Assignment 1

20210368 이민서

2025-10-21

Problem 1

- GitHub Link : https://github.com/minseo-02/Project_Route.git

Problem 2

2-(a)

```
set.seed(1)
x = runif(10)
```

```
bubble_sort = function(vec, order = "asc") {

  n = length(vec)

  for (i in 1:(n-1))
    {
    for (j in 1:(n-i))
      {
      if ((order == "asc" && vec[j] > vec[j + 1]) ||
          (order == "desc" && vec[j] < vec[j + 1]))
        {
        temp = vec[j]
        vec[j] = vec[j + 1]
        vec[j + 1] = temp
      }
    }
  }
  return(vec)
}
```

```
bubble_sort(x)
```

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

```r
bubble_sort(x, order = "desc")
```

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

2-(b)

```r
quick_sort <- function(x, order = "asc") {

  if (length(x) <= 1) return(x)

  a <- x[1]
  b <- x[ceiling(length(x) / 2)]
  c <- x[length(x)]
  pivot <- median(c(a, b, c))

  if (order == "asc") {
    left  <- x[x < pivot]
    mid   <- x[x == pivot]
    right <- x[x > pivot]

    return(c(quick_sort(left, "asc"), mid, quick_sort(right, "asc")))

  } else if (order == "desc") {
    left  <- x[x > pivot]
    mid   <- x[x == pivot]
    right <- x[x < pivot]

    return(c(quick_sort(left, "desc"), mid, quick_sort(right, "desc")))

  } else {
    stop("order must be either 'asc' or 'desc'")
  }
}
```

```r
quick_sort(x, order = "asc")
```

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

```r
quick_sort(x, order = "desc")
```

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

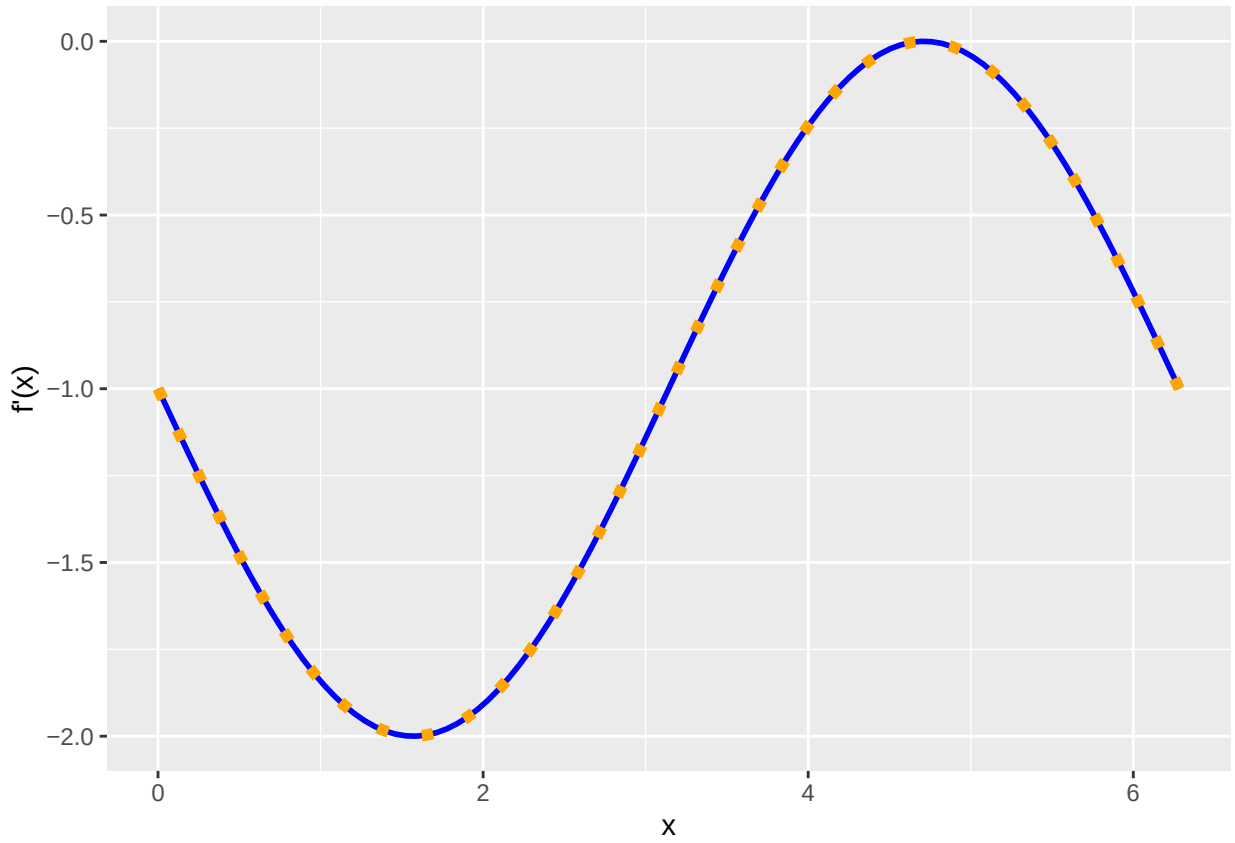
Problem 3

3-(a)

```
numerical_derivative <- function(f, x, h = 1e-6, method = c("forward", "backward", "central"))
  {
  method <- match.arg(method)

  if (method == "forward") {
    (f(x+h) - f(x)) / h
  } else if (method == "backward") {
    (f(x) - f(x-h)) / h
  } else {
    (f(x+h) - f(x-h)) / (2*h)
  }
}
```

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

x_problem3 <- seq(0, 2*pi, length.out = 100)

f_num <- numerical_derivative(f, x_problem3, h = 1e-6, method = "central")
f_prime_num <- f_prime(x_problem3)

problem3_df <- data.frame(x = x_problem3,
                          analysis = f_prime_num,
                          num = f_num)
```

```
library(ggplot2)

problem3_a_plot <- ggplot(problem3_df, aes(x)) +
  geom_line(aes(y = analysis), linewidth = 1, color = "blue") +
  geom_line(aes(y = num), linetype = 3, linewidth = 2, color = "orange") +
  labs(y = "f'(x)")

print(problem3_a_plot)
```



3-(b)

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

  for (t in 1:maxiter) {
    g <- if(is.null(fprime)) {
      (f(x+h) - f(x-h)) / (2*h)
    } else {
      fprime(x)
    }

    x_new <- x - f(x) / g

    if (abs(x_new - x) < epsilon) {
      return(list(root = x_new, iter = t, converged = TRUE))
    }

    x <- x_new

  }
```

```
  return(list(root = x, iter = maxiter, converged = FALSE))

}
```

3-(c)

```
f <- function(x) cos(x) - x
f_prime <- function(x) -sin(x) - 1
x0 = 0.5
```

```
cos_analysis <- newton_raphson(f, fprime = f_prime, x0 = x0,
                               maxiter = 100, h = 1e-6, epsilon = 1e-10)

cos_num <- newton_raphson(f, fprime = NULL, x0 = x0,
                               maxiter = 100, h = 1e-6, epsilon = 1e-10)
```

```
print(cos_analysis)
```

```
## $root
## [1] 0.7390851
##
## $iter
## [1] 5
##
## $converged
## [1] TRUE
```

```
print(cos_num)
```

```
## $root
## [1] 0.7390851
##
## $iter
## [1] 5
##
## $converged
## [1] TRUE
```

Problem 4

4-(a)

```
left_rectangle <- function(f, a, b, n)
{
  h <- (b-a) / n
  x_i <- a + h * (0:(n-1))
```

```
  area <- h * sum(f(x_i))
  return(area)
}
```

4-(b)

```
trapezoid <- function(f, a, b, n)
{
  h <- (b-a) / n
  x_i <- a + h * (0:n)
  f_i <- f(x_i)
  area <- h * (0.5 * f_i[1] + sum(f_i[2:n]) + 0.5* f_i[n+1])
  return(area)
}
```

4-(c)

```
simpson <- function(f, a, b, n)
{
  if (n %% 2 == 1) {
    n <- n+1
  }

  h <- (b-a) / n
  x_i <- a + h * (0:n)
  f_i <- f(x_i)

  odd_idx <- seq(2, n, by=2)
  even_idx <- seq(3, n-1, by=2)
  area <- (h/3) * (f_i[1] + 4*sum(f_i[odd_idx]) + 2*sum(f_i[even_idx]) + f_i[n+1])
  return(area)
}
```

4-(d)

```
f_problem4 <- function(x) sin(x)
a <- 0; b <- pi; n <- 100

left_val <- left_rectangle(f = f_problem4, a, b, n)
trap_val <- trapezoid(f = f_problem4, a, b, n)
simp_val <- simpson(f = f_problem4, a, b, n)

left_val
```

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

```
trap_val
```

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

```
simp_val
```

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

4-(e)

```
true_val <- -cos(b) + cos(a)
true_val
```

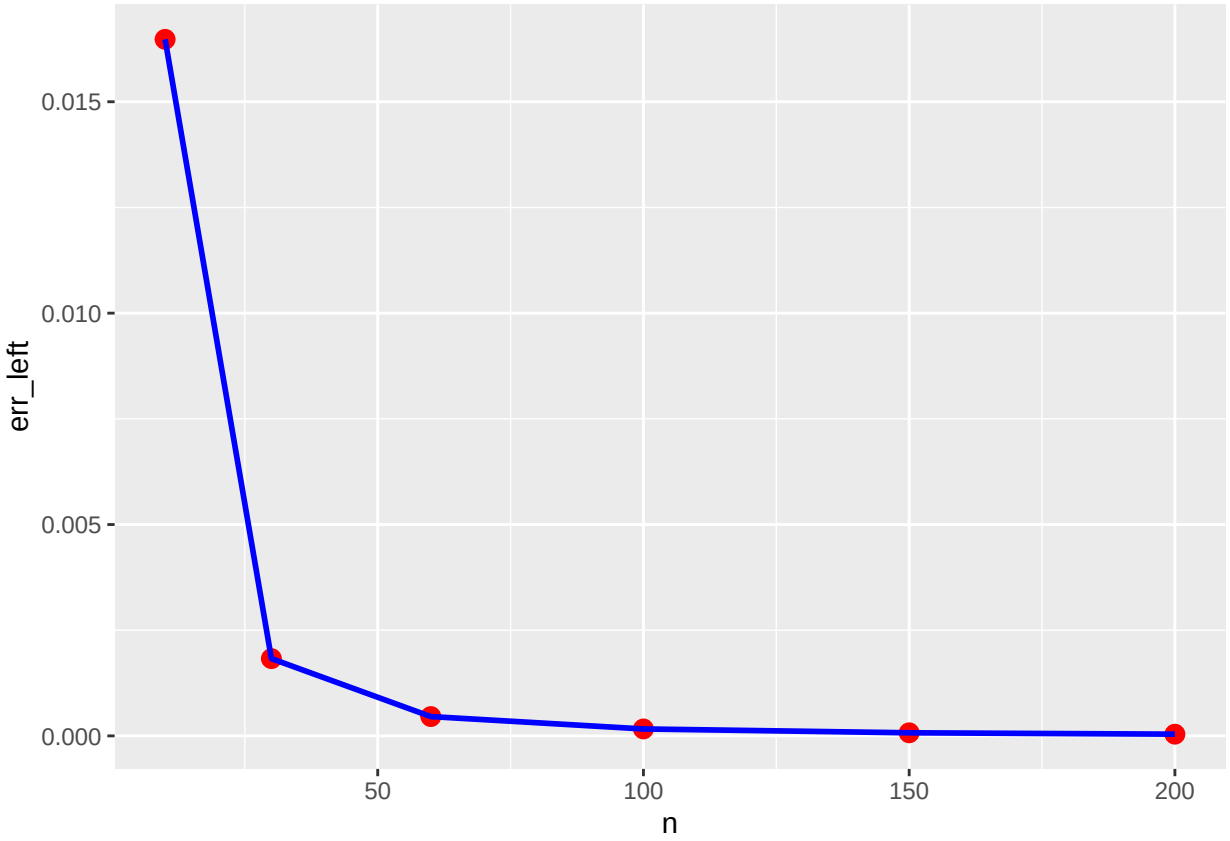
```
## [1] 2
```

```
num_problem4 <- c(10, 30, 60, 100, 150, 200)

left <- numeric(length(num_problem4))
trap <- numeric(length(num_problem4))
simp <- numeric(length(num_problem4))

for (i in 1:length(num_problem4)) {
  nn <- num_problem4[i]
  left[i] <- left_rectangle(f_problem4, a, b, nn)
  trap[i] <- trapezoid(f_problem4, a, b, nn)
  simp[i] <- simpson(f_problem4, a, b, nn)
}

err_left <- abs(left - true_val)
err_trap <- abs(trap - true_val)
err_simp <- abs(simp - true_val)
```

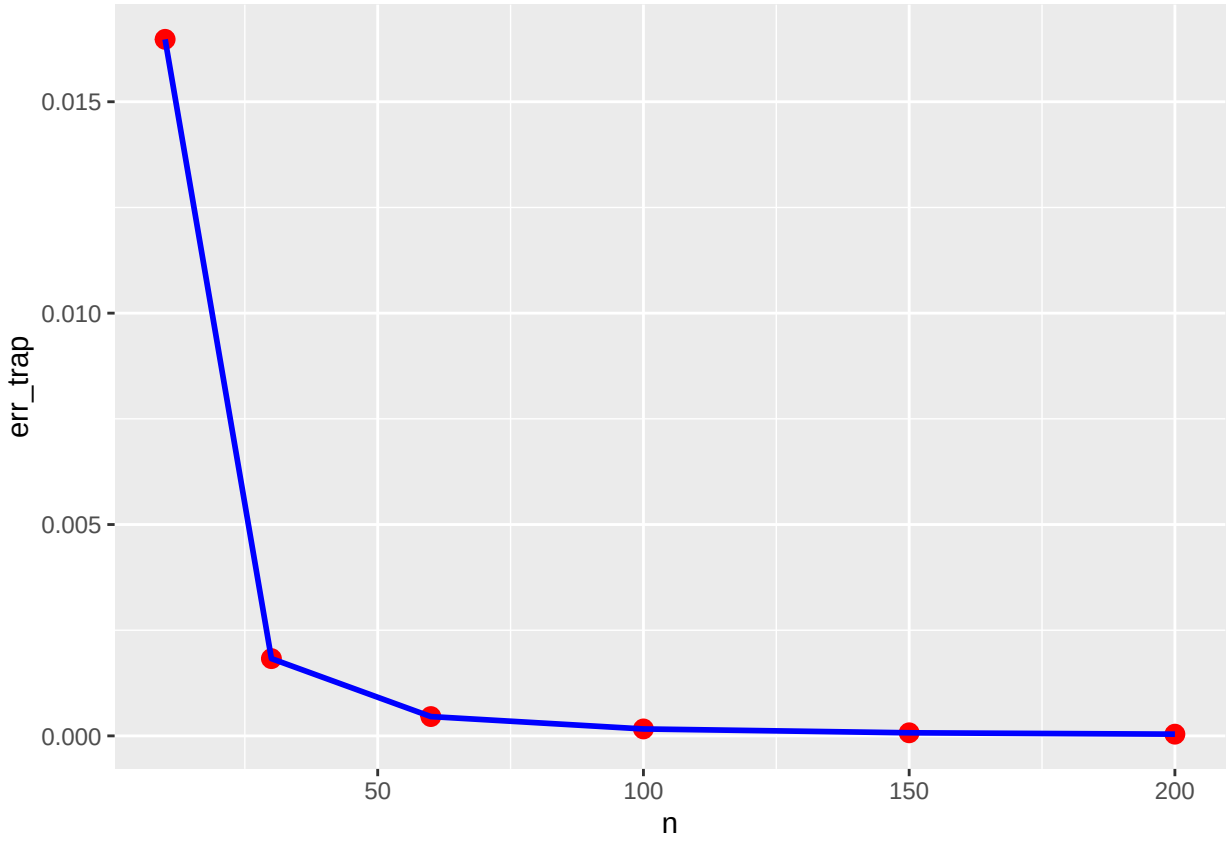
```
df_problem4 <- data.frame(n = num_problem4,
                          Left = left, Trap = trap, Simp = simp,
                          Left_err = err_left, Trap_err = err_trap, Simp_err = err_simp)

df_problem4
```

```
##     n     Left     Trap     Simp     Left_err     Trap_err     Simp_err
## 1  10 1.983524 1.983524 2.000110 1.647646e-02 1.647646e-02 1.095173e-04
## 2  30 1.998172 1.998172 2.000001 1.828039e-03 1.828039e-03 1.337948e-06
## 3  60 1.999543 1.999543 2.000000 4.569470e-04 4.569470e-04 8.353986e-08
## 4 100 1.999836 1.999836 2.000000 1.644961e-04 1.644961e-04 1.082450e-08
## 5 150 1.999927 1.999927 2.000000 7.310872e-05 7.310872e-05 2.138034e-09
## 6 200 1.999959 1.999959 2.000000 4.112352e-05 4.112352e-05 6.764722e-10
```

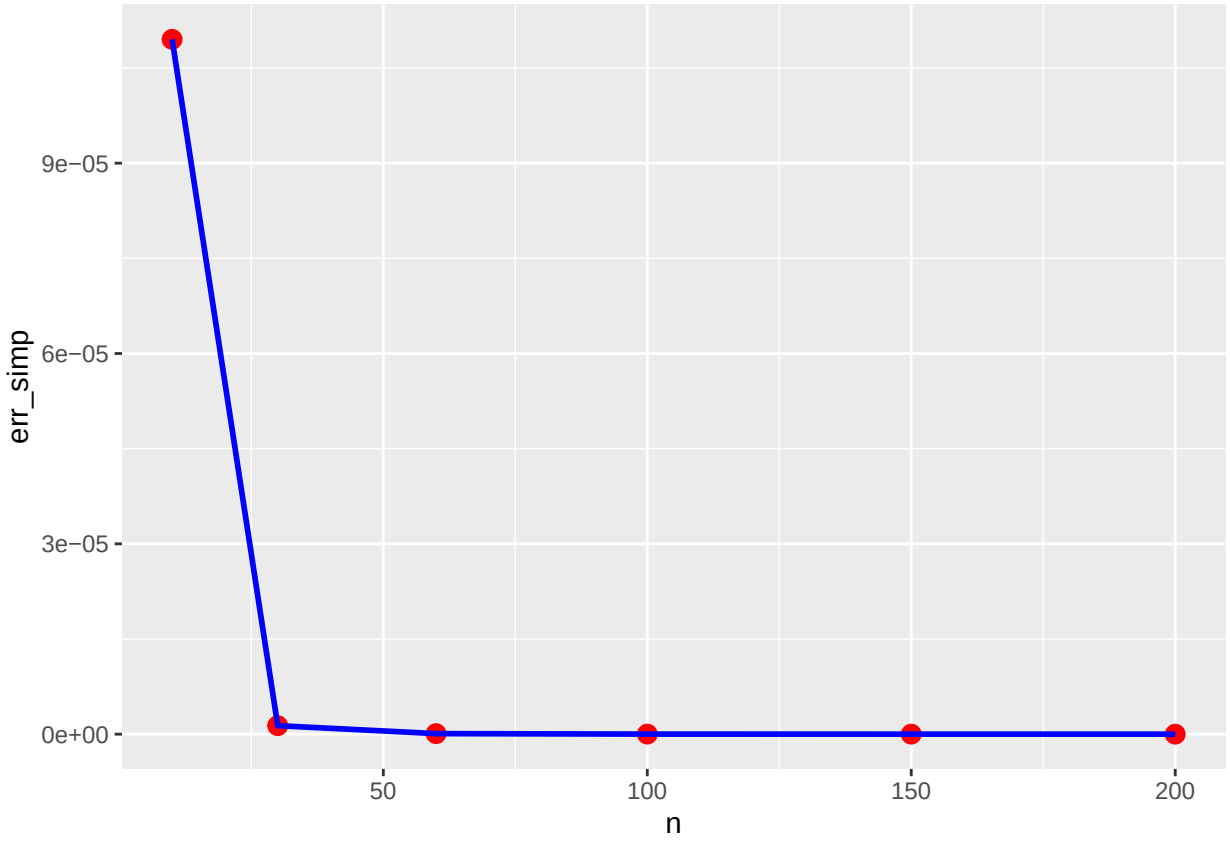
```
ggplot(df_problem4, aes(x = n, y = err_left)) +
  geom_point(color = "red", size = 3) +
  geom_line(color = "blue", linewidth = 1)
```




```
ggplot(df_problem4, aes(x = n, y = err_trap)) +
  geom_point(color = "red", size = 3) +
  geom_line(color = "blue", linewidth = 1)
```




```
ggplot(df_problem4, aes(x = n, y = err_simp)) +
  geom_point(color = "red", size = 3) +
  geom_line(color = "blue", linewidth = 1)
```



Problem 5

5-(a)

```
A = matrix(c(4,2,2,2,5,1,2,1,3), 3)
A
```

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

```
U <- chol(A)
L <- t(U)

L %*% t(L)
```

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

```r
all.equal(A, L %*% t(L))
```

```
## [1] TRUE
```

5-(b)

```r
forward <- function(L, b)
{
  n <- length(b)
  z <- numeric(n)

  for (i in 1:n) {
    if (i == 1) {
      s <- 0
    } else {
      s <- sum(L[i, 1:(i-1)] * z[1:(i-1)])
    }
    z[i] <- (b[i] - s) / L[i, i]
  }
  return(z)
}
```

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

z <- forward(L, b)
z
```

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

```r
forwardsolve(L, b)
```

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

```r
all.equal(z, forwardsolve(L, b))
```

```
## [1] TRUE
```

5-(c)

- $L = t(U)$ 이므로 $t(L)$ 대신 U를 사용했습니다.

```r
backward <- function(U, z)
{
  n <- length(z)
  x <- numeric(n)

  for (i in n:1) {
    if (i == n) {
```

```
      s <- 0
    } else {
      s <- sum(U[i, (i+1):n] * x[(i+1):n])
    }
    x[i] <- (z[i] - s) / U[i, i]
  }
  return(x)
}
```

```
backward_result_x <- backward(t(L), z)
backward_result_x
```

```
## [1] -0.5625  0.3750  1.2500
```

```
backsolve(t(L), z)
```

```
## [1] -0.5625  0.3750  1.2500
```

```
all.equal(backward_result_x, backsolve(t(L), z))
```

```
## [1] TRUE
```

5-(d)

```
b_pro5_d <- c(1, -2, 3)

z_d <- forward(L, b_pro5_d)
x_d <- backward(t(L), z_d)
x_d
```

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

```
x_solve <- solve(A, b_pro5_d)
x_solve
```

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

```
all.equal(x_d, x_solve)
```

```
## [1] TRUE
```

Problem 6

6-(a)

```r
gaussian_kernel <- function(x, x_prime, rho = 1)
{
  diff <- x - x_prime
  kernel_val <- exp(-rho * sum(diff^2))
  return(kernel_val)
}
```

```r
gaussian_kernel_matrix <- function(X, Y = NULL, rho = 1)
{
  X <- as.matrix(X)

  if (is.null(Y))
    Y <- X
  else Y <- as.matrix(Y)


  n <- nrow(X); m <- nrow(Y)
  K <- matrix(0, n, m)

  for (i in 1:n)
  {
    for (j in 1:m)
    {
      diff <- X[i,] - Y[j,]
      K[i,j] <- exp(-rho * sum(diff^2))
    }
  }

  return(K)
}
```

6-(b)

```r
KRR <- function(X, y, lambda = 1e-4, rho = 1)
{
  X <- as.matrix(X)
  y <- as.numeric(y)
  n <- nrow(X)

  K <- gaussian_kernel_matrix(X, rho = rho)
  K_reg <- K + lambda + diag(n)

  alpha <- as.numeric(solve(K_reg, y))

  fitted <- as.numeric(K %*% alpha)

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

```
                fitted = fitted)

  class(result) <- "krr"

  return(result)

}
```

6-(c)

```
predict.krr <- function(object, newdata = NULL, ...)
{
  if (is.null(newdata)) {
    return(object$fitted)
  }

  newX <- as.matrix(newdata)

  K_new <- gaussian_kernel_matrix(newX, object$X, rho = object$rho)

  as.numeric(K_new %*% object$alpha)
}
```

6-(d)

```
plot_krr <- function(x, xlim = NULL, ngrid = 400,
                     col_points = "gray40", pch = 16, cex = 0.9,
                     col_line = "steelblue", lty = 1, lwd = 2, ...)
{
  X <- x$X

  xr <- if (is.null(xlim)) range(X[,1]) else xlim
  grid <- seq(xr[1], xr[2], length.out = ngrid)
  yhat <- predict(x, newdata = matrix(grid, ncol = 1))

  plot(X[,1], x$y, col = col_points, pch = pch, cex = cex,
       xlab = "x", ylab = "y / f(x)", ...)

  matlines(x = grid, y = cbind(yhat),
           lty = lty, lwd = lwd, col = col_line)
}
```

6-(e)

```
set.seed(1)
n = 150
X = matrix(runif(n,-1, 1), ncol = 1)
```

```r
ftrue = function(x) sin(2*pi*x) + 0.5*cos(4*pi*x)
y = ftrue(X[,1]) + rnorm(n, sd = 0.1)
```

```r
fit <- KRR(X = X, y = y, lambda = 1e-4, rho = 1)

x_grid <- matrix(seq(-1, 1, length.out = 400), ncol = 1)
y_hat  <- predict(fit, newdata = x_grid)
y_true <- ftrue(x_grid[,1])
```

```r
plot_krr(fit, xlim = c(-1, 1), ngrid = 400,
        col_points = "gray55", pch = 16, cex = 0.9,
        col_line = "steelblue", lty = 1, lwd = 2,
        main = "KRR fit (Gaussian)")
```