FINAL EXAM

2020321163_엄상준

1.

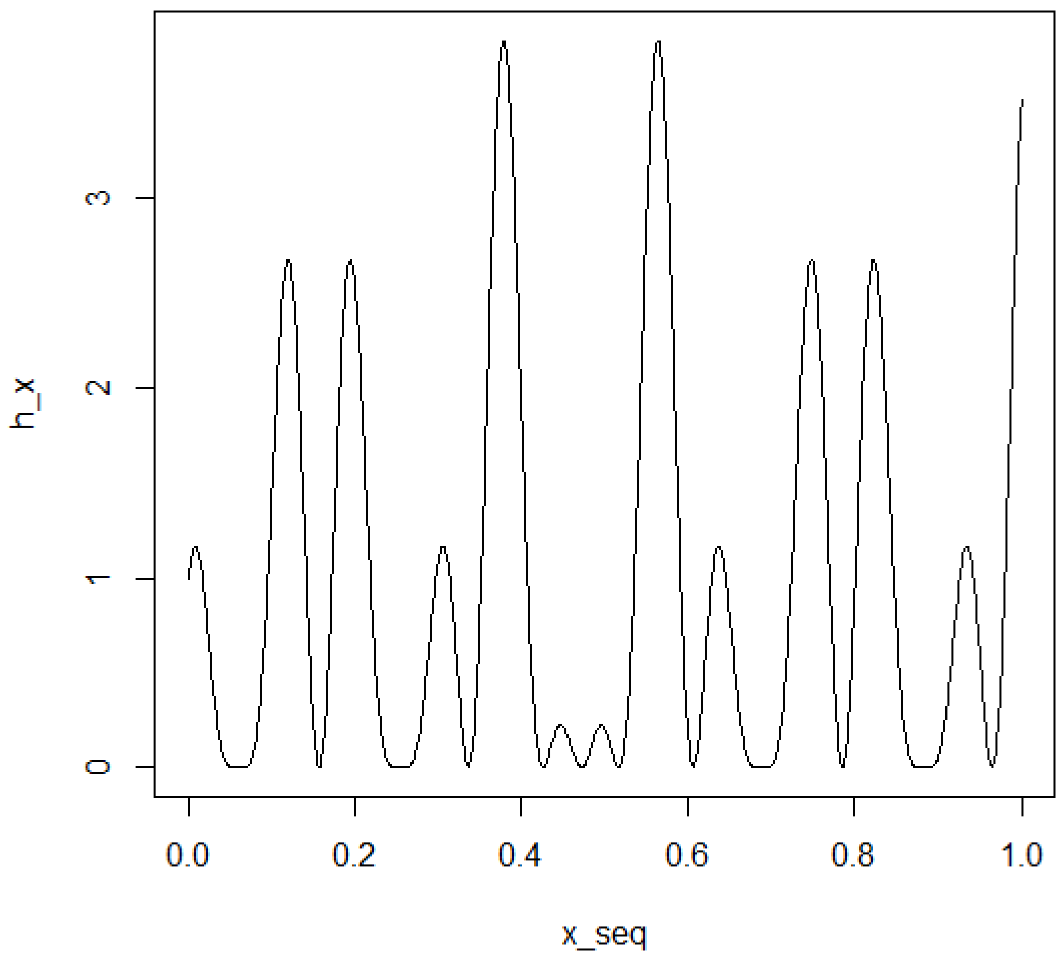
(a)

우선 h(x) function을 r 코드로 구현하면 다음과 같다.

```
h_function <- function(x){
  res <- (cos(50*x) + sin(20*x))^2
  return(res)
}
```

그리고 이를 그래프로 그려보면(x의 범위는 0과 1사이로 둔다.)

```
x_seq <- seq(0, 1, length.out = 1000)
h_x <- h_function(x_seq)
plot(x_seq, h_x, type='l')
```



→ 봉우리가 굉장히 많은 형태를 가지고 있다는 것을 알 수 있다.

다음으로는 Newton-Raphson 방법을 위해 h prime과 h two prime 함수를 작성하였다.

```
h_prime <- function(x){
  res <- 2*(cos(50*x)+sin(20*x))*(20*cos(20*x)-50*sin(50*x))
  return(res)
}

h_2prime <- function(x){
  res <- 2*(20*cos(20*x)-50*sin(50*x))^2 +
    2*(-2500*cos(50*x)-400*sin(20*x))*(cos(50*x)+sin(20*x))
  return(res)
}
```

그리고 Newton-Raphson 코드를 작성했다.

```
Newton <- function(max_iteration, epsilon, x){
  diff = 1
  i = 0
  while(abs(diff) > epsilon){
    diff = -h_prime(x)/h_2prime(x)
    x = x + diff
    i = i + 1
    if(i == max_iteration){
      print("Iteration is over. Need More Max_iter")
      break
    }
    if(x <0 | x>1){
      print('Out of Range. Please Check the Starting value')
      break
    }
  }
  return(x)
}
```

세 구간을 설정한 뒤 각 구간 별을 10개로 나누어 여러 starting point를 설정하고 그에 따른 최적화값이 어떻게 나오는지를 살펴보았다.

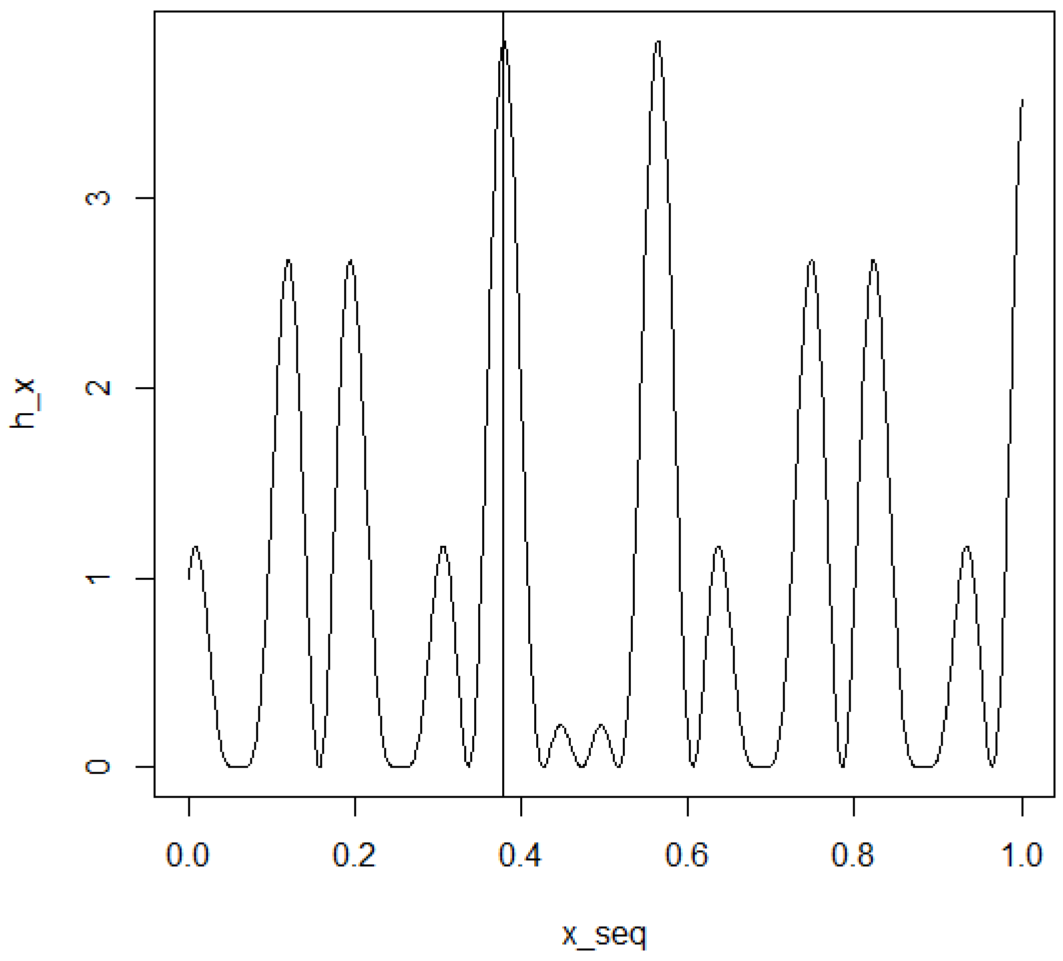
1. 0.36 ~ 0.38

```
Newton(10000, 0.001, seq(0.36, 0.38, length.out = 10))
```

결과: 0.6364301 0.4263579 0.3771837 0.3791384 0.3791384 0.3791384 0.3791384 0.3791384 0.3791384 0.3791384

0.3791384 값으로 대부분 수렴하였다. 0.3791384 값을 살펴보면,

```
abline(v=0.3791384)
```



local maximum 값으로 수렴했다는 것을 알 수 있다.

그리고 그에 따른 h(x)값은

```
h_function(0.3791384)
```

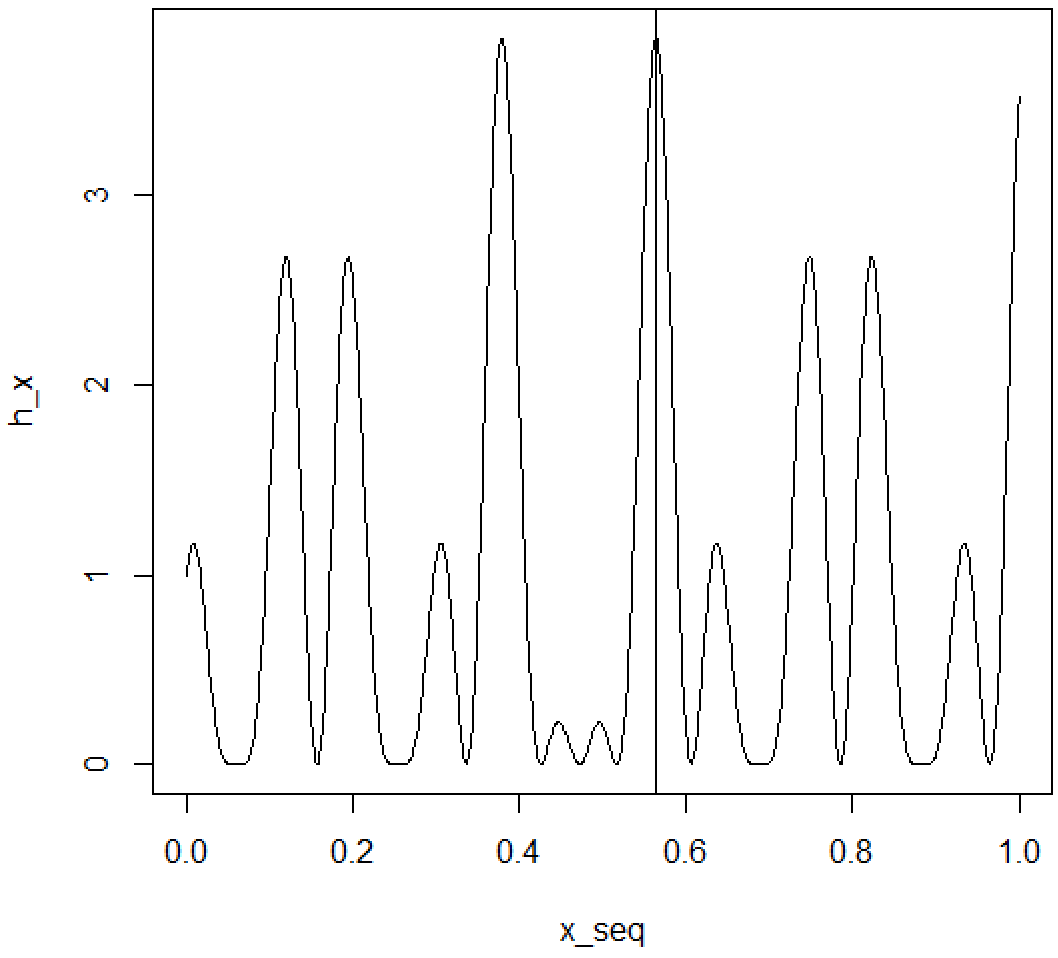
3.832544 이다.

2. 0.55 ~ 0.58

같은 방식으로,

```
Newton(10000, 0.001, seq(0.55, 0.58, length.out = 10))
abline(v=0.5633394)
h_function(0.5633394)
```

- 0.5633394 0.5633394 0.5633394 0.5633394 0.5633394 0.5633394 0.5633394 0.5633394 0.5633394 0.5161188



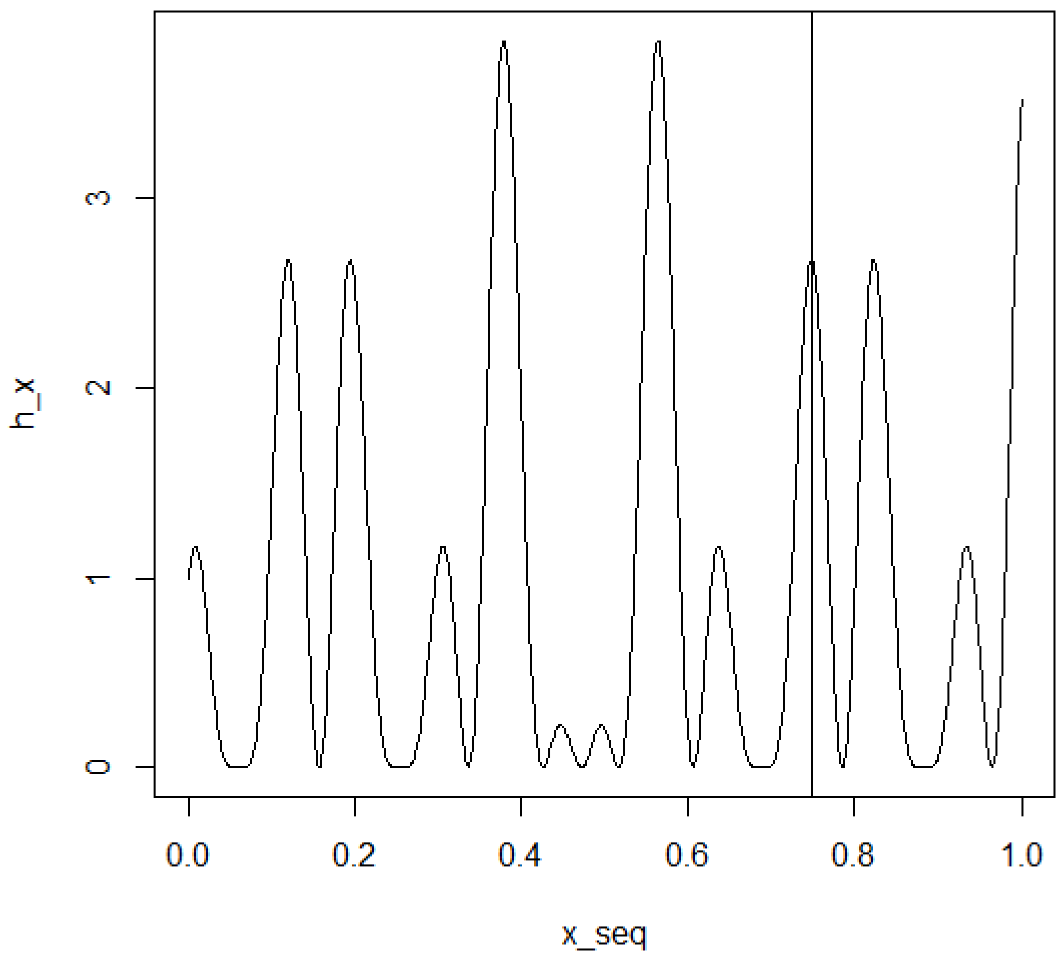
- 3.832544

3. 0.72 ~0.75

역시 같은 방법으로,

```
Newton(10000, 0.001, seq(0.72, 0.75, length.out = 10))
abline(v=0.7480271)
h_function(0.7480271)
```

- 0.6956686 0.6959677 0.6805383 1.0074570 0.7867541 0.7480271 0.7480271 0.7480271 0.7480271 0.7480271



- 2.675667

이제 optimize function을 이용해서 최적값을 찾아보면,

```
optimize(h_function, c(0,1), tol=0.0001, maximum = TRUE)
```

> optimize(h_function, c(0,1), tol=0.0001, maximum = TRUE)
> $maximum
> [1] 0.3791249
>
> $objective
> [1] 3.832543

맨 처음 구간은 global maximum 값을 잡아낸 반면, 나머지 두 구간은 local maximum 값을 잡아냈다는 것을 알 수 있다. (단, 마지막 구간은 local maximum이지만 global maximum과 거의 동일.)

또한 맨 처음 구간의 경우도 local maximum이 global maximum과 우연히 맞은 경우라고 볼 수 있다.

이는 봉우리가 많은 multimodal인 경우, newton-raphson 방식은 starting point에 따라 local-maximum에 빠질 수도 있다는 것을 보여준다.

(b)

iteration은 2500, r=0.5로 하는, annealing function을 다음과 같이 설정하였다.

```
N <- 2500
u <- runif(N)
xval1 <- rep(0, 2500)
r <- 0.5

annealing <- function(start_value, r,iteration){
  run.current <- start_value
  run.best <- run.current
  best <- h_function(start_value)
  runs <- c()
  for(i in 1:iteration){
    run_h <- h_function(run.current)
    u <- runif(1, min = max((run.current-r),0), max=min((run.current+r),1))
    stat <- exp(log(i) * (h_function(u) - h_function(run.current)))
    p.current <- min(stat, 1)
    if(runif(1)<p.current){
      run.current <- u
      run_h <- h_function(u)
    }
    if(run_h > best){
      run.best <- run.current
      best <- run_h
    }
    runs[i] <- run.current
  }
  res <- list(x= runs, max = c(run.best, best))
  return(res)
}
```

- 이제 starting point를 0.2, 0.4, 0.6, 0.8로 하였을 때 어떤 값으로 수렴하는 지를 확인하자.

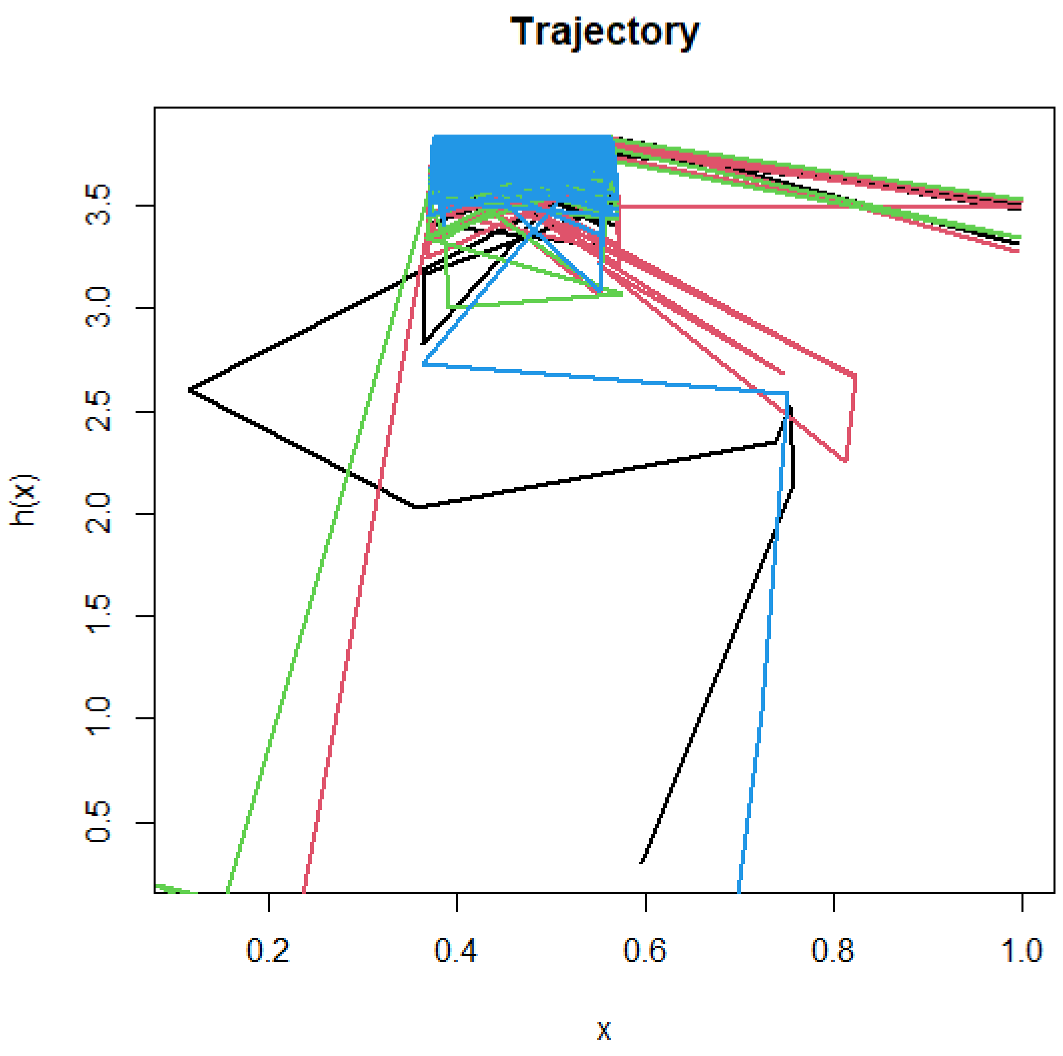
```
anneal1 <- annealing(0.2, 0.5, 10000)
anneal2 <- annealing(0.4, 0.5, 10000)
anneal3 <- annealing(0.6, 0.5, 10000)
anneal4 <- annealing(0.8, 0.5, 10000)

anneal1$max
anneal2$max
anneal3$max
anneal4$max
```

> anneal1$max
> [1] 0.5633207 3.8325422
> anneal2$max
> [1] 0.5633067 3.8325382
> anneal3$max
> [1] 0.3790788 3.8325242
> anneal4$max
> [1] 0.5633398 3.8325442

starting point에 상관없이 global maximum값에 수렴하는 것을 알 수 있다.

- Trajectory를 그려보면,

```
plot(anneal1$x, h_function(anneal1$x), type='l', lwd=2, xlab='x', ylab='h(x)',
main='Trajectory')
lines(anneal2$x, h_function(anneal2$x), col='2', lwd=2)
lines(anneal3$x, h_function(anneal3$x), col='3', lwd=2)
lines(anneal4$x, h_function(anneal4$x), col='4', lwd=2)
```



모두 maximum값으로 잘 수렴하는 형태를 보인다.

2

(a)

2.

Model을 다음과 같이 정의하자.

$y_i = X_i\beta + A_i\eta_i + e_i$ for $i=1, \cdots, N$

where y_i : i번째 individual의 $n_i \times 1$ observation vector.

X_i : $n_i \times d$ design matrix

β : $d \times 1$ fixed effects vector

A_i : $n_i \times p$ design matrix

η_i : $p \times 1$ random effects vector , $\eta_i \overset{iid}{\sim} N(0_p, G)$

e_i : $n_i \times 1$ residual errors vector. $e_i \overset{iid}{\sim} N(0_{n_i}, \sigma^2 I_{n_i})$

$\Rightarrow$ $y_i \sim N(X_i\beta, A_i G A_i^T + \sigma^2 I_{n_i})$

다음과 같이 쓸 수도 있다.

$y = X\beta + A\eta + e$

where $y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$, $X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{pmatrix}$ $A = \begin{pmatrix} A_1 & & & O \\ & A_2 & & \\ & & \ddots & \\ O & & & A_N \end{pmatrix}$ $\eta = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_N \end{pmatrix}$, $e = \begin{pmatrix} e_1 \\ \vdots \\ \vdots \\ e_N \end{pmatrix}$

Let $\theta = (\beta, G, \sigma^2)$ be the set of model parameters.

만약 η를 안다고 했을 때, ML Estimator of θ는 다음의 Complete log-likelihood를 최대화한다.

$\ell(\theta) = \log(p(y, \eta; \theta))$

$= \log(p(y|\eta; \theta)) + \log(p(\eta; \theta))$

$= \log(p(y|\eta; \beta, \sigma^2)) + \log(p(\eta; G))$

그럼 이 때 estimator $\hat{\beta}$와 $\hat{\sigma}^2$은 다음의 식을 최소화 하는 값.

$$-2\log(p(y|\eta;\beta,\sigma^2)) = n\log(2\pi\sigma^2) + \frac{\|y-X\beta-A\eta\|^2}{\sigma^2}$$

Where $n = \sum_{i=1}^{N} n_i$

그리고 $\hat{G}$는 다음을 최소화 한다.

$$-2\log(p(\eta;G)) = N\log(2\pi) + N\log(|G|) + \sum_{i=1}^{N} \eta_i^T G^{-1}\eta_i$$

$\therefore \quad \hat{\beta} = (X^TX)X^T(y-A\eta)$

$$\hat{G} = \frac{1}{N}\sum_{i=1}^{N}\eta_i\eta_i^T$$

$$\hat{\sigma}^2 = \frac{1}{n}\|y-X\hat{\beta}-A\eta\|^2 = \frac{1}{n}\left(\|y-X\hat{\beta}\|^2 + \|A\eta\|^2 - 2\langle y-X\hat{\beta}, A\eta\rangle\right)$$

$$= \frac{1}{n}\left(\|y-X\hat{\beta}\|^2 + \sum_{i=1}^{N} tr(A_i^TA_i\eta_i\eta_i^T) - 2\langle y-X\hat{\beta}, A\eta\rangle\right)$$

Let $S(y,\eta) = (\eta_1, \cdots, \eta_N, \eta_1\eta_1^T, \cdots, \eta_N\eta_N^T)$

즉, $\hat{\theta}$는 $S(y,\eta)$에 관한 어떤 함수 값.

$l(\theta)$를 다시 써보면

$$l(\theta) = \log(p(y_1, \cdots, y_N;\theta))$$

$$= \sum_{i=1}^{N}\log(p(y_i;\theta))$$

$$= \sum_{i=1}^{N}\left\{-\frac{n_i}{2}\log(2\pi) - \frac{1}{2}\log(|A_iGA_i^T+\sigma^2I_{n_i}|) - \frac{1}{2}(y_i-X_i\beta)^T(A_iGA_i^T+\sigma^2I_{n_i})^{-1}(y_i-X_i\beta)\right\}$$

$\Rightarrow$ 이 때 η이 unknown인 경우 $S(y,\eta)$는 계산할 수 없다. $\Rightarrow$ 대신 $E(S(y,\eta)|y;\theta)$를 구하자.

(1) E-Step.

By Bayes theorem,

$$p(\eta_i|y_i) = \frac{p(y_i|\eta_i)p(\eta_i)}{p(y_i)}$$

$$= C_1 \times \exp\{-\frac{1}{2\sigma^2}\|y_i - x_i\beta - A_i\eta_i\|^2 - \frac{1}{2}\eta_i^T G^{-1}\eta_i\}$$

$$= C_2 \times \exp\{-\frac{1}{2}(\eta_i - \mu_i)^T T_i^{-1}(\eta_i - \mu_i)\}$$

where $T_i = \left(\frac{A_i^T A_i}{\sigma^2} + G^{-1}\right)^{-1}$

$$\mu_i = \frac{T_i A_i^T (y_i - x_i\beta)}{\sigma^2}$$

$\therefore$ $E[\eta_i^{(t)}|y_i] = \mu_i^{(t)} = \frac{T_i^{(t)} A_i^T (y_i - x_i\hat{\beta}^{(t)})}{\hat{\sigma}^{2(t)}}$ where $T_i^{(t)} = \left(\frac{A_i^T A_i}{\hat{\sigma}^{2(t)}} + G^{(t)-1}\right)^{-1}$

$$E[\eta_i^{(t)}\eta_i^{(t)T}|y_i] = Var(\eta_i^{(t)}|y_i) + E[\eta_i^{(t)}|y_i]E[\eta_i|y_i]^T$$

$$= T_i^{(t)} + \mu_i^{(t)}\mu_i^{(t)T}$$

(2) M-Step.

$$\hat{\beta}^{(t+1)} = (X^T X)^{-1} X^T (y - A E[\eta_i^{(t)}|y_i])$$

$$\hat{G}^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N} E[\eta_i^{(t)}\eta_i^{(t)T}|y]$$

$$\hat{\sigma}^{(t+1)2} = \frac{1}{n}\left(\|y - X\hat{\beta}^{(t+1)}\|^2 + \sum_{i=1}^{N} tr(A_i^T A_i \underline{E[\eta_i^{(t)}\eta_i^{(t)T}|y]}) - 2\sum_{i=1}^{N}(y_i - x_i\hat{\beta}^{(t+1)})^T A_i E[\eta_i^{(t)}|y_i]\right)$$

(b)

```r
#setting
rikz <- read.table('rikz.txt', header=TRUE)
x <- rikz$NAP
y <- rikz$Richness
z <- rikz$Beach
uid <- c(1:45)

#EM Algorithm
EM <- function(y, x, z, uid, iteration){
  y <- as.matrix(y)
  x <- as.matrix(x)
  z <- as.matrix(z)
  N <- length(uid)
  n <- length(y)

  ##Initial value
  beta <- as.vector(solve(t(x)%*%x)%*%t(x)%*%y)
  g <- diag(rep(1, ncol(z)))
  sig2 <- 1
  residu <- as.vector(y - x%*%beta)

  ##Iteration
  for(j in 1:iteration){
    ##E-step
    P <- 0
    R <- 0
    C <- 0
    mu <- NULL
    u <- NULL
    for(i in uid){
      ##E-step
      xi <- x[i,]
      zi <- z[i,]
      residui <- residu[i]
      gammai <- solve(t(zi)%*%zi/sig2 + solve(g))
      mui <- (gammai%*%t(zi)%*%residui)/sig2
      mu <- c(mu, mui)
      u <- c(u, zi%*%mui)
      si <- gammai + mui %*% t(mui)
      R <- R + si
      P <- P + sum(diag(si%*%t(zi)%*%zi))
      C <- C + t(mui) %*% t(zi) %*% residui
    }
    ##M-step
    beta <- as.vector(solve(t(x) %*% x) %*% t(x) %*% (y-u))
    residu <- as.vector(y-x%*%beta)
    sig2 <- (sum(residu^2)-2*C[1] + P)/n
    g <- as.matrix(R/N)
  }
  return(list(beta=beta, g=g, sigma2 = sig2))
}
```

만들어진 function을 이용해서, 10000번 iteration을 했을 때 Beta, G, sigma 제곱의 값의 추정치는 다음과 같다.

```r
EM(y,x,z,uid, 10000)
```

```
$beta
[1] -0.7296889
```

```
$g
          [,1]
[1,] 0.0004261411
```

```
$sigma2
[1] 56.26341
```

3.

EXAM PDF 파일에서의 Utility Function을 보면

$$S_k(x|W_i, A_i)$$

라고 되어 있는 반면, 교수님이 질문 시간에 표기하신 notation은

$$S_k(X_i|W_i, A_i)$$

였기 때문에 EXAM PDF 파일에서의 notation을 이용한 것을 Version1, 교수님이 필기해주신 notation을 이용한 것을 Version2 라고 해서 두 방식으로 풀었습니다.

- Version1

우선 Utility Function을 풀어서 쓰면 다음과 같이 나온다.

$$\begin{aligned}
U(W_i, A_i) = & \Big\{ \eta_{11} \exp(c_1) \int_0^{1.5} \exp\{-(\eta_{11}\exp(c_1)+\eta_{21}\exp(c_2))x\}\,dx \\
& + \eta_{12}\exp(c_1) \int_{1.5}^{3} \exp\{-(\eta_{12}\exp(c_1)+\eta_{22}\exp(c_2))x\}\,dx \Big\} \times 0.5 \\
& + \Big\{ \eta_{21} \exp(c_2) \int_0^{1.5} \exp\{-(\eta_{11}\exp(c_1)+\eta_{21}\exp(c_2))x\}\,dx \\
& + \eta_{22}\exp(c_2) \int_{1.5}^{3} \exp\{-(\eta_{12}\exp(c_1)+\eta_{22}\exp(c_2))x\}\,dx \Big\} \times 5 \\
& + \Big\{ \eta_{13} \exp(c_1) \int_3^{4.5} \exp\{-(\eta_{13}\exp(c_1)+\eta_{23}\exp(c_2))x\}\,dx \\
& + \eta_{14}\exp(c_1) \int_{4.5}^{6} \exp\{-(\eta_{14}\exp(c_1)+\eta_{24}\exp(c_2))x\}\,dx \Big\} \times 10 \\
& + \Big\{ \eta_{23} \exp(c_2) \int_3^{4.5} \exp\{-(\eta_{13}\exp(c_1)+\eta_{23}\exp(c_2))x\}\,dx \\
& + \eta_{24}\exp(c_2) \int_{4.5}^{6} \exp\{-(\eta_{14}\exp(c_1)+\eta_{24}\exp(c_2))x\}\,dx \Big\} \times 20
\end{aligned}$$

$$\text{where } \; c_1 = r_{11} W_i + r_{12} A_i + r_{13} W_i A_i$$

$$c_2 = r_{21} W_i + r_{22} A_i + r_{23} W_i A_i$$

이를 RCPP로 구현하면 다음과 같다.

```
//필요한 함수 setting
double c1(double w, double a){
  double gam_11 = -0.1987954;
  double gam_12 = -0.6744738;
  double gam_13 = 0.1701579;
  double res = exp(gam_11 * w + gam_12 * a + gam_13 * w * a);
  return(res);
}

double c2(double w, double a){
  double gam_21 = -0.2518877;
  double gam_22 = 0.3991342;
  double gam_23 = 0.25158;
  double res = exp(gam_21 * w + gam_22 * a + gam_23 * w * a);
  return(res);
}

double f(double x, double eta1, double eta2, double w, double a){
  double res1 = eta1*c1(w,a) + eta2*c2(w,a);
  double res2 = exp(-res1*x);
  return(res2);
}

double trapezoidal_cpp(NumericVector interval,
                       double n,
                       double eta1,
                       double eta2,
                       double w,
                       double a){
  vec x(n-1);
  vec f_list(n-1);
  double b = interval[0];
  double c = interval[1];
  double h = (c-b)/n;
  for(int i = 1; i < n; i++){
    x[i-1] = b + i*h;
  }
  double low = h/2*f(b, eta1, eta2, w, a);
  double high = h/2*f(c, eta1, eta2, w, a);
  for(int j = 1; j < n; j++){
    f_list[j-1] = f(x[j-1], eta1, eta2, w, a);
  }
  double middle = h*accu(f_list);
  double res = low + high + middle;
  return(res);
}

double first_function(double w, double a, double n){
  double eta_11 = 0.07116282;
  double eta_12 = 0.07766563;
  double eta_21 = 0.02380466;
  double eta_22 = 0.02865413;
  NumericVector inter1 = NumericVector::create(0.0, 1.5);
  NumericVector inter2 = NumericVector::create(1.5,3.0);
  double part1 = eta_11 * c1(w,a) *
    trapezoidal_cpp(inter1, n, eta_11, eta_21, w, a);
  double part2 = eta_12 * c1(w,a) *
    trapezoidal_cpp(inter2, n, eta_12, eta_22, w, a);
  double res = (part1 + part2) * 0.5;
  return(res);
}

double second_function(double w, double a, double n){
  double eta_11 = 0.07116282;
  double eta_12 = 0.07766563;
  double eta_21 = 0.02380466;
  double eta_22 = 0.02865413;
  NumericVector inter1 = NumericVector::create(0.0, 1.5);
  NumericVector inter2 = NumericVector::create(1.5, 3.0);
  double part1 = eta_21 * c2(w,a) *
    trapezoidal_cpp(inter1, n, eta_11, eta_21, w, a);
  double part2 = eta_22 * c2(w,a) *
    trapezoidal_cpp(inter2, n, eta_12, eta_22, w, a);
  double res = (part1 + part2) * 5.0;
  return(res);
}

double third_function(double w, double a, double n){
  double eta_13 = 0.1052774;
  double eta_14 = 0.1061366;
  double eta_23 = 0.03215047;
  double eta_24 = 0.03584044;
  NumericVector inter1 = NumericVector::create(3.0, 4.5);
  NumericVector inter2 = NumericVector::create(4.5, 6.0);
  double part1 = eta_13 * c1(w,a) *
    trapezoidal_cpp(inter1, n, eta_13, eta_23, w, a);
  double part2 = eta_14 * c1(w,a) *
    trapezoidal_cpp(inter2, n, eta_14, eta_24, w, a);
  double res = (part1 + part2) * 10.0;
  return(res);
}

double fourth_function(double w, double a, double n){
  double eta_13 = 0.1052774;
  double eta_14 = 0.1061366;
  double eta_23 = 0.03215047;
  double eta_24 = 0.03584044;
  NumericVector inter1 = NumericVector::create(3.0, 4.5);
  NumericVector inter2 = NumericVector::create(4.5, 6.0);
  double part1 = eta_23 * c2(w,a) *
    trapezoidal_cpp(inter1, n, eta_13, eta_23, w, a);
  double part2 = eta_24 * c2(w,a) *
    trapezoidal_cpp(inter2, n, eta_14, eta_24, w, a);
  double res = (part1 + part2) * 20.0;
  return(res);
}


//trapezoidal function
// [[Rcpp::export]]
double trape(double w, double a, double n){
  double res1 = first_function(w, a, n);
  double res2 = second_function(w, a, n);
  double res3 = third_function(w, a, n);
  double res4 = fourth_function(w, a, n);
  double res = res1 + res2 + res3 + res4;
  return(res);
}


//simpsons도 동일한 방식으로 한다.
double simpsons_cpp(NumericVector interval,
                    double n,
                    double eta1,
                    double eta2,
                    double w,
                    double a){
  vec x(n+1);
  vec out(n/2);
  double b = interval[0];
  double c = interval[1];
  double h = (c-b)/n;
  for(int i = 0; i < n+1; i++){
    x[i] = b + i*h;
  }
  for(int j = 0; j < n/2; j++){
    out[j] = h/3*f(x[2*j], eta1, eta2, w, a);
    out[j] = out[j] + 4*h/3*f(x[2*j+1], eta1, eta2, w, a);
    out[j] = out[j] + h/3*f(x[2*j+2], eta1, eta2, w, a);
  }
  double res = accu(out);
  return(res);
}

double first_function2(double w, double a, double n){
  double eta_11 = 0.07116282;
  double eta_12 = 0.07766563;
  double eta_21 = 0.02380466;
  double eta_22 = 0.02865413;
  NumericVector inter1 = NumericVector::create(0.0, 1.5);
  NumericVector inter2 = NumericVector::create(1.5,3.0);
  double part1 = eta_11 * c1(w,a) *
    simpsons_cpp(inter1, n, eta_11, eta_21, w, a);
  double part2 = eta_12 * c1(w,a) *
    simpsons_cpp(inter2, n, eta_12, eta_22, w, a);
  double res = (part1 + part2) * 0.5;
  return(res);
}

double second_function2(double w, double a, double n){
  double eta_11 = 0.07116282;
  double eta_12 = 0.07766563;
  double eta_21 = 0.02380466;
  double eta_22 = 0.02865413;
  NumericVector inter1 = NumericVector::create(0.0, 1.5);
  NumericVector inter2 = NumericVector::create(1.5, 3.0);
  double part1 = eta_21 * c2(w,a) *
    simpsons_cpp(inter1, n, eta_11, eta_21, w, a);
  double part2 = eta_22 * c2(w,a) *
    simpsons_cpp(inter2, n, eta_12, eta_22, w, a);
  double res = (part1 + part2) * 5.0;
  return(res);
}

double third_function2(double w, double a, double n){
  double eta_13 = 0.1052774;
  double eta_14 = 0.1061366;
  double eta_23 = 0.03215047;
  double eta_24 = 0.03584044;
  NumericVector inter1 = NumericVector::create(3.0, 4.5);
  NumericVector inter2 = NumericVector::create(4.5, 6.0);
  double part1 = eta_13 * c1(w,a) *
    simpsons_cpp(inter1, n, eta_13, eta_23, w, a);
  double part2 = eta_14 * c1(w,a) *
    simpsons_cpp(inter2, n, eta_14, eta_24, w, a);
  double res = (part1 + part2) * 10.0;
  return(res);
```

```
}

double fourth_function2(double w, double a, double n){
  double eta_13 = 0.1052774;
  double eta_14 = 0.1061366;
  double eta_23 = 0.03215047;
  double eta_24 = 0.03584044;
  NumericVector inter1 = NumericVector::create(3.0, 4.5);
  NumericVector inter2 = NumericVector::create(4.5, 6.0);
  double part1 = eta_23 * c2(w,a) *
    simpsons_cpp(inter1, n, eta_13, eta_23, w, a);
  double part2 = eta_24 * c2(w,a) *
    simpsons_cpp(inter2, n, eta_14, eta_24, w, a);
  double res = (part1 + part2) * 20.0;
  return(res);
}

// [[Rcpp::export]]
double simpson(double w, double a, double n){
  double res1 = first_function2(w, a, n);
  double res2 = second_function2(w, a, n);
  double res3 = third_function2(w, a, n);
  double res4 = fourth_function2(w, a, n);
  double res = res1 + res2 + res3 + res4;
  return(res);
}
```

그리고 이를 통해 주어진 clinic data의 처음 100개 data의 utility function 값을 구하면 다음과 같다. (결과값은 'q3_list1.csv' 파일에 저장했다.)

```
trape_list1 <- c()
simpson_list1 <- c()
for(i in 1:100){
  trape_list1[i] <- trape(clinic_df[i,1], clinic_df[i,2], 1000)
  simpson_list1[i] <- simpson(clinic_df[i,1], clinic_df[i,2], 1000)
}

list1_df <- data.frame(trape = trape_list1, simpson = simpson_list1)
```

```
list1_df
      trape    simpson
1   4.159748 4.159748
2   3.476871 3.476871
3   3.476871 3.476871
4   4.159748 4.159748
5   2.870668 2.870668
6   2.870668 2.870668
7   3.476871 3.476871
8   3.476871 3.476871
9   3.476871 3.476871
10  2.870668 2.870668
11  3.476871 3.476871
12  3.476871 3.476871
13  4.159748 4.159748
14  2.870668 2.870668
```

```
15  3.476871 3.476871
16  3.476871 3.476871
17  3.476871 3.476871
18  3.466787 3.466787
19  3.466787 3.466787
20  4.159748 4.159748
21  2.870668 2.870668
22  2.870668 2.870668
23  2.870668 2.870668
24  3.476871 3.476871
25  4.159748 4.159748
26  2.870668 2.870668
27  4.159748 4.159748
28  2.870668 2.870668
29  4.159748 4.159748
30  3.476871 3.476871
31  3.466787 3.466787
32  3.476871 3.476871
33  4.159748 4.159748
34  4.159748 4.159748
35  3.466787 3.466787
36  4.159748 4.159748
37  2.870668 2.870668
38  3.476871 3.476871
39  3.476871 3.476871
40  4.159748 4.159748
41  4.159748 4.159748
42  4.159748 4.159748
43  4.159748 4.159748
44  4.159748 4.159748
45  4.159748 4.159748
46  3.476871 3.476871
47  3.466787 3.466787
48  3.476871 3.476871
49  4.159748 4.159748
50  2.870668 2.870668
51  2.870668 2.870668
52  2.870668 2.870668
53  2.870668 2.870668
54  2.870668 2.870668
55  3.466787 3.466787
56  2.870668 2.870668
57  2.870668 2.870668
58  2.870668 2.870668
59  2.870668 2.870668
60  4.159748 4.159748
61  4.159748 4.159748
62  2.870668 2.870668
63  2.870668 2.870668
64  3.476871 3.476871
65  3.466787 3.466787
66  2.870668 2.870668
```

```
67  4.159748 4.159748
68  4.159748 4.159748
69  2.870668 2.870668
70  3.476871 3.476871
71  4.159748 4.159748
72  3.476871 3.476871
73  4.159748 4.159748
74  4.159748 4.159748
75  4.159748 4.159748
76  3.466787 3.466787
77  4.159748 4.159748
78  2.870668 2.870668
79  2.870668 2.870668
80  2.870668 2.870668
81  3.476871 3.476871
82  3.466787 3.466787
83  3.476871 3.476871
84  4.159748 4.159748
85  4.159748 4.159748
86  3.466787 3.466787
87  3.476871 3.476871
88  3.466787 3.466787
89  2.870668 2.870668
90  2.870668 2.870668
91  4.159748 4.159748
92  3.476871 3.476871
93  3.476871 3.476871
94  3.466787 3.466787
95  3.476871 3.476871
96  3.466787 3.466787
97  2.870668 2.870668
98  3.466787 3.466787
99  3.466787 3.466787
100 3.466787 3.466787
```

- Version2

Utility Function을 전개하면 다음과 같다.

$$U(w_i, A_i) = \left[\int_0^3 (S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i)) \lambda_1(x|w_i, A_i) dx\right] \times O_1$$

$$+ \left[\int_0^3 (S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i)) \lambda_2(x|w_i, A_i) dx\right] \times O_2$$

$$+ \left[\int_3^6 (S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i)) \lambda_1(x|w_i, A_i) dx\right] \times O_3$$

$$+ \left[\int_3^6 (S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i)) \lambda_2(x|w_i, A_i) dx\right] \times O_4.$$

$$S_1(x_i|w_i, A_i) = \exp\left\{-\int_0^{x_i} \lambda_1(x|w_i, A_i) dx\right\}$$

$$= \exp\left\{-\int_0^{x_i} \eta_1(x) \exp(\gamma_{11} w_i + \gamma_{12} A_i + \gamma_{13} w_i A_i) dx\right\}.$$

$$\eta_1(x) = \eta_{11} I(0 < x \leq 1.5) + \eta_{12} I(1.5 < x \leq 3) + \eta_{13} I(3 < x \leq 4.5) + \eta_{14} I(4.5 < x \leq 6)$$

if $0 < x_i \leq 1.5$

$$\exp\left\{-\int_0^{x_i} \eta_{11} C_1\right\} = \exp\{-\eta_{11} C_1 x_i\} \qquad (\text{let } C_1 = \exp(r_{11} w_i + r_{12} A_i + r_{13} w_i A_i))$$

if $1.5 < x_i \leq 3$

$$\exp\left\{-\int_0^{1.5} \eta_{11} C_1 dx - \int_{1.5}^{x_i} \eta_{12} \quad dx\right\}$$

$$= \exp\{-\eta_{11} C_1 \times 1.5 - \eta_{12} C_1 \times x_i + \eta_{12} C_1 \times 1.5\}$$

if $3 < x_i \leq 4.5$

$$\exp\{-1.5\eta_{11} C_1 - 1.5\eta_{12} C_1 - \eta_{13} C_1 x_i + 3\eta_{13} C_1\}$$

if $4.5 < x_i \leq 6$

$$\exp\{-1.5\eta_{11} C_1 - 1.5\eta_{12} C_1 - 1.5\eta_{13} C_1 - \eta_{14} C_1 x_i + 4.5\eta_{14} C_1\}$$

$\therefore$ if $0 < x_i \leq 1.5$

$$S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i) = \exp\{-\eta_{11} C_1 x_i - \eta_{21} C_2 x_i\} \qquad (\text{let } C_2 = \exp(r_{21} w_i + r_{22} A_i + r_{23} w_i A_i))$$

if $1.5 < x_i \leq 3$

$$S_1(x_i|w_i, A_i) S_2(x_i|w_i, A_i) = \exp\{-1.5\eta_{11} C_1 - 1.5\eta_{21} C_2 + 1.5\eta_{12} C_1 + 1.5\eta_{22} C_2 - \eta_{12} C_1 x_i - \eta_{22} C_2 x_i\}$$

if $3 < x_i \le 4.5$

$$S_1(x_i|w_i,A_i)S_2(x_i|w_i,A_i) = \exp\{-1.5(\eta_{11}C_1+\eta_{21}C_2) - 1.5(\eta_{12}C_1+\eta_{22}C_2) + 3(\eta_{13}C_1+\eta_{23}C_2) - x_i(\eta_{13}C_1+\eta_{23}C_2)\}$$

if $4.5 < x_i \le 6$

$$S_1(x_i|w_i,A_i)S_2(x_i|w_i,A_i) = \exp\{-1.5(\eta_{11}C_1+\eta_{21}C_2) - 1.5(\eta_{12}C_1+\eta_{22}C_2) - 1.5(\eta_{13}C_1+\eta_{23}C_2) + 4.5(\eta_{14}C_1+\eta_{24}C_2)$$
$$- x_i(\eta_{14}C_1+\eta_{24}C_2)\}$$

$$S_1(x_i|w_i,A_i)\,S_2(x_i|w_i,A_i) \overset{\text{let}}{=} S$$

$$\begin{aligned}
U(w_i,A_i) &= \left[\int_0^3 S\times\lambda_1(x|w_i,A_i)\,dx\right]\times O_1 \\
&\quad + \left[\int_0^3 S\times\lambda_2(x|w_i,A_i)\,dx\right]\times O_2 \\
&\quad + \left[\int_3^6 S\times\lambda_1(x|w_i,A_i)\,dx\right]\times O_3 \\
&\quad + \left[\int_3^6 S\times\lambda_2(x|w_i,A_i)\,dx\right]\times O_4 \\
&= \left[\int_0^{1.5} S\times\lambda_1(x|w_i,A_i)\,dx + \int_{1.5}^3 S\times\lambda_1(x|w_i,A_i)\right]\times O_1 \\
&\quad + \left[\int_0^{1.5} S\times\lambda_2(x|w_i,A_i)\,dx + \int_{1.5}^3 S\times\lambda_2(x|w_i,A_i)\right]\times O_2 \\
&\quad + \left[\int_3^{4.5} S\times\lambda_1(x|w_i,A_i) + \int_{4.5}^6 S\times\lambda_1(x|w_i,A_i)\right]\times O_3 \\
&\quad + \left[\int_3^{4.5} S\times\lambda_2(x|w_i,A_i) + \int_{4.5}^6 S\times\lambda_2(x|w_i,A_i)\right]\times O_4 \\
&= \int_0^{1.5} 0.5\times S\times\lambda_1(x|w_i,A_i) + 5\times S\times\lambda_2(x|w_i,A_i)\,dx \\
&\quad + \int_{1.5}^{3.0} 0.5\times S\times\lambda_1(x|w_i,A_i) + 5\times S\times\lambda_2(x|w_i,A_i)\,dx \\
&\quad + \int_{3.0}^{4.5} 10\times S\times\lambda_1(x|w_i,A_i) + 20\times S\times\lambda_2(x|w_i,A_i)\,dx \\
&\quad + \int_{4.5}^{6.0} 10\times S\times\lambda_1(x|w_i,A_i) + 20\times S\times\lambda_2(x|w_i,A_i)\,dx
\end{aligned}$$

$$\begin{aligned}
&= \int_0^{1.5} 0.5\times\eta_{11}\times C_1\times S + 5\times\eta_{21}\times C_2\times S\,dx \\
&\quad + \int_{1.5}^3 0.5\times\eta_{12}\times C_1\times S + 5\times\eta_{22}\times C_2\times S\,dx \\
&\quad + \int_3^{4.5} 10\times\eta_{13}\times C_1\times S + 20\times\eta_{23}\times C_2\times S\,dx \\
&\quad + \int_{4.5}^6 10\times\eta_{14}\times C_1\times S + 20\times\eta_{24}\times C_2\times S\,dx
\end{aligned}$$

이를 RCPP로 구현해보자.

```
double c1(double w, double a){
  double gam_11 = -0.1987954;
  double gam_12 = -0.6744738;
  double gam_13 = 0.1701579;
  double res = exp(gam_11 * w + gam_12 * a + gam_13 * w * a);
  return(res);
}

double c2(double w, double a){
  double gam_21 = -0.2518877;
  double gam_22 = 0.3991342;
  double gam_23 = 0.25158;
  double res = exp(gam_21 * w + gam_22 * a + gam_23 * w * a);
  return(res);
}

double s(double x, double w, double a){
  double res;
  if(x>0 && x<=1.5){
    res = exp(-0.07116282*c1(w,a)*x -0.02380466*c2(w,a)*x);
  }
  else if(x>1.5 && x<=3){
    res = exp(-1.5*c1(w,a)*(0.07116282-0.07766563) - 1.5*c2(w,a)*(0.02380466-
0.02865413) -
      0.07766563*c1(w,a)*x - 0.02865413*c2(w,a)*x);
  }
  else if(x>3 && x<=4.5){
    res = exp(-1.5*c1(w,a)*(0.07116282+0.07766563) - 1.5*c2(w,a)*
(0.02380466+0.02865413) +
      3*(0.1052774*c1(w,a)+0.03215047*c2(w,a)) - x*
(0.1052774*c1(w,a)+0.03215047*c2(w,a)));
  }
  else{
    res = exp(-1.5*c1(w,a)*(0.07116282+0.07766563+0.1052774) - 1.5*c2(w,a)*
(0.02380466+0.02865413+0.03215047) +
      4.5*(0.1061366*c1(w,a) + 0.03584044*c2(w,a)) - x*(0.1061366*c1(w,a) +
0.03584044*c2(w,a)));
  }
  return(res);
}

double f1(double x, double t, double w, double a){
  double eta_11 = 0.07116282;
  double eta_21 = 0.02380466;
  double res1 = 0.5 * eta_11 * c1(w,a) * s(t,w,a);
  double res2 = 5.0 * eta_21 * c2(w,a) * s(t,w,a);
  double res = res1+res2;
  return(res);
}

double f2(double x, double t, double w, double a){
  double eta_12 = 0.07766563;
  double eta_22 = 0.02865413;
  double res1 = 0.5 * eta_12 * c1(w,a) * s(t,w,a);
  double res2 = 5.0 * eta_22 * c2(w,a) * s(t,w,a);
  double res = res1+res2;
  return(res);
}

double f3(double x, double t, double w, double a){
  double eta_13 = 0.1052774;
  double eta_23 = 0.03215047;
  double res1 = 10.0 * eta_13 * c1(w,a) * s(t,w,a);
  double res2 = 20.0 * eta_23 * c2(w,a) * s(t,w,a);
  double res = res1+res2;
  return(res);
}

double f4(double x, double t, double w, double a){
  double eta_14 = 0.1061366;
  double eta_24 = 0.03584044;
  double res1 = 10.0 * eta_14 * c1(w,a) * s(t,w,a);
  double res2 = 20.0 * eta_24 * c2(w,a) * s(t,w,a);
  double res = res1+res2;
  return(res);
}

// [[Rcpp::export]]
double trape2(double t, double w, double a, int n){
  double h = 1.5/n;
  double fsum;

  fsum = 0;
  for(int i = 1; i < n; i++){
    fsum = fsum + f1(i*h, t, w, a);
  }
  double res1 = h*(fsum + 0.5 * f1(0,t,w,a) + 0.5 * f1(1.5,t,w,a));

  fsum = 0;
  for(int i = 1; i < n; i++){
    fsum = fsum + f2((1.5+i*h),t, w, a);
  }
  double res2 = h*(fsum + 0.5 * f2(1.5,t,w,a) + 0.5 * f2(3.0,t,w,a));

  fsum = 0;
  for(int i = 1; i < n; i++){
    fsum = fsum + f3((3.0+i*h),t, w, a);
  }
  double res3 = h*(fsum + 0.5 * f3(3.0,t,w,a) + 0.5 * f3(4.5,t,w,a));

  fsum = 0;
  for(int i = 1; i < n; i++){
    fsum = fsum + f4((4.5+i*h),t, w, a);
  }
  double res4 = h*(fsum + 0.5 * f4(4.5,t,w,a) + 0.5 * f3(6.0,t,w,a));

  double res = res1+res2+res3+res4;
  return(res);
}

// [[Rcpp::export]]
double simpson2(double t, double w, double a, int n){
  double h = 1.5/n;
  double fsum1;
  double fsum2;
  double fsum3;

  fsum1 = 0;
  fsum2 = 0;
  fsum3 = 0;
  for(int i = 1; i < (n/2); i++){
    fsum1 = fsum1 + f1((2*i-2)*h, t, w, a);
    fsum2 = fsum2 + 4.0 * f1((2*i-1)*h, t, w, a);
    fsum3 = fsum3 + f1((2*i)*h, t, w, a);
  }
  double res1 = h*(fsum1 + fsum2 + fsum3)/3.0;

  fsum1 = 0;
  fsum2 = 0;
  fsum3 = 0;
  for(int i = 1; i < (n/2); i++){
    fsum1 = fsum1 + f2(1.5+(2*i-2)*h, t, w, a);
    fsum2 = fsum2 + 4.0 * f2(1.5+(2*i-1)*h, t, w, a);
    fsum3 = fsum3 + f2(1.5+(2*i)*h, t, w, a);
  }
  double res2 = h*(fsum1 + fsum2 + fsum3)/3.0;

  fsum1 = 0;
  fsum2 = 0;
  fsum3 = 0;
  for(int i = 1; i < (n/2); i++){
    fsum1 = fsum1 + f3(3.0+(2*i-2)*h, t, w, a);
    fsum2 = fsum2 + 4.0 * f3(3.0+(2*i-1)*h, t, w, a);
    fsum3 = fsum3 + f3(3.0+(2*i)*h, t, w, a);
  }
  double res3 = h*(fsum1 + fsum2 + fsum3)/3.0;

  fsum1 = 0;
  fsum2 = 0;
  fsum3 = 0;
  for(int i = 1; i < (n/2); i++){
    fsum1 = fsum1 + f4(4.5+(2*i-2)*h, t, w, a);
    fsum2 = fsum2 + 4.0 * f4(4.5+(2*i-1)*h, t, w, a);
    fsum3 = fsum3 + f4(4.5+(2*i)*h, t, w, a);
  }
  double res4 = h*(fsum1 + fsum2 + fsum3)/3.0;

  double res = res1+res2+res3+res4;
  return(res);
}
```

그리고 이를 통해 주어진 clinic data의 처음 100개 data의 utility function 값을 구하면 다음과 같다. (결과값은 'q3_list2.csv' 파일에 저장했다.)

```
trape_list2 <- c()
simpson_list2 <- c()
for(i in 1:100){
  trape_list2[i] <- trape2(clinic_df[i,4],clinic_df[i,1], clinic_df[i,2], 1000)
  simpson_list2[i] <- simpson2(clinic_df[i,4],clinic_df[i,1], clinic_df[i,2],
1000)
}

list2_df <- data.frame(trape = trape_list2, simpson = simpson_list2)
```

```
list2_df
      trape  simpson
1   3.640627 3.633418
2   3.073878 3.067780
3   3.073878 3.067780
4   3.640627 3.633418
5   2.556955 2.551868
6   2.876558 2.870835
7   3.073878 3.067780
8   3.073878 3.067780
9   3.073878 3.067780
10  2.556955 2.551868
11  3.278981 3.272476
12  4.783927 4.774437
13  5.570043 5.559013
14  2.556955 2.551868
15  3.073878 3.067780
16  3.073878 3.067780
17  3.073878 3.067780
18  4.035060 4.027056
19  3.069860 3.063770
20  3.640627 3.633418
21  3.860613 3.852933
22  2.556955 2.551868
23  2.556955 2.551868
24  3.073878 3.067780
25  3.640627 3.633418
26  2.556955 2.551868
27  3.640627 3.633418
28  3.206946 3.200566
29  4.680640 4.671372
30  3.073878 3.067780
31  3.849679 3.842042
32  3.073878 3.067780
33  5.475167 5.464325
34  3.640627 3.633418
35  4.337820 4.329215
36  3.640627 3.633418
37  3.252514 3.246043
38  3.582135 3.575028
39  3.073878 3.067780
40  3.655609 3.648371
41  3.640627 3.633418
42  4.063188 4.055143
43  3.640627 3.633418
44  3.764313 3.756859
45  4.579508 4.570440
46  3.073878 3.067780
47  3.158022 3.151757
48  3.073878 3.067780
49  3.640627 3.633418
50  3.869656 3.861957
```

```
51  2.556955 2.551868
52  3.756409 3.748936
53  2.556955 2.551868
54  2.556955 2.551868
55  4.958211 4.948375
56  2.556955 2.551868
57  3.148326 3.142062
58  2.556955 2.551868
59  2.556955 2.551868
60  4.000229 3.992308
61  3.640627 3.633418
62  2.556955 2.551868
63  2.668485 2.663176
64  3.073878 3.067780
65  3.069860 3.063770
66  4.288610 4.280078
67  3.640627 3.633418
68  3.640627 3.633418
69  3.568798 3.561698
70  3.623803 3.616614
71  5.276845 5.266396
72  3.073878 3.067780
73  5.408689 5.397980
74  3.640627 3.633418
75  3.640627 3.633418
76  3.069860 3.063770
77  3.640627 3.633418
78  2.556955 2.551868
79  2.556955 2.551868
80  2.960945 2.955055
81  3.073878 3.067780
82  3.415244 3.408469
83  3.073878 3.067780
84  3.640627 3.633418
85  3.713398 3.706045
86  3.069860 3.063770
87  4.563031 4.553979
88  4.415656 4.406897
89  2.556955 2.551868
90  2.556955 2.551868
91  3.640627 3.633418
92  3.073878 3.067780
93  4.379461 4.370772
94  3.069860 3.063770
95  3.073878 3.067780
96  3.153690 3.147434
97  2.556955 2.551868
98  5.168819 5.158565
99  3.069860 3.063770
100 3.256576 3.250116
```

4.

우선 cancer 데이터를 불러오고, cancer 데이터에서 0은 -1로, 2는 0으로 치환하자.

```r
##data
uscancer <- readLines('UScancer.txt')
uscancer <- as.data.frame(uscancer)
dim(uscancer)
length(strsplit(as.character(uscancer[1,]), split='')[[1]])

cancer <- matrix(NA, nrow=58, ncol = 66)
for(i in 1:58){
  charc <- strsplit(as.character(uscancer[i,]), split='')[[1]]
  num <- as.numeric(charc)
  cancer[i,] <- num
}

cancer[cancer==0] <- -1
cancer[cancer==2] <- 0
```

그 다음 RCPP를 이용해 S2값을 구해주는 function을 만들자.

```cpp
// [[Rcpp::export]]
mat zero_pad(mat dat){
  mat zero_mat = dat;
  zero_mat.insert_cols(0,1);
  zero_mat.insert_cols(dat.n_cols+1, 1);
  zero_mat.insert_rows(0,1);
  zero_mat.insert_rows(dat.n_rows+1, 1);
  return zero_mat;
}

// [[Rcpp::export]]
double neigh_sum(mat dat){
  mat zero_mat = zero_pad(dat);
  mat neigh_mat = zero_pad(dat);
  for(int i=1; i<(neigh_mat.n_rows-1); i++){
    for(int j=1; j<(neigh_mat.n_cols-1); j++){
      neigh_mat(i,j) = zero_mat(i,j)*(zero_mat(i-1,j) +
        zero_mat(i+1,j) + zero_mat(i,j-1) + zero_mat(i, j+1));
    }
  }
  double res = accu(neigh_mat);
  return(res);
}
```

그 다음 s2값을 구해주는 function을 이용하여 Bootstrap sampling 한 개를 만들어주는 function을 우선 만들자.

bootstrap을 했을 때 초기 값에 영향을 받지 않도록 10개를 sampling 했을 때 하나의 sample을 얻을 수 있도록 하였다.

```
// [[Rcpp::export]]
mat boot(double alpha, double beta, mat dat){
  mat temp_mat = dat;
  mat u_mat(1,1);
  double u;
  for(int i=1; i<11; i++){
    for(int j=0; j<dat.n_rows; j++){
      for(int k=0; k<dat.n_cols; k++){
        if(temp_mat(j,k)==0){
          temp_mat(j,k) = 0;
        }
        else{
          temp_mat(j,k) = 1;
          double dens1 = exp(alpha * accu(temp_mat) +
0.5*beta*neigh_sum(temp_mat));
          temp_mat(j,k) = -1;
          double dens2 = exp(alpha * accu(temp_mat) +
0.5*beta*neigh_sum(temp_mat));
          double r = dens1/(dens1+dens2);
          u_mat.randu();
          u = u_mat(0,0);
          if(u < r){
            temp_mat(j,k)=1;
          }
          else{
            temp_mat(j,k)=-1;
          }
        }
      }
    }
  }
  return(temp_mat);
}
```

다음으로, iteration 숫자만큼 sampling한 s1, s2값을 얻을 수 있는 function을 만들었다.

```
// [[Rcpp::export]]
DataFrame bootdf(int iteration, double alpha, double beta, mat dat){
  vector<double> s1;
  vector<double> s2;
  mat temp_mat;
  for(int i = 0; i < iteration; i++){
    temp_mat = boot(alpha, beta, dat);
    s1.push_back(accu(temp_mat));
    s2.push_back(neigh_sum(temp_mat));
  }
  DataFrame res = DataFrame::create(Named("s1") = s1,
                                    Named("s2") = s2);
  return(res);
}
```

이제 r에서 주어진 alpha와 beta를 가지고 1000번의 bootstrapping 한 것을 구하였다.

```
mple_df <- bootdf(1000, -0.3205, 0.1115, cancer)
dmh_df <- bootdf(1000, -0.3030, 0.1227, cancer)
aex_df <- bootdf(1000, -0.3017, 0.1224, cancer)
```

주어진 결과를 이용하여 MPLE, DMH, AEX 세 방식의 s1과 s2의 RMSE값을 dataframe으로 만들었다.

```
s1_obs <- sum(cancer)
s2_obs <- neigh_sum(cancer)

mple_s1_bias2 <- sum((mple_df$s1-s1_obs)^2)/1000
mple_s1_var <- sum((mple_df$s1-mean(mple_df$s1))^2)/1000
mple_s1 <- sqrt(mple_s1_bias2+mple_s1_var)

dmh_s1_bias2 <- sum((dmh_df$s1-s1_obs)^2)/1000
dmh_s1_var <- sum((dmh_df$s1-mean(dmh_df$s1))^2)/1000
dmh_s1 <- sqrt(dmh_s1_bias2+dmh_s1_var)

aex_s1_bias2 <- sum((aex_df$s1-s1_obs)^2)/1000
aex_s1_var <- sum((aex_df$s1-mean(aex_df$s1))^2)/1000
aex_s1 <- sqrt(aex_s1_bias2+aex_s1_var)

mple_s2_bias2 <- sum((mple_df$s2-s2_obs)^2)/1000
mple_s2_var <- sum((mple_df$s2-mean(mple_df$s2))^2)/1000
mple_s2 <- sqrt(mple_s2_bias2+mple_s2_var)

dmh_s2_bias2 <- sum((dmh_df$s2-s2_obs)^2)/1000
dmh_s2_var <- sum((dmh_df$s2-mean(dmh_df$s2))^2)/1000
dmh_s2 <- sqrt(dmh_s2_bias2+dmh_s2_var)

aex_s2_bias2 <- sum((aex_df$s2-s2_obs)^2)/1000
aex_s2_var <- sum((aex_df$s2-mean(aex_df$s2))^2)/1000
aex_s2 <- sqrt(aex_s2_bias2+aex_s2_var)

rmse_df <- data.frame(mple = c(mple_s1, mple_s2), dmh = c(dmh_s1, dmh_s2), aex =
c(aex_s1, aex_s2))
rownames(rmse_df) <- c('S1', 'S2')
rmse_df
```

> ```
> rmse_df
>         mple       dmh      aex
> S1  74.39391  74.67163  72.3243
> S2 284.75499 263.10962 264.7592
> ```

s1의 경우 aex가 제일 낮으며, s2의 경우 dmh 방식이 제일 낮다. 제일 좋지 않은 방식은 mple 방식이라는 것을 알 수 있다.