HW01

KimHyewon

2025-10-15

Problem 01

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

(a)

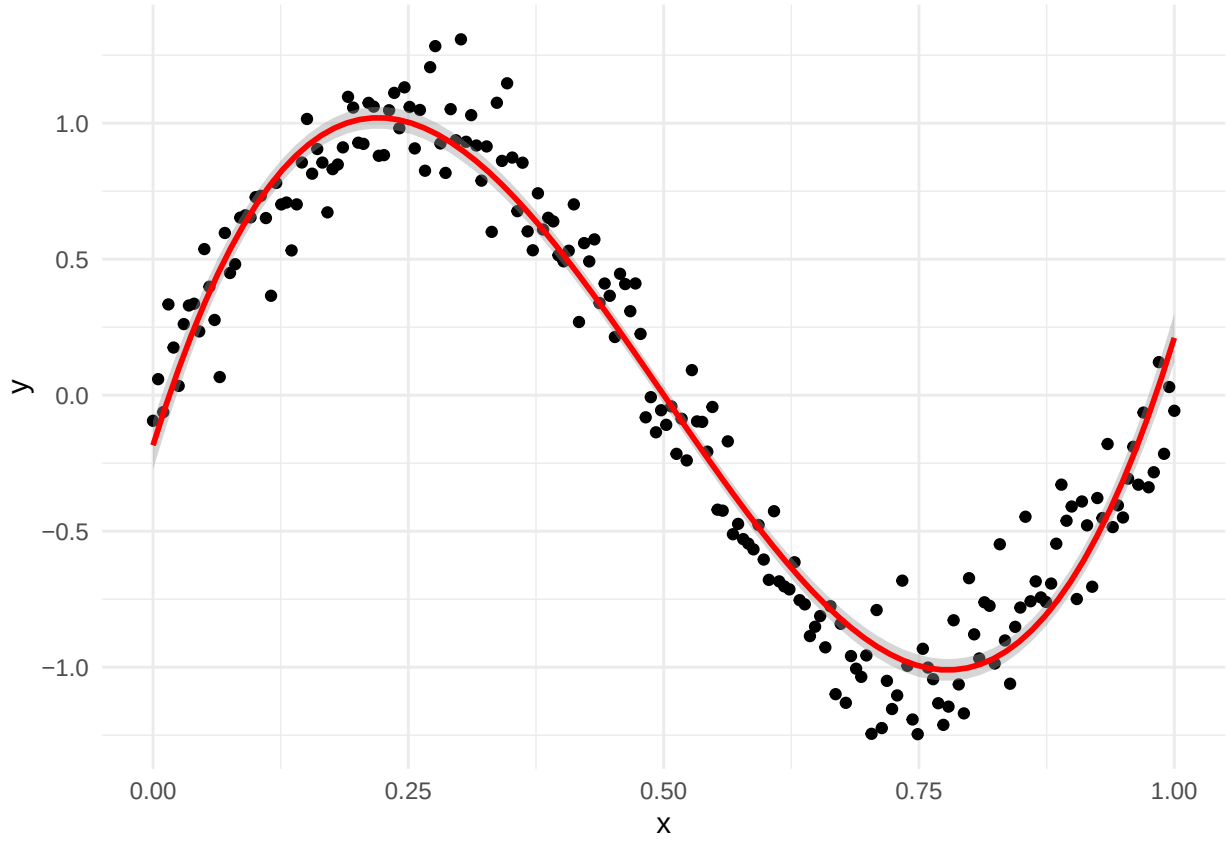
```
# dir.create("./data")
# dir.create("./R")
# dir.create("./plots")

df <- data.frame(x = x, y = y)
write.csv(df, "./data/data.csv", row.names = F)
```

(b)

```
read.csv("./data/data.csv")

library(ggplot2)
ggplot(df, aes(x = x, y = y)) +
  geom_point(color = "black") +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 3),
              color = "red") +
  theme_minimal()
```



(c)

```
p = ggplot(df, aes(x = x, y = y)) +
  geom_point(color = "black") +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 3),
              color = "red") +
  theme_minimal()
ggsave(filename = "./plots/plot.png", plot = p,
       width = 6, height = 4, units = "in",)
```

(d)

https://github.com/kimhyew1/Advanced_Statistical_Analysis

Problem 02

(a)

```
bubble_sort <- function(x, ascending = T) {
  n = length(x)

  if (ascending == F) {
    x = -x
  }

  for (i in 1:(n-1)) {
    flag = F

    for (j in 1:(n-i)) {
      if (x[j] > x[j+1]) {
        tmp = x[j]
        x[j] = x[j+1]
        x[j+1] = tmp
        flag = T
      }
    }
    if (flag == F) {
      break
    }
  }

  if (ascending == F) {
    x = -x
  }
  return (x)
}
```

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

bubble_sort(x, ascending = T)
```

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

```
bubble_sort(x, ascending = F)
```

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

(b)

```
quick_sort <- function(x, ascending = T) {
  n <- length(x)
  if (n <= 1) return(x)

  pivot <- x[1]
```

```
  i <- 2
  j <- n

  while (TRUE) {
    while (i <= n && x[i] <= pivot) {
      i <- i + 1
    }

    while (j >= 2 && x[j] > pivot) {
      j <- j - 1
    }

    if (i > j) break

    tmp <- x[i]
    x[i] <- x[j]
    x[j] <- tmp
  }

  tmp <- x[1]
  x[1] <- x[j]
  x[j] <- tmp

  small <- if (j > 1) quick_sort(x[1:(j-1)], ascending) else numeric(0)
  large <- if (j < n) quick_sort(x[(j+1):n], ascending) else numeric(0)
  # small = quick_sort(x[1:(j-1)])
  # large = quick_sort(x[(j+1):n])

  if (ascending == T) {
    result = c(small, x[j], large)
  } else {
    result = c(large, x[j], small)
  }

  return(result)
}
```

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

quick_sort(x, ascending = T)
```

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

```
quick_sort(x, ascending = F)
```

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

Problem 03

(a)

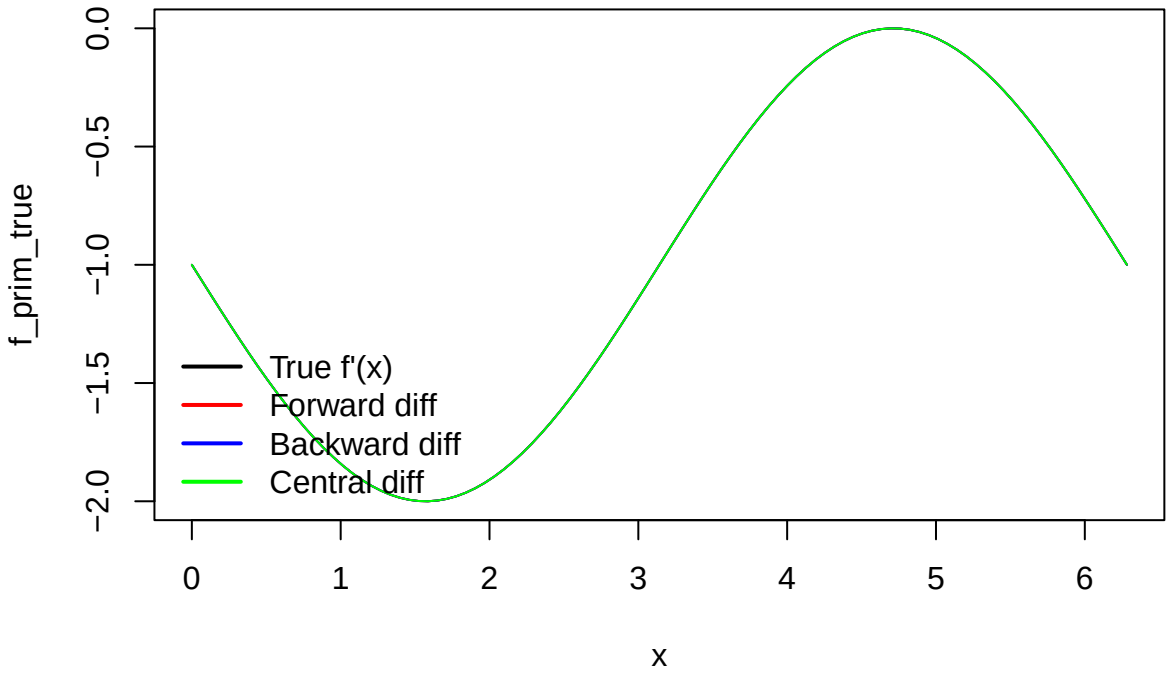
```
derivative_function = function(x, f, h = 1e-6, method = "forward") {
  if (method == "forward") {
    h = (f(x+h) - f(x)) / h
  } else if (method == "backward") {
    h = (f(x) - f(x-h)) / h
  } else if (method == "central") {
    h = (f(x+h) - f(x-h)) / (2*h)
  }
  return(h)
}
```

```
x = seq(0, 2*pi, length.out = 100)
f = function(x) {
  return(cos(x) - x)
}
f_prim_true = -sin(x) - 1

forw = derivative_function(x, f, method = "forward")
back = derivative_function(x, f, method = "backward")
cent = derivative_function(x, f, method = "central")

plot(x, f_prim_true, type = "l")
lines(x, forw, col = "red")
lines(x, back, col = "blue")
lines(x, cent, col = "green")

legend("bottomleft",
       legend = c("True f'(x)", "Forward diff", "Backward diff", "Central diff"),
       col = c("black", "red", "blue", "green"),
       lty = 1, lwd = 2, bty = "n")
```



(b)

```r
NewtonRapshon <- function(f, fprime = NULL, x0, maxiter = 100, h = 1e-6, epsilon = 1e-10) {
  x_old <- x0
  for (iter in 1:maxiter) {
    if (is.null(fprime)) {
      fprime_val <- derivative_function(x_old, f)
    } else {
      fprime_val <- fprime(x_old)
    }
    x_new = x_old - f(x_old) / fprime_val

    if (abs(x_new - x_old) < 1e-6) {
      break
    }
    x_old = x_new
  }
  return(x_new)
}
```

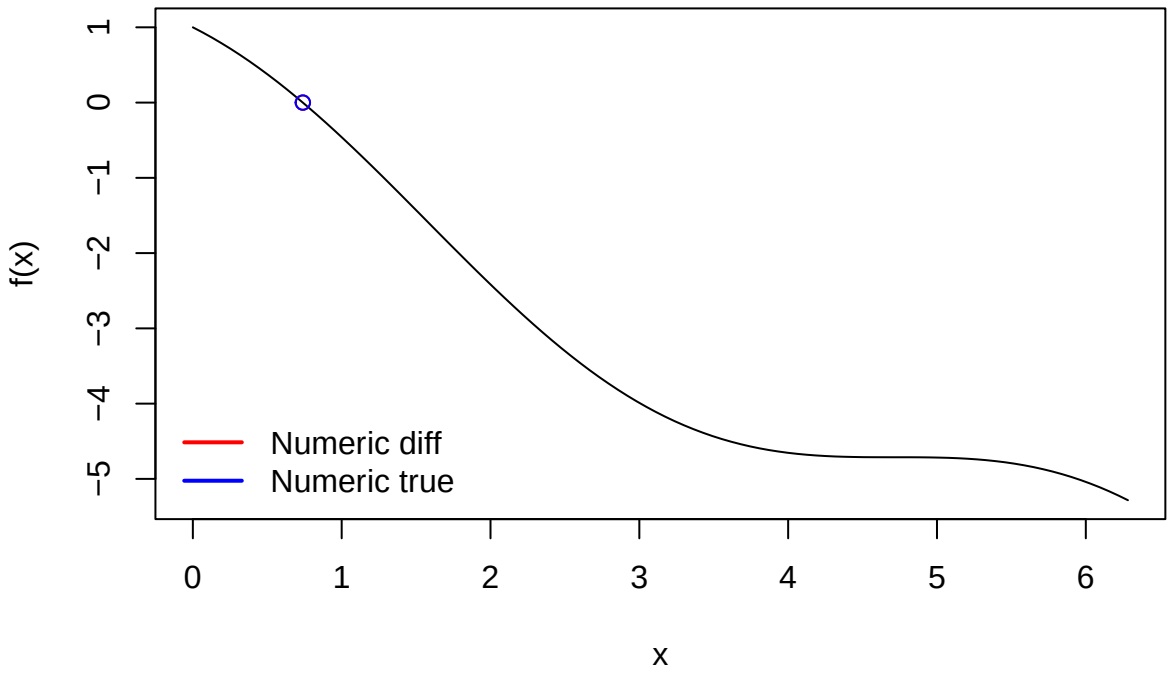
(c)

```
f = function(x) {
  return(cos(x) - x)
}
f_prim_true = function(x) {
  return(-sin(x) - 1)
}

num_diff = NewtonRapshon(f, NULL, 0.5)
num_true = NewtonRapshon(f, f_prim_true, 0.5)

plot(x, f(x), type = "l")
points(num_diff, f(num_diff), col = "red")
points(num_true, f(num_true), col = "blue")

legend("bottomleft",
       legend = c("Numeric diff", "Numeric true"),
       col = c("red", "blue"),
       lty = 1, lwd = 2, bty = "n")
```



Problem 04

(a)

```
LeftRectangle = function(f, a, b, n) {
  h = (b - a) / n
  answer = 0

  for (i in 0:(n-1)) {
    x0 = a + h * i
    answer = answer + f(x0)
  }
  return(h * answer)
}
```

(b)

```
Trapezoid = function(f, a, b, n) {
  h = (b - a) / n
  answer = 0

  for (i in 1:(n-1)) {
    x0 = a + h * i
    answer = answer + f(x0)
  }
  f0 = f(a)
  fn = f(a + h * n)
  answer = h/2 * (f0 + 2 * answer + fn)
  return(answer)
}
```

(c)

```
Simpson = function(f, a, b, n) {
  h = (b - a) / n
  odd_sum = 0; even_sum = 0

  for (i in 1:(n/2)) {
    odd_idx = a + h * 2*(i-1)
    odd_sum = odd_sum + f(odd_idx)
    even_idx = a + h * 2*i
    even_sum = even_sum + f(even_idx)
  }
  f0 = f(a)
  fn = f(a + n * h)
  answer = h/3 * (f0 + 4*odd_sum + 2*even_sum - fn)

  return(answer)
}
```

(d)

```
f = function(x) {
  return(sin(x))
}
a = 0; b = pi; n = 100

LeftRectangle_val = LeftRectangle(f, a, b, n)
Trapezoid_val = Trapezoid(f, a, b, n)
Simpson_val = Simpson(f, a, b, n)
```

Method	Estimated Value
Left Rectangle	1.99984
Trapezoid	1.99984
Simpson	1.99934

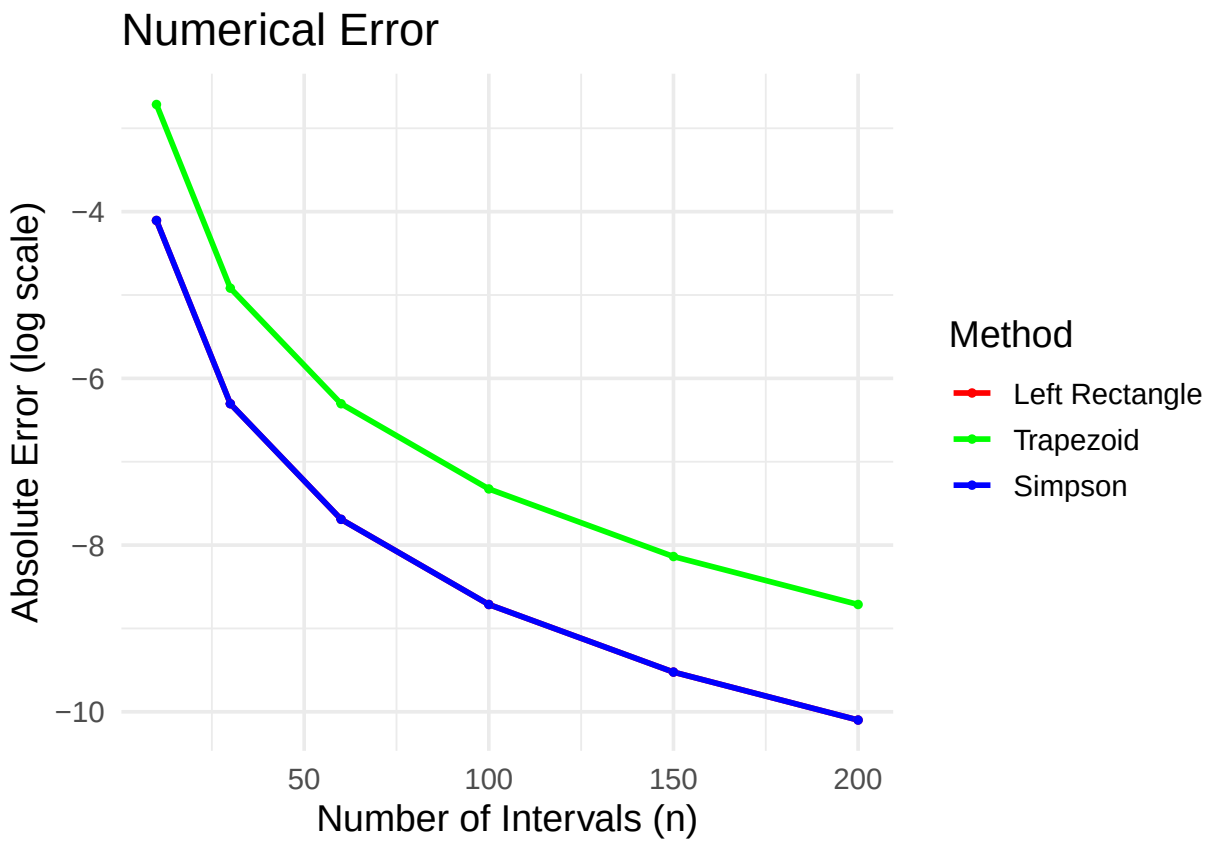
(e)

```
true_val = -cos(b) + cos(a)
n_list = c(10, 30, 60, 100, 150, 200)
errors = data.frame(
  n = n_list,
  left = sapply(n_list, function(n) abs(true_val - LeftRectangle(f, a, b, n))),
  trap = sapply(n_list, function(n) abs(true_val - Trapezoid(f, a, b, n))),
  simp = sapply(n_list, function(n) abs(true_val - Simpson(f, a, b, n)))
)
```

```
library(tidyr)

errors_long <- errors %>%
  pivot_longer(cols = c(left, trap, simp),
               names_to = "Method",
               values_to = "Error")

ggplot(errors_long, aes(x = n, y = log(Error), color = Method)) +
  geom_line(linewidth = 1) +
  geom_point(size = 1) +
  theme_minimal(base_size = 14) +
  labs(
    title = "Numerical Error",
    x = "Number of Intervals (n)",
    y = "Absolute Error (log scale)"
  ) +
  scale_color_manual(
    values = c("left" = "red", "trap" = "blue", "simp" = "green"),
    labels = c("Left Rectangle", "Trapezoid", "Simpson")
  )
```



Problem 05

(a)

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

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

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

(b)

```
forward_function = function(L, b) {
  n = nrow(L)
  z = c()
  for (i in 1:n) {
```

```
    z[i] = (b[i] - sum(L[i, 1:(i-1)] * z[1:(i-1)])) / L[i, i]
  }
  return(z)
}
```

(c)

```
backward_function = function(L, z) {
  n = nrow(L)
  x = c()
  x[n] = z[n] / L[n, n]
  for (i in (n-1):1) {
    x[i] = (z[i] - sum(L[(i+1):n, i] * x[(i+1):n])) / L[i, i]
  }
  return(x)
}
```

(d)

```
b = c(1, -2, 3)
z = forward_function(L, b)
x = backward_function(L, z)

all.equal(x, solve(A, b))
```

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

Problem 06

(a)

```
GaussianKernal = function(xi, xj, rho = 1) {
  answer = exp(-rho * abs(xi - xj)**2)
  return(answer)
}
```

##(b)

```
KernalRidgeRegression = function(X, y, lambda = 0.0001) {
  n = nrow(X)
  krr_model = matrix(NA, nrow = n, ncol = n)
  for (i in 1:n) {
    for (j in 1:n) {
      krr_model[i, j] = GaussianKernal(X[i], X[j])
    }
```

```
  }

  positive_matrix = krr_model + diag(rep(lambda, n))
  L = t(chol(positive_matrix))

  alpha = backward_function(L, z = forward_function(L, y))

  class(krr_model) = "krr"
  attr(krr_model, "alpha") = alpha
  attr(krr_model, "X") = X
  attr(krr_model, "y") = y

  return(krr_model)
}
```

(c)

```
predict.krr = function(model, x) {
  X_train = attr(model, "X")
  alpha = attr(model, "alpha")

  k_val = sapply(1:nrow(X_train), function(i) GaussianKernal(x, X_train[i]))
  y_pred = k_val %*% alpha
  return(y_pred)
}
```

(d)

```
plot.krr = function(model, ...) {
  x_grid = seq(-1, 1, length.out = 300)
  y_pred = predict(model, x_grid)

  x = attr(model, "X")
  y = attr(model, "y")

  plot(x, y, col = "black")
  lines(x_grid, y_pred, col = "red")
}
```

(e)

```
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)
```

```
kernal_model = KernalRidgeRegression(X, y)
plot(kernal_model)
```