고급통계분석 - Assignment01

20220866 문가영

2025-10-16

### Problem 1.

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

#### (a).

df = data.frame(x, y)  
write.csv(df, file = 'data/problem1.csv', row.names = FALSE)

#### (d).

**GitHub 저장소 링크:** [20220866 문가영 GitHub 링크](https://github.com/Ga-young-Moon/Advanced-Statistical-Analysis)

### Problem 2.

#### (a).

버블 정렬 함수 구현:

# 버블 정렬  
bubble\_sort <- function(vec, decreasing = FALSE){  
 n = length(vec) # 길이가 n인 입력 벡터  
   
 # 오름차순 옵션  
 if (decreasing == FALSE){  
 for (i in 1:(n-1)){  
 for (j in 1:(n-i)){  
 if (vec[j] > vec[j+1]){  
 temp = vec[j]  
 vec[j] = vec[j+1]  
 vec[j+1] = temp  
 }  
 }  
 }  
 }  
   
 # 내림차순 옵션  
 else{  
 for (i in 1:(n-1)){  
 for (j in 1:(n-i)){  
 if (vec[j] < vec[j+1]){  
 temp = vec[j]  
 vec[j] = vec[j+1]  
 vec[j+1] = temp  
 }  
 }  
 }  
 }  
 return(vec)  
}

구현한 버블 정렬 알고리즘 실행 결과 확인:

set.seed(1)  
x = runif(10)  
  
print('\*\*\*원본 벡터 (x)\*\*\*')

## [1] "\*\*\*원본 벡터 (x)\*\*\*"

print(x)

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

# 1. 오름차순 정렬 (decreasing = FALSE)  
x\_asc = bubble\_sort(x, decreasing = FALSE)  
print('\*\*\*1. 오름차순 결과\*\*\*')

## [1] "\*\*\*1. 오름차순 결과\*\*\*"

print(x\_asc)

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

# 2. 내림차순 정렬(decreasing = TRUE)  
x\_desc = bubble\_sort(x, decreasing = TRUE)  
print('\*\*\*2. 내림차순 결과\*\*\*')

## [1] "\*\*\*2. 내림차순 결과\*\*\*"

print(x\_desc)

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

#### (b).

퀵 정렬 함수 구현:

# 퀵 정렬  
quick\_sort <- function(vec, decreasing = FALSE){  
 n = length(vec) # 길이가 n인 입력 벡터  
   
 # 재귀 종료 조건  
 if (n <= 1){  
 return(vec)  
 }  
   
 # Pivot 분리  
 pivot <- vec[1]  
 rest <- vec[-1]  
   
 # 분할  
 # 오름차순 옵션  
 if (decreasing == FALSE){  
 Left <- rest[rest <= pivot]  
 Right <- rest[rest > pivot]  
 }  
   
 # 내림차순 옵션  
 else{  
 Left <- rest[rest >= pivot]  
 Right <- rest[rest < pivot]  
 }  
   
 # 재귀 호출  
 Left\_sort <- quick\_sort(Left, decreasing)  
 Right\_sort <- quick\_sort(Right, decreasing)  
   
 # 결합  
 return(c(Left\_sort, pivot, Right\_sort))  
}

구현한 퀵 정렬 알고리즘 실행 결과 확인:

set.seed(1)  
x = runif(10)  
  
print('\*\*\*원본 벡터 (x)\*\*\*')

## [1] "\*\*\*원본 벡터 (x)\*\*\*"

print(x)

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

# 1. 오름차순 정렬 (decreasing = FALSE)  
x\_asc = quick\_sort(x, decreasing = FALSE)  
print('\*\*\*1. 오름차순 결과\*\*\*')

## [1] "\*\*\*1. 오름차순 결과\*\*\*"

print(x\_asc)

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

# 2. 내림차순 정렬(decreasing = TRUE)  
x\_desc = quick\_sort(x, decreasing = TRUE)  
print('\*\*\*2. 내림차순 결과\*\*\*')

## [1] "\*\*\*2. 내림차순 결과\*\*\*"

print(x\_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.

#### (a).

수치 미분 함수 구현:

# 수치 미분  
num\_diff <- function(f, x, h= 1e-6, method){  
 if (method == 'forward'){  
 result <- (f(x+h) - f(x)) / h  
 return(result)  
 }  
 else if (method == 'backward'){  
 result <- (f(x) - f(x-h)) / h  
 return(result)  
 }  
 else if (method == 'central'){  
 result <- (f(x+h) - f(x-h)) / (2\*h)  
 return(result)  
 }  
 else{  
 stop('Choose the method you want to use.')  
 }  
}

구현한 수치 미분 알고리즘 실행 결과 확인:

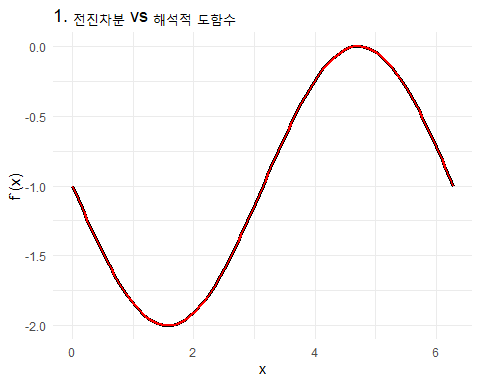
n <- 100  
x <- seq(0, 2\*pi, length.out = n)  
  
# f 함수와 실제 도함수(f`) 정의  
f <- function(x){  
 return(cos(x) - x)  
}  
  
f\_prime <- function(x){  
 return(-sin(x) - 1)  
}  
  
# 구현한 함수 적용  
# 1. 전진차분  
y\_for <- num\_diff(f, x, method= 'forward')  
  
# 2. 후진차분  
y\_back <- num\_diff(f, x, method= 'backward')  
  
# 3. 중심차분  
y\_cen <- num\_diff(f, x, method= 'central')

시각화:

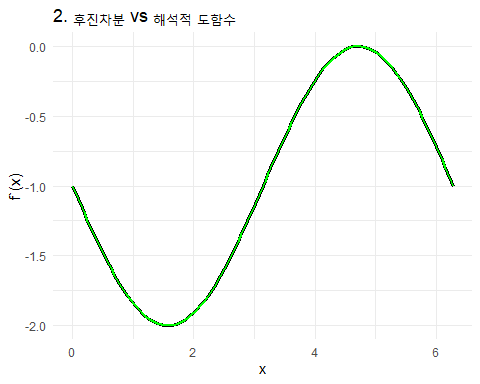
library(ggplot2)  
  
# 해석적 도함수 정의  
y\_prime <- f\_prime(x)  
  
# 시각화  
# 1. 전진차분  
df\_for <- data.frame(x = x, analytic = y\_prime, forward = y\_for)  
  
p1 <- ggplot(df\_for, aes(x = x)) +  
 geom\_line(aes(y = analytic), color= 'black', size= 1.15) +  
 geom\_line(aes(y = forward), color= 'red', size = 0.9) +  
 labs(title= '1. 전진차분 vs 해석적 도함수',  
 x= 'x',  
 y= 'f`(x)') +  
 theme\_minimal()

## 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.

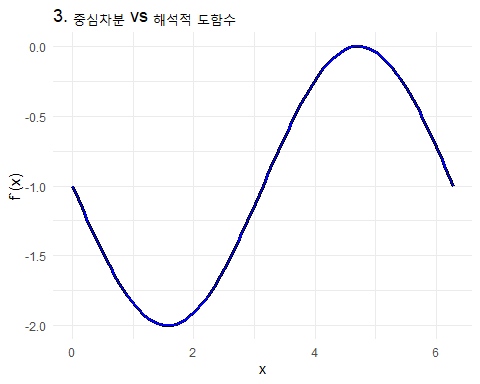
# 2. 후진차분  
df\_back <- data.frame(x= x, analytic = y\_prime, backward = y\_back)  
  
p2 <- ggplot(df\_back, aes(x = x)) +  
 geom\_line(aes(y = analytic), color= 'black', size= 1.15) +  
 geom\_line(aes(y = backward), color= 'green', size = 0.9) +  
 labs(title= '2. 후진차분 vs 해석적 도함수',  
 x= 'x',  
 y= 'f`(x)') +  
 theme\_minimal()  
  
# 3. 중심차분  
df\_cen <- data.frame(x= x, analytic = y\_prime, central = y\_cen)  
  
p3 <- ggplot(df\_cen, aes(x = x)) +  
 geom\_line(aes(y = analytic), color= 'black', size= 1.15) +  
 geom\_line(aes(y = central), color= 'blue', size = 0.9) +  
 labs(title= '3. 중심차분 vs 해석적 도함수',  
 x= 'x',  
 y= 'f`(x)') +  
 theme\_minimal()  
  
  
# plot 출력  
print(p1)



print(p2)



print(p3)



#### (b).

Newton-Rapshon 알고리즘 구현:

# Newton-Rapshon 방법  
newton\_rapshon <- function(f, fprime= NULL, x0, maxiter = 100, h = 1e-6, epsilon = 1e-10){  
   
 x\_current <- x0  
 iter <- 0  
   
 # fprime 값이 입력되지 않은 경우  
 if (is.null(fprime)){  
 f\_prime <- function(x\_value){  
 return((f(x\_value+ h) - f(x\_value - h)) / (2\*h))  
 }  
 }  
   
 # fprime 값이 입력된 경우  
 else{  
 f\_prime <- fprime  
 }  
   
 # while문 사용해 반복 진행  
 while(iter < maxiter){  
 x\_prev <- x\_current  
   
 # x\_t 계산  
 x\_current = x\_prev - f(x\_prev) / f\_prime(x\_prev)  
 iter <- iter + 1  
   
 if (abs(x\_current - x\_prev) < epsilon){  
 break  
 }  
 }  
 return(x\_current)  
}

#### (c).

f(x) = cos(x) - x = 0을 만족하는 해 찾기

* 수치미분 버전:

n <- 100  
x <- seq(0, 2\*pi, length.out = n)  
  
# f 함수와 실제 도함수(f`) 정의  
f <- function(x){  
 return(cos(x) - x)  
}  
  
newton\_rapshon(f, NULL, 0.5)

## [1] 0.7390851

* 도함수 제공 버전:

# 도함수 제공  
f\_prime <- function(x){  
 return(-sin(x) - 1)  
}  
  
newton\_rapshon(f, f\_prime, 0.5)

## [1] 0.7390851

### Problem 4.

#### (a).

Left Rectangle 방식 구현:

left\_rec <- function(f, a, b, n){  
 # 등간격 h 계산  
 h <- (b - a) / n  
   
 integral\_sum <- 0  
   
 for (i in 0:(n-1)){  
 # 현재 좌측 끝점 위치 계산  
 x\_i = a + (i \* h)  
   
 integral\_sum = integral\_sum + f(x\_i)  
 }  
 return(integral\_sum \* h)  
}

#### (b).

Trapezoid 방식 구현:

trapezoid <- function(f, a, b, n){  
 # 등간격 h 계산  
 h <- (b - a) / n  
   
 integral\_sum <- f(a) + f(b)  
   
 for (i in 1:(n-1)){  
 # 현재 좌측 끝점 위치 계산  
 x\_i = a + (i \* h)  
   
 integral\_sum = integral\_sum + 2\*f(x\_i)  
 }  
 return((h / 2) \* integral\_sum)  
}

#### (c).

Simpson 방식 구현:

simpson <- function(f, a, b, n){  
   
 # n = 짝수  
 if (n%%2 != 0){  
 stop('n must be even.')  
 }  
   
 # 등간격 h 계산  
 h <- (b- a) / n  
 integral\_sum <- f(a) + f(b)  
   
 for(i in 1:(n-1)){  
 # 현재 좌측 끝점 위치 계산  
 x\_i = a + (i \* h)  
   
 # i = odd인 경우  
 if (i%%2 != 0){  
 int\_odd <- 4 \* f(x\_i)  
 integral\_sum <- integral\_sum + int\_odd  
 }  
   
 # i = even인 경우  
 else{  
 int\_even <- 2 \* f(x\_i)  
 integral\_sum <- integral\_sum + int\_even  
 }  
 }  
 return((h / 3) \* integral\_sum)  
}

#### (d).

f(x) = sin(x)인 경우의 수치적분 결과:

library(flextable)  
  
f <- function(x){  
 return(sin(x))  
}  
n <- 100  
  
integral <- data.frame(  
 '방법' = c('Left Rectangle', 'Trapezoid', 'Simpson'),  
 '계산 결과' = c(left\_rec(f, 0, pi, n), trapezoid(f, 0, pi, n), simpson(f, 0, pi, n))  
)  
flextable(integral)

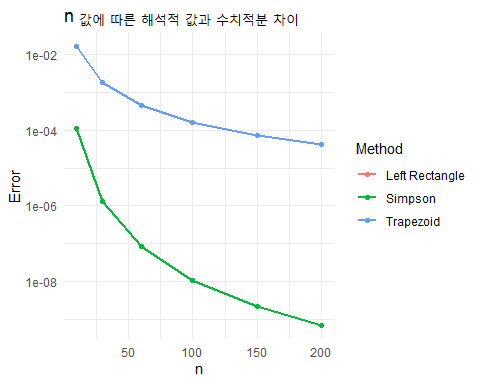
| 방법 | 계산.결과 |
| --- | --- |
| Left Rectangle | 1.999836 |
| Trapezoid | 1.999836 |
| Simpson | 2.000000 |

#### (e).

해석적 값:

시각화:

# 해석적 적분 값  
f\_true <- 2  
  
# n 값 벡터  
n\_values <- c(10, 30, 60, 100, 150, 200)  
  
f <- function(x){  
 return(sin(x))  
}  
  
a<- 0; b<- pi  
  
# 오차 계산  
results <- data.frame(n = integer(), Method = character(), Error = numeric())  
  
for (n in n\_values){  
 rec\_val <- left\_rec(f, a, b, n)  
 trap\_val <- trapezoid(f, a, b, n)  
 simps\_val <- simpson(f, a, b, n)  
   
 rec\_err <- abs(f\_true - rec\_val)  
 trap\_err <- abs(f\_true - trap\_val)  
 simps\_err <- abs(f\_true - simps\_val)  
   
 results <- rbind(results,  
 data.frame(n = n, Method = 'Left Rectangle', Error = rec\_err),  
 data.frame(n = n, Method = 'Trapezoid', Error = trap\_err),  
 data.frame(n = n, Method = 'Simpson', Error = simps\_err))  
}  
  
# 시각화  
library(scales)  
p <- ggplot(results, aes(x = n)) +  
 geom\_line(aes(y = Error, color= Method), size = 1) +  
 geom\_point(aes(y = Error, color= Method), size = 1.5) +  
 scale\_y\_log10(labels = scales::label\_scientific()) +  
 labs(title = 'n 값에 따른 해석적 값과 수치적분 차이', x= 'n', y= 'Error') +  
 theme\_minimal()  
print(p)



### Problem 5.

#### (a).

A <- matrix(c(4, 2, 2, 2, 5, 1, 2, 1, 3), 3)  
U <- chol(A)  
  
# L = U^T  
L <- t(U)  
  
print(L%\*%t(L))

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

print(A)

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

#### (b).

forward <- function(L, y){  
 # 행렬의 크기(n) 구하기  
 n <- nrow(L)  
   
 # 결과 벡터 z 초기화  
 z <- numeric(n)  
   
 for (i in 1:n){  
   
 }  
}

#### (c).

#### (d).

### Problem 6.

#### (a).

#### (b).

#### (c).

#### (d).

#### (e).