% 2 != 0) {
    stop("n    .")
  }

  h <- (b - a) / n
```

```r
  x <- seq(a, b, length.out = n + 1)
  f_vals <- f(x)
  #
  odd_indices <- seq(2, n, by = 2)
  odd_sum <- sum(f_vals[odd_indices])
  #
  even_indices <- seq(3, n, by = 2)
  even_sum <- if(length(even_indices) > 0) sum(f_vals[even_indices]) else 0
  # Simpson
  result <- h / 3 * (f_vals[1] + 4 * odd_sum + 2 * even_sum + f_vals[n + 1])
  return(result)
}
```

Problem 4 - (d)

- sin(x) 함수를 [0,⊠] 구간에서 적분한 값을 세 개의 알고리즘으로 계산하시오. (n = 100으로 설정한다.)

```r
f <- function(x) sin(x)
n <- 100

integrate_left_rectangle(f, 0, pi, n)
```

```
## [1] 1.999836
```

```r
integrate_trapezoid(f, 0, pi, n)
```

```
## [1] 1.999836
```

```r
integrate_simpson(f, 0, pi, n)
```

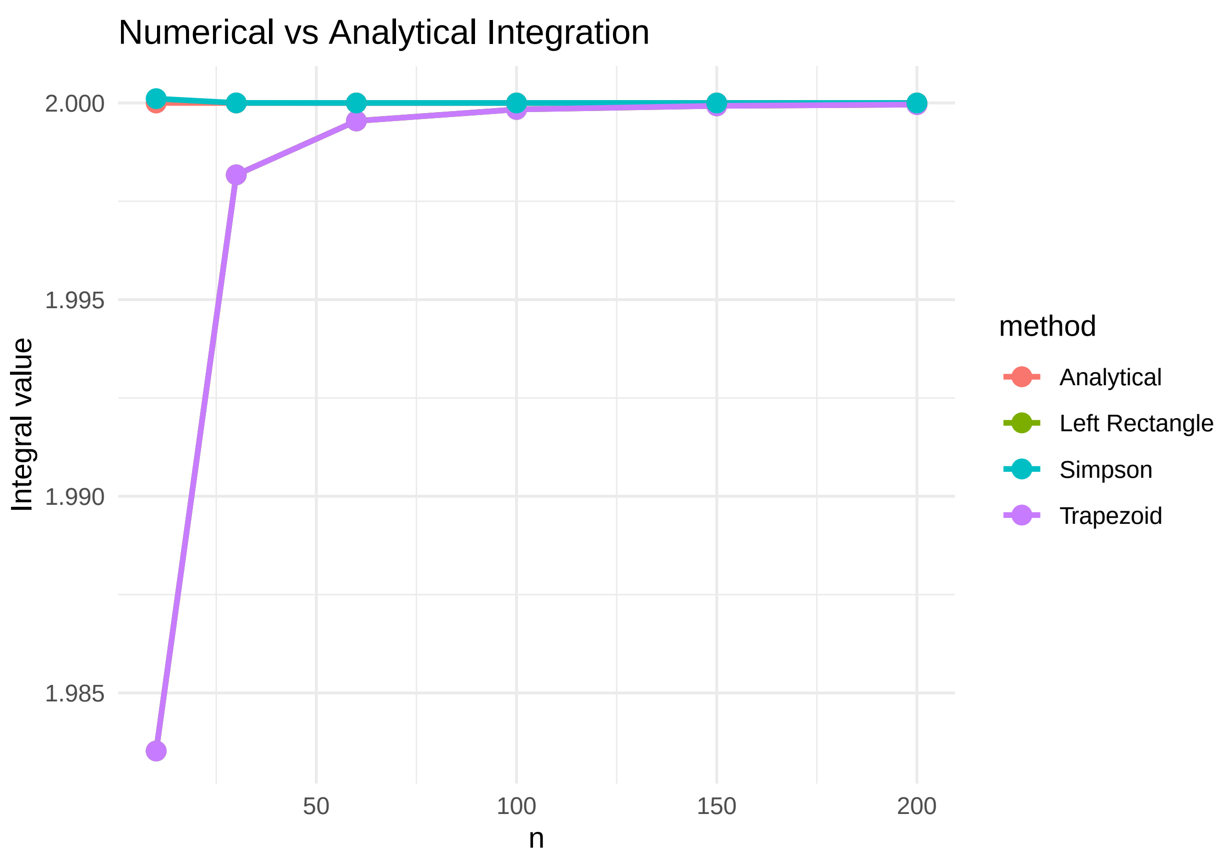
```
## [1] 2
```

Problem 4 - (e)

- 해석적으로 구한 값과 위 알고리즘의 차이를 ⊠=10, 30, 60, 100, 150, 200에 대해 계산하고 알고리즘 비교를 위한 시각화를 수행하시오.

```r
library(ggplot2)

n_values <- c(10, 30, 60, 100, 150, 200)

#
left_results <- sapply(n_values, function(n) integrate_left_rectangle(f, 0, pi, n))
trap_results <- sapply(n_values, function(n) integrate_trapezoid(f, 0, pi, n))
simp_results <- sapply(n_values, function(n) integrate_simpson(f, 0, pi, n))

#       (       )
results <- data.frame(
```

```r
  n = rep(n_values, 4),
  value = c(rep(2, length(n_values)), left_results, trap_results, simp_results),
  method = rep(c("Analytical", "Left Rectangle", "Trapezoid", "Simpson"), each = length(n_values))
)

#
ggplot(results, aes(x = n, y = value, color = method)) +
  geom_line(linewidth = 1) +
  geom_point(size = 3) +
  labs(title = "Numerical vs Analytical Integration",
       x = "n", y = "Integral value") +
  theme_minimal()
```



Problem 5

Problem 5 - (a)

- 아래 행렬 ☒에 해당하는 ☒을 구하고, $LL^\top$을 출력하여 ☒와 같음을 확인하시오. (roundingerror허용)

```r
A <- matrix(c(4,2,2,2,5,1,2,1,3), 3)
U <- chol(A)
L <- t(U)
```

```r
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    4    2    2
## [2,]    2    5    1
## [3,]    2    1    3
```

```r
print(L %*% t(L))
```

```
##      [,1] [,2] [,3]
## [1,]    4    2    2
## [2,]    2    5    1
## [3,]    2    1    3
```

Problem 5 - (b)

- 식을 풀기 위한 forward 함수를 작성하시오. (힌트: R의 forwardsovle 함수와 결과를 비교할 수 있다.)

```r
forward_solve <- function(L, b)
{
  n <- length(b)
  z <- numeric(n)
  for (i in 1:n) {
    sum_val <- 0
    if (i > 1) {
      sum_val <- sum(L[i, 1:(i-1)] * z[1:(i-1)])
    }
    z[i] <- (b[i] - sum_val) / L[i, i]
  }
  return(z)
}

test <- c(1, 2, 3)
forward_solve(L, test)
```

```
## [1] 0.500000 0.750000 1.767767
```

```r
print(forwardsolve(L, test))
```

```
## [1] 0.500000 0.750000 1.767767
```

Problem 5 - (c)

- 식을 풀기 위한 backward 함수를 작성하시오. (힌트: R의 backwardsolve 함수와 결과를 비교할 수 있다.)

```r
backward_solve <- function(L, z)
{
  n <- length(z)
  x <- numeric(n)
  for (i in n:1) {
    sum_val <- 0
    if (i < n) {
      sum_val <- sum(L[(i+1):n, i] * x[(i+1):n])
    }
    x[i] <- (z[i] - sum_val) / L[i, i]
  }
  return(x)
}

test <- c(1, 2, 3)
backward_solve(t(L), test)
```

```
## [1] 0.50000 1.00000 2.12132
```

```r
print(backsolve(t(L), test))
```

```
## [1] -1.06066  1.00000  2.12132
```

Problem 5 - (d)

- (b)에서 작성한 forward() 함수와 (c)에서 작성한 backward() 함수를 이용하여, 다음 벡터에 대해선 형방정식($Ax = b$)를 푸시오.

```r
b <- c(1, -2, 3)

z <- forward_solve(L, b)
x <- backward_solve(t(L), z)
print(x)
```

```
## [1]  0.250 -0.625  1.250
```

```r
x_solve <- solve(A, b)
print(x_solve)
```

```
## [1] -0.0625 -0.6250  1.2500
```

Problem 6

Problem 6 - (a)

- Gaussian kernel 함수를 계산하는 R 함수를 작성하시오. 이때 $\rho = 1$을 default로 사용한다

```r
gaussian_kernel <- function(x, xprime, rho = 1)
{
  exp(-rho * (x - xprime)^2)
}
```

Problem 6 - (b)

- KRR을 적합하는 함수를 작성하시오.

```r
fit_krr <- function(X, y, lambda = 0.0001, rho = 1)
{
  n <- length(y)

  K <- matrix(0, n, n)
  for (i in 1:n) {
    for (j in 1:n) {
      K[i, j] <- gaussian_kernel(X[i], X[j], rho)
    }
  }

  alpha <- solve(K + lambda * diag(n), y)

  result <- list(
    X = X,
    y = y,
    alpha = alpha,
    lambda = lambda,
    rho = rho,
    K = K
  )

  class(result) <- "krr"
  return(result)
}
```

Problem 6 - (c)

- predict 함수를 krr 클래스에 대해 확장하 KRR 예측값을 계산해주는 함수를 작성하시오.

```r
predict.krr <- function(object, newdata, ...)
{
  n_train <- length(object$X)
  n_new <- length(newdata)
  predictions <- numeric(n_new)

  for (i in 1:n_new) {
    k_vec <- sapply(object$X, function(x) {
      gaussian_kernel(newdata[i], x, object$rho)
    })
    predictions[i] <- sum(k_vec * object$alpha)
```

```
  }
  return(predictions)
}
```

Problem 6 - (d)

- plot 함수를 krr 클래스에 대해 확장하여 데이터의 산점도와 예측함수 $f(x)$를 시각화하는 함수를 작성하시오.

```
plot.krr <- function(x, ...)
{
  library(ggplot2)
  x_plot <- seq(min(x$X), max(x$X), length.out = 200)
  y_pred <- predict(x, x_plot)

  df_data <- data.frame(x = x$X, y = x$y, type = "Data")
  df_pred <- data.frame(x = x_plot, y = y_pred, type = "Prediction")

  p <- ggplot() +
    geom_point(data = df_data, aes(x = x, y = y),
               color = "black", size = 2) +
    geom_line(data = df_pred, aes(x = x, y = y),
              color = "red", linewidth = 1) +
    labs(title = "Kernel Ridge Regression",
         x = "x", y = "y") +
    theme_minimal()

  print(p)
}
```

Problem 6 - (e)

- 아래의 데이터를 시뮬레이션하고, KRR을 적합한 이후 predict와 plot함수를 통해 결과를 시각화하시오

```
set.seed(1)
n = 150
X = matrix(runif(n,-1, 1), ncol = 1)
ftrue = function(x) sin(2*pi*x) + 0.5*cos(4*pi*x)
y = ftrue(X[,1]) + rnorm(n, sd = 0.1)

#
model <- fit_krr(X[,1], y, lambda = 0.0001, rho = 1)

# predict
#    new data
x_test <- seq(-1, 1, length.out = 10)
y_pred <- predict(model, x_test)

# plot
plot(model)
```

Kernel Ridge Regression