함수추정의 응용 및 실습

Assignment #2

서울대학교 통계학과 2017-11362 박건도

2022년 05월 02일

1. Nadaraya-Watson estimator

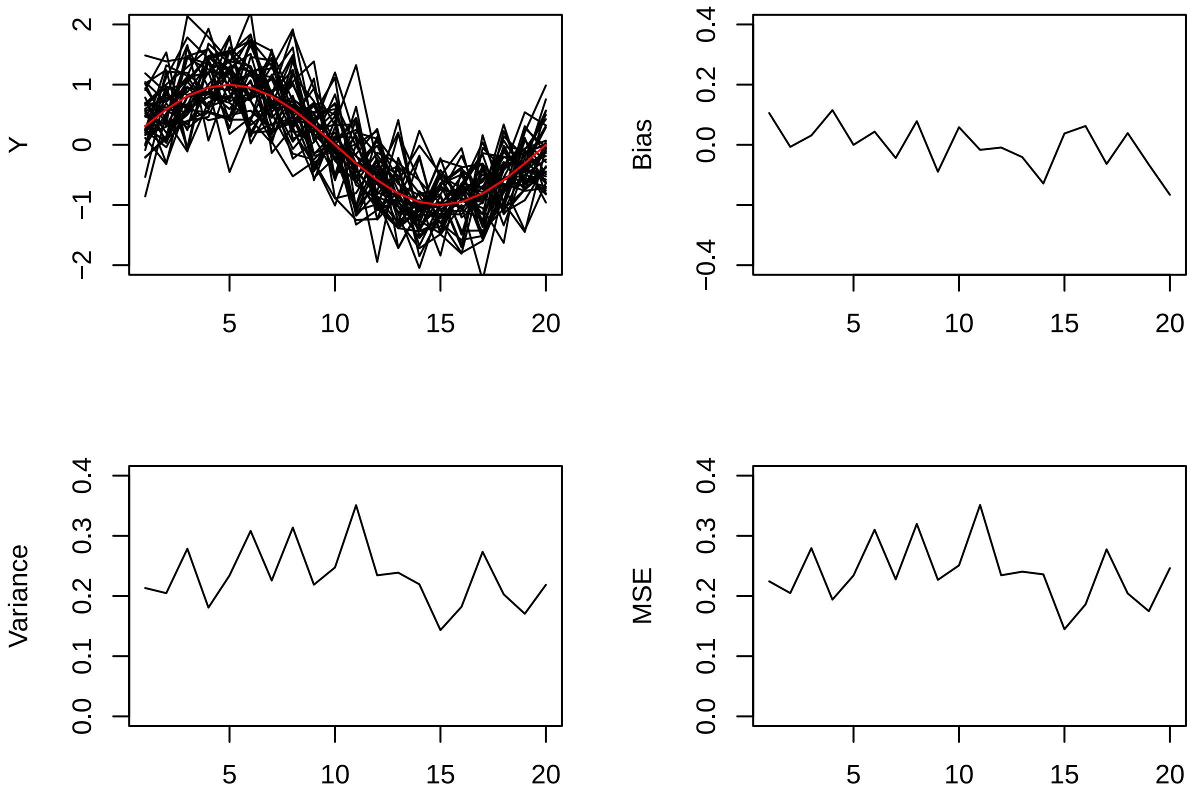
(a) bias-variance trade-off R object

```r
bias_var_trade_off <- function(X, Y, bw){
  n <- length(X)
  par(mfrow=c(2,2), mar=c(2,4,4,2))
  plot(1, type='n', xlim=range(X), ylim=c(-2, 2),
       xlab='X', ylab='Y')

  bwsplus <- bw/0.3708159

  Bias <- rep(0, n)
  MSE <- rep(0, n)

  for (i in 1:40){
    eps <- rnorm(n, sd=0.5)
    Yi <- Y + eps
    fit.ks <- ksmooth(X, Yi, kernel = "normal",
                      bandwidth = bwsplus, x.points = X)
    lines(X, fit.ks$y)
    Bias <- Bias + fit.ks$y
    MSE <- MSE + (fit.ks$y - Y)^2
  }
  lines(X, Y, col="red")

  Bias <- Bias / 40 - Y
  plot(X, Bias, type='l', ylim=c(-0.4,0.4))
  Variance <- MSE / 40 - Bias^2
  plot(X, Variance, type='l', ylim=c(0,0.4))

  MSE <- MSE / 40
  plot(X, MSE, type='l', ylim=c(0,0.4))
}
```

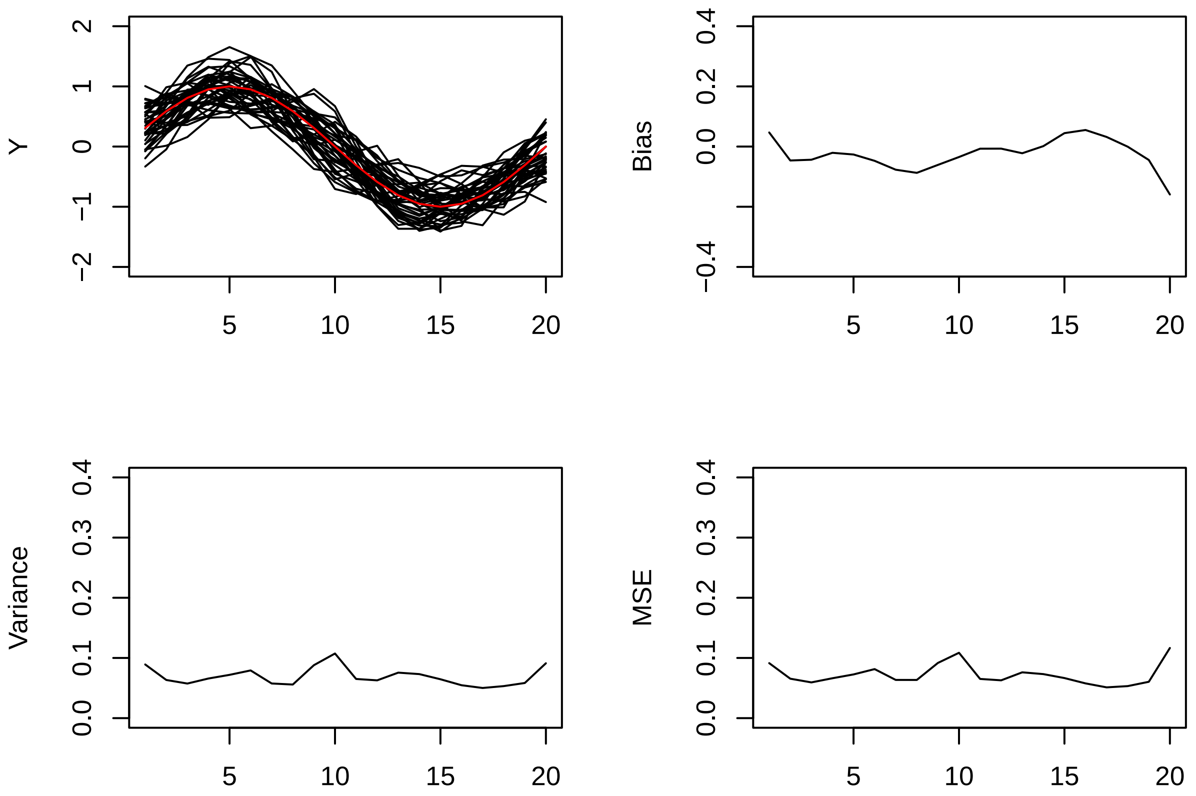
(b) Influence of the bandwidth

먼저 bandwidth의 변화에 따른 Bias, Var, MSE의 변화를 살펴보자.

```
X <- 1:20
Y1 <- sin(0.1 * pi * X)

bias_var_trade_off(X, Y1, 0.18)
```



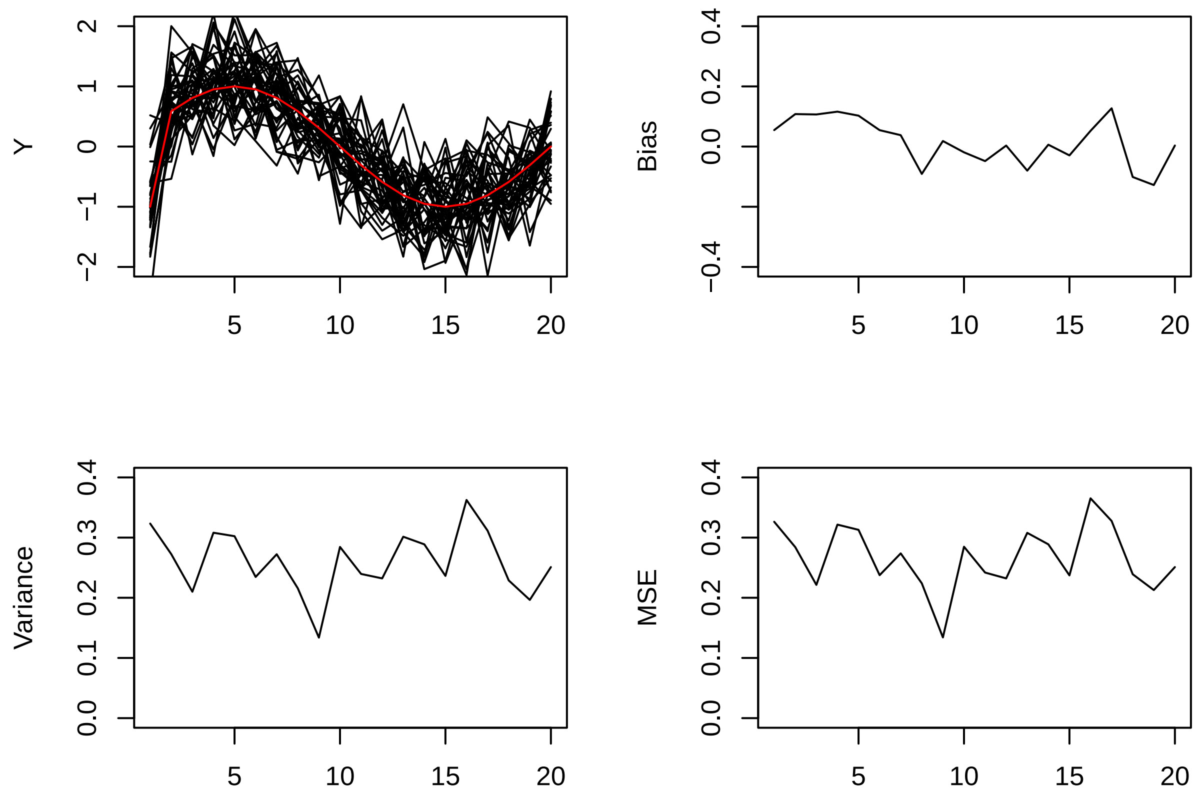
```
bias_var_trade_off(X, Y1, 1)
```



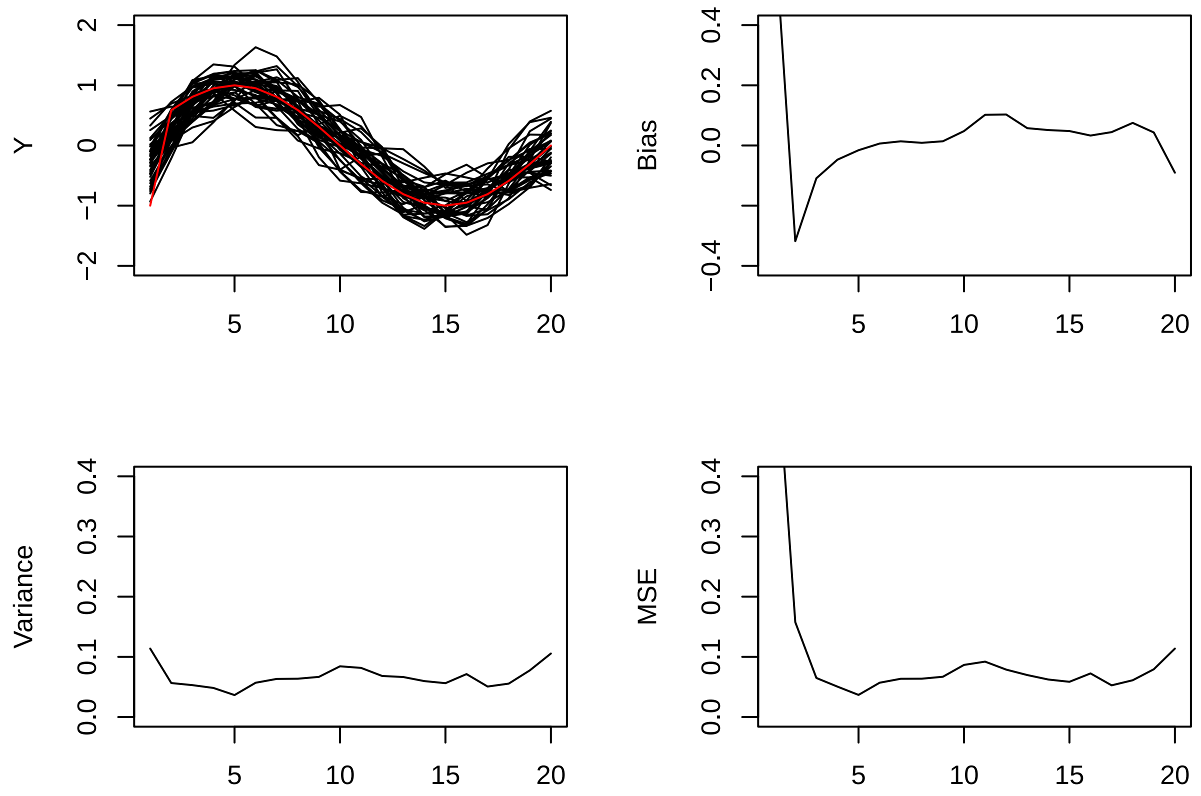
bandwidth가 작을수록 Bias는 비교적 0 근방에서 움직이고, Var은 비교적 크며 MSE 또한 비교적 크다. 반대로 bandwidth가 크면 Bias는 크게 변동하지만, Var과 MSE는 비교적 작다.

다음으로, data의 appearance에 따른 값의 변화를 살펴보자. 위의 모형에서 일부 값 (한쪽 끝, 중간) 을 변경한 후 그래프를 그려보았다.

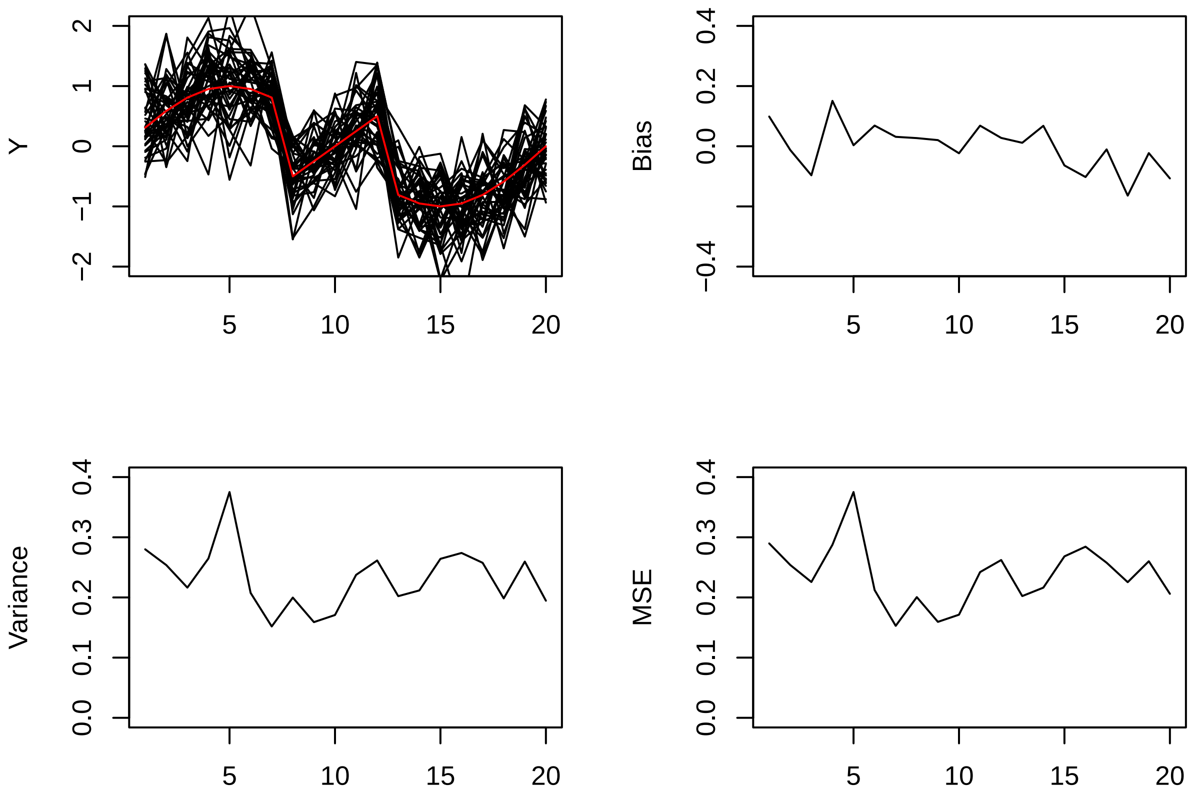
```
Y2 <- Y1
Y2[1] <- -1
bias_var_trade_off(X, Y2, 0.18)
```



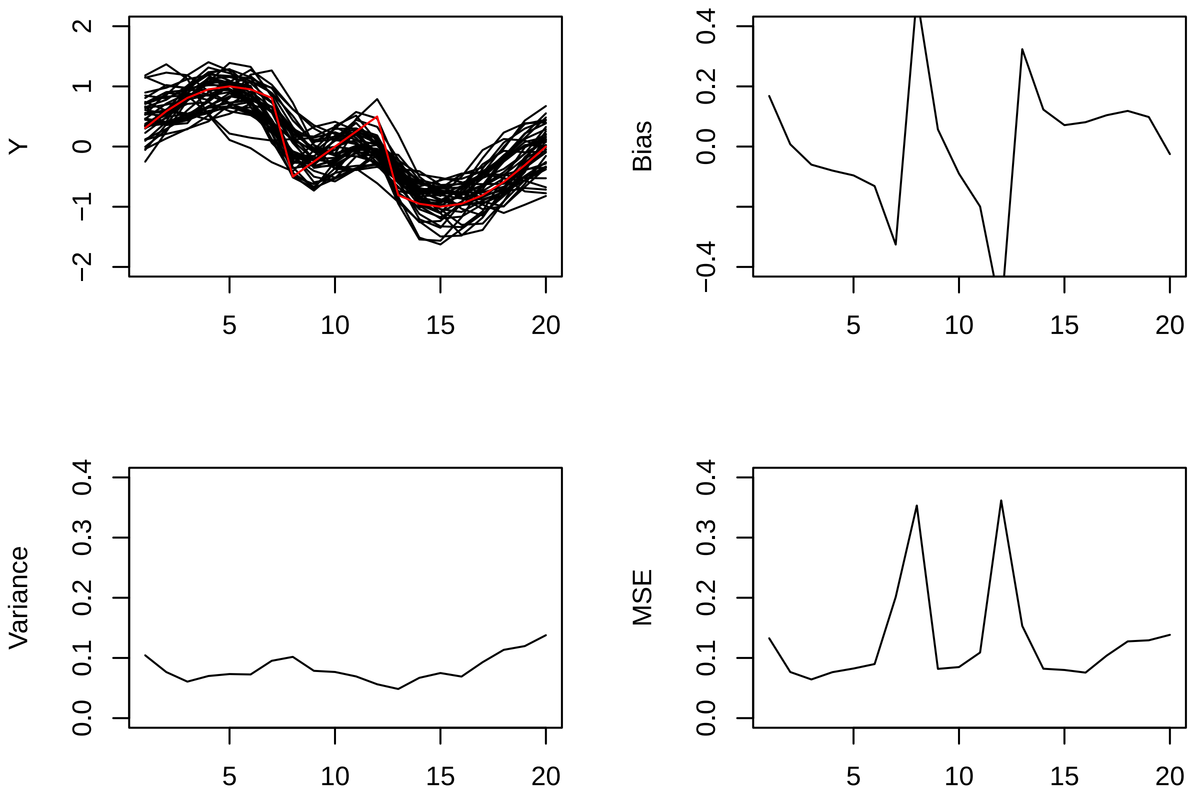
```
bias_var_trade_off(X, Y2, 1)
```



```r
Y3 <- Y1
Y3[8:12] <- seq(-0.5, 0.5, length=5)
bias_var_trade_off(X, Y3, 0.18)
```



```r
bias_var_trade_off(X, Y3, 1)
```



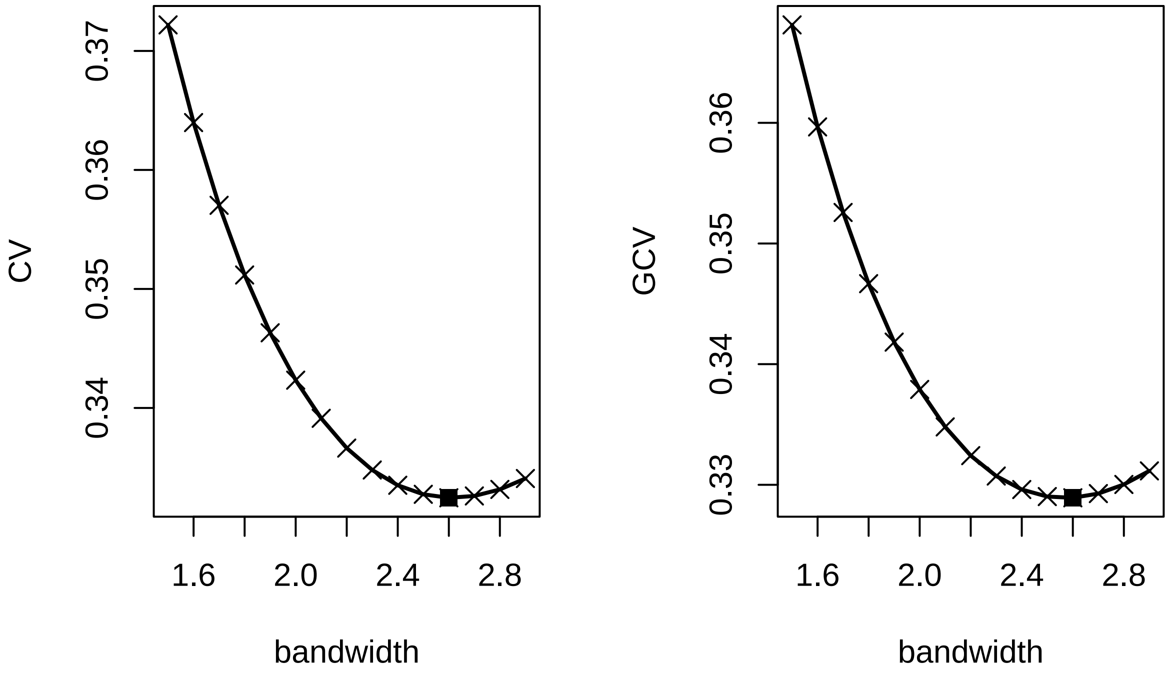
bandwidth를 0.18로 설정한 경우, 두 변형된 data 모두 비슷하게 잘 따라가는 모습을 보였으며, Bias, Var, MSE 모두 초기의 data와 유사한 값을 나타냈다. 하지만 bandwidth를 1로 설정한 경우, 변경된 값 근처의 point에서 Bias와 MSE가 급격히 커지는 현상이 발견되었다. Var 또한 미세하게 증가하였다.

마지막으로, 양 끝점에서의 값을 관찰하면, bandwidth가 작은 경우 평균과 비슷한 값을 보였다. 반면에 bandwidth가 큰 경우에는 양 극단에서 소폭 상승하는 모습을 보인다.

(c) CV, GCV

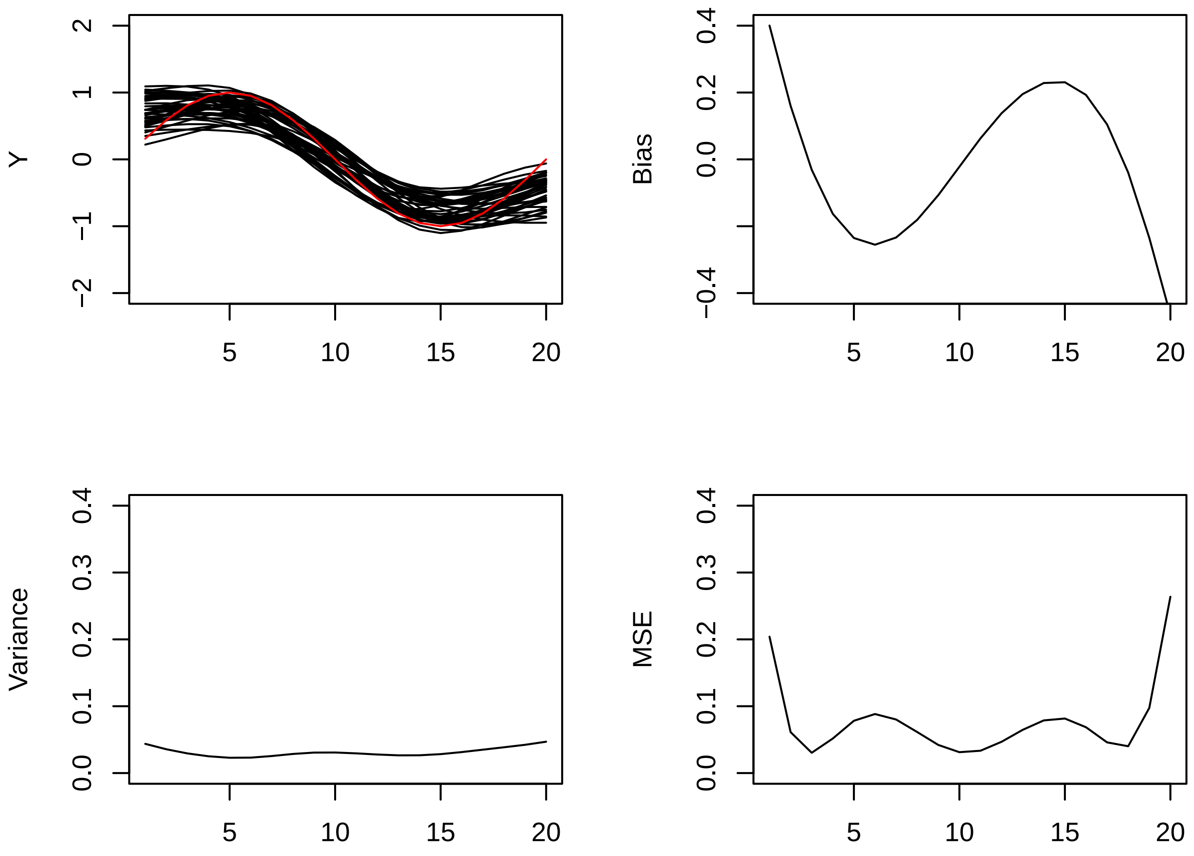
```r
# use (B) - kscv1 function
Y <- Y1 + rnorm(20, sd=0.5)
ntrial <- 15
bandw <- seq(from = 1.5, by = 0.1, length = ntrial)
output.ks <- kscv1(X, Y, 20, bandw, ntrial)
cv <- output.ks$cv
gcv <- output.ks$gcv
par(mfrow = c(1, 2))
plot(bandw, cv, type = "n", #ylim = c(0.04, 0.07),
     xlab = "bandwidth", ylab = "CV")
points(bandw, cv, cex = 1.2, pch = 4)
lines(bandw, cv, lwd = 2)
```

```
cvmin <- min(cv)
icvmin <- (1.:ntrial)[cv == cvmin]
bandcv <- bandw[icvmin]
points(bandcv, cvmin, cex = 1.2, pch = 15)
plot(bandw, gcv, type = "n", #ylim = c(0.04, 0.07),
     xlab = "bandwidth", ylab = "GCV")
points(bandw, gcv, cex = 1.2, pch = 4)
lines(bandw, gcv, lwd = 2)
gcvmin <- min(gcv)
igcvmin <- (1.:ntrial)[gcv == gcvmin]
bandgcv <- bandw[igcvmin]
points(bandgcv, gcvmin, cex = 1.2, pch = 15)
```



CV, GCV 모두 `bandwidth=2.6`일 때 적당하다고 한다. 이를 대입하여 그림을 그리면 아래와 같다.

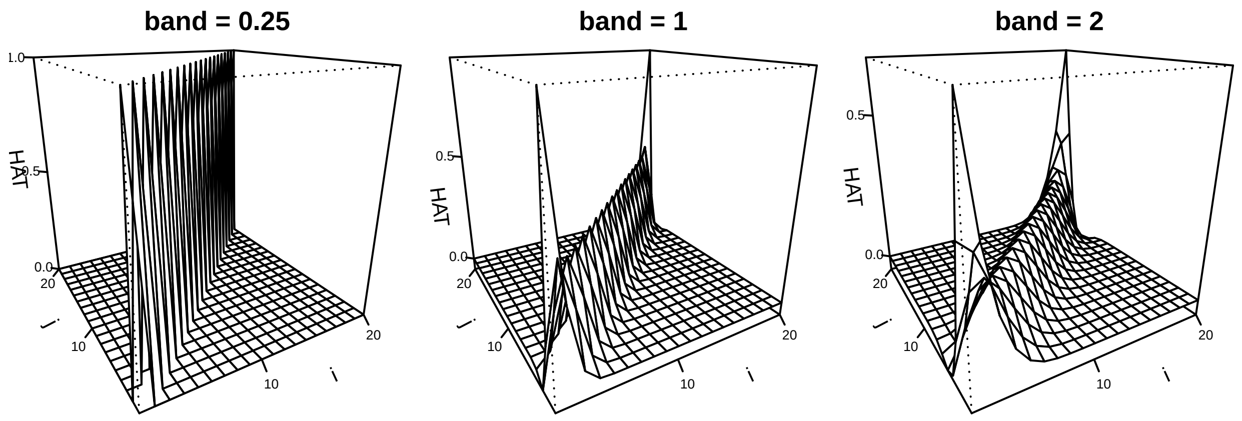
```
bias_var_trade_off(X, Y1, 2.6)
```



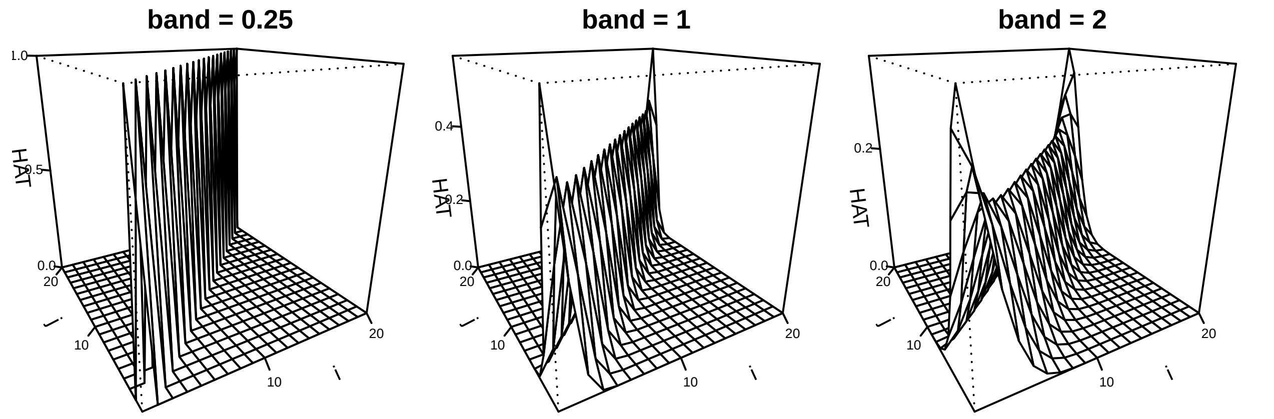
그래프를 보았을 때, 위의 bandwidth가 optimal하다고 할 수 있다.

2. Weight diagrams

```
# use (C) - ll1 function
ll_weight_diagram <- function(band){
  hat_matrix <- matrix(nrow = 20, ncol = 20)
  for (i in 1:20){
    for (j in 1:20){
      hat_matrix[i,j] <- ll1(i, 1:20, diag(20)[,j], band)
    }
  }
  persp(1:20, 1:20, hat_matrix, xlab = 'i', ylab='j', zlab='HAT',
        lab = c(3, 3, 3), theta = -30, phi = 20,
        ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6,
        main=paste('band =', band))
  hat_matrix
}
NW_weight_diagram <- function(band){
  bwplus <- band/0.3708159
  hat_matrix <- matrix(nrow = 20, ncol = 20)
  for (i in 1:20){
    hat_matrix[,i] <- ksmooth(1:20, diag(20)[,i], kernel = "normal",
                              bandwidth = bwplus, x.points = 1:20)$y
  }
  persp(1:20, 1:20, hat_matrix, xlab = 'i', ylab='j', zlab='HAT',
        lab = c(3, 3, 3), theta = -30, phi = 20,
        ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6,
        main=paste('band =', band))
}

par(mfrow=c(1,3),mar=c(0,0,1.2,0))
# local linear regression
hat1 <- ll_weight_diagram(0.25)
hat2 <- ll_weight_diagram(1)
hat3 <- ll_weight_diagram(2)
```



```
# Nadaraya-Watson estimator
NW_weight_diagram(0.25)
NW_weight_diagram(1)
NW_weight_diagram(2)
```



```
# verification of eq(3.130)
apply(hat1, 1, sum)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
apply(hat2, 1, sum)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
apply(hat3, 1, sum)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

equivalent kernels는 symmetric 하지 않다. 또한 kernel의 bandwidth와 같은 bandwidth를 Nadaraya-Watson estimator에 사용했을 때, local linear regression이 더 빠르게 감소한다.

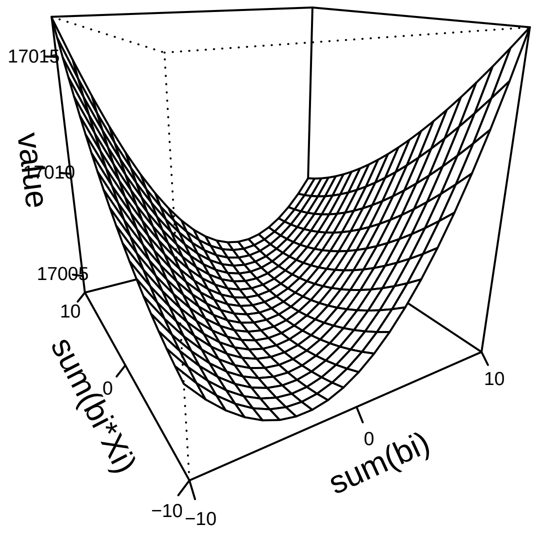
3. Boundary Conditions

(a) Cubic spline function

```
CS_boundary <- function(X, Y, boundary=c(0,0)){
  n <- length(X)
  R <- sapply(X, function(x) abs(x-X)^3/12)
  Q <- matrix(c(X^0, X), nrow=2, byrow=T)
  A <- matrix(0, nrow=n+2, ncol=n+2)
  A[1:n, 1:n] <- R
  A[1:n, (n+1):(n+2)] <- t(Q)
  A[(n+1):(n+2), 1:n] <- Q
  B <- c(Y, boundary)
  sol <- solve(A, B)
  b <- sol[1:n]
  a <- sol[(n+1):(n+2)]
  list(a=a, b=b)
}
```

(b) minimization

```
eq3.171 <- function(X, Y, boundary=c(0,0)){
  b <- CS_boundary(X, Y, boundary)$b
  rng <- range(X)
  f <- function(x) sapply(x, function(x) sum(b*abs(x-X))/2)^2
  integrate(f, rng[1], rng[2], subdivisions=200)$value
}

TEST31 <- read.csv('TEST31.CSV', header=F)
X <- TEST31[,1]
Y <- TEST31[,2]

xx <- seq(-10, 10, by=1)
Z <- matrix(0, ncol=length(xx), nrow=length(xx))
for (x1 in seq(xx)){
  for (y1 in seq(xx)){
    Z[x1, y1] <- eq3.171(X, Y, c(xx[x1], xx[y1]))
  }
}
par(mfrow=c(1,1))
persp(xx, xx, Z, xlab = 'sum(bi)', ylab='sum(bi*Xi)', zlab='value',
      lab = c(3, 3, 3), theta = -30, phi = 20,
      ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6)
```

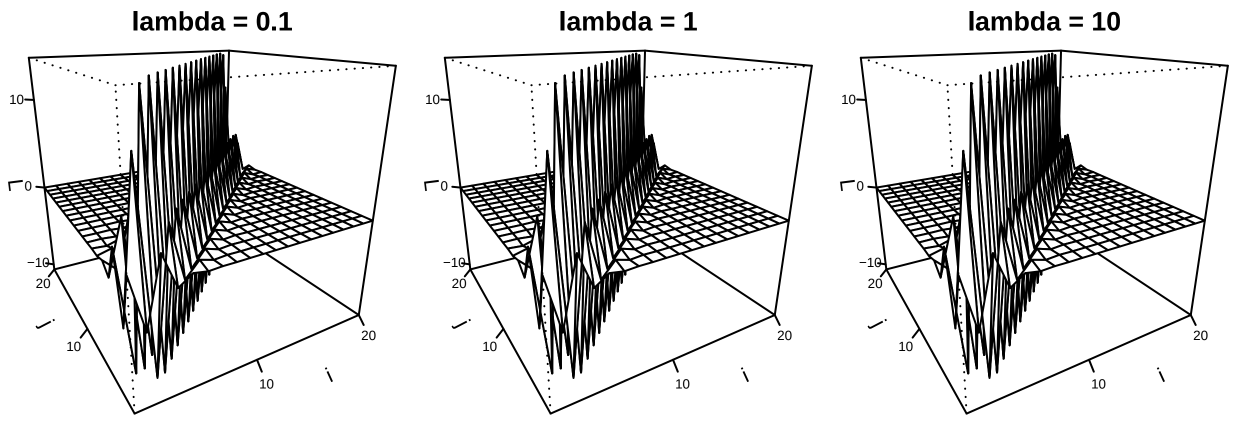


```r
min(Z) == eq3.171(X,Y) # check that natural boundary condition is the minimum
```

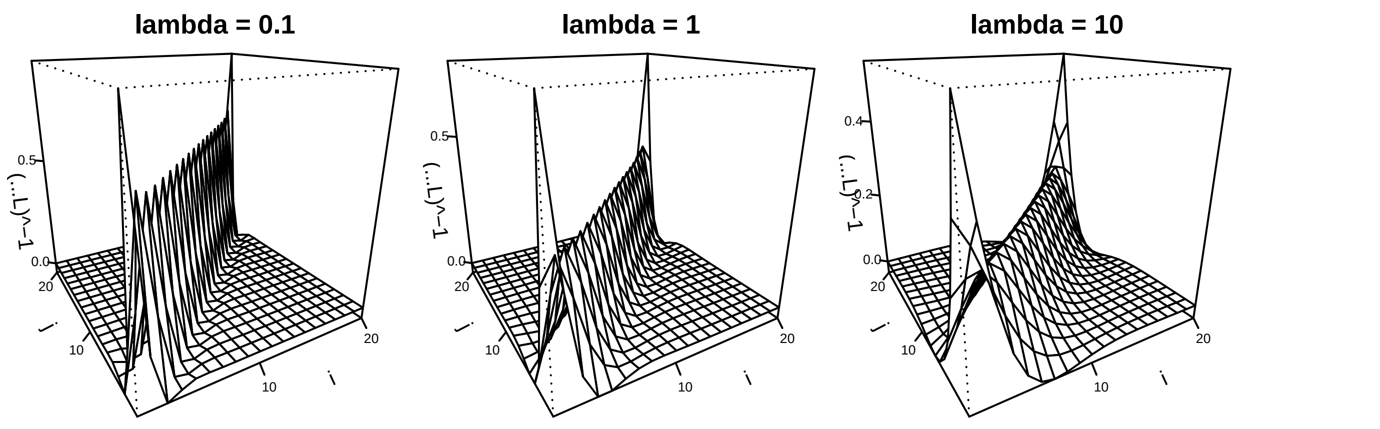
```
[1] TRUE
```

4. $L, (I + \lambda L)^{-1}$.

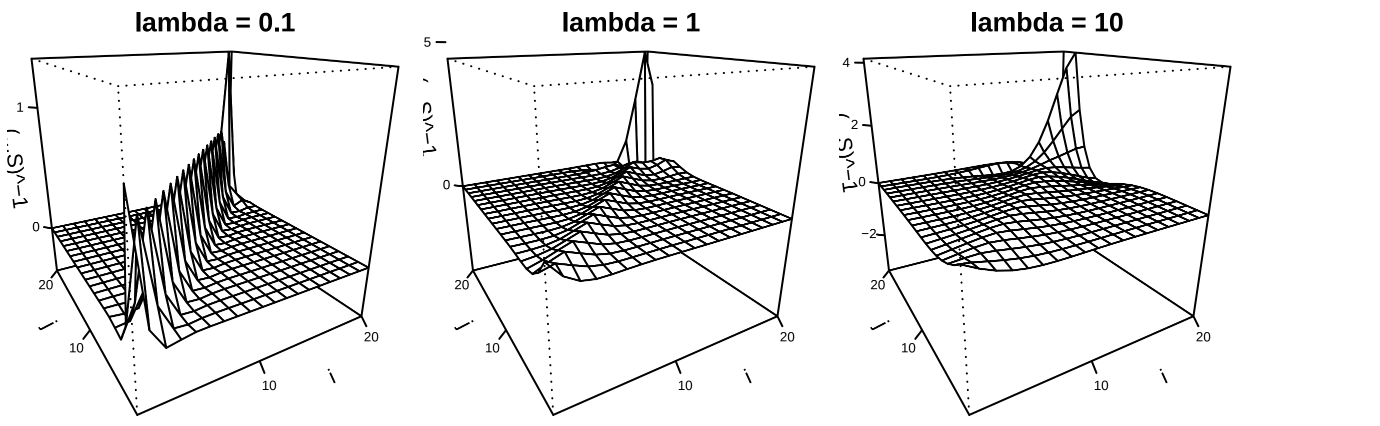
```r
L_value <- function(lambda){
  R <- sapply(1:20, function(x) abs(x-1:20)^3/12)
  Q <- matrix(c(rep(1,20), 1:20), nrow=2, byrow=T)
  A <- matrix(0, nrow=22, ncol=22)
  A[1:20, 1:20] <- R + lambda * diag(20)
  A[1:20, 21:22] <- t(Q)
  A[21:22, 1:20] <- Q
  A_inv <- solve(A)
  Tmat <- A_inv[1:20, 1:20]
  H <- (cbind(R, t(Q)) %*% A_inv)[1:20, 1:20]
  H_inv <- solve(H)
  L <- t(Tmat %*% H_inv) %*% R %*% Tmat %*% H_inv
  L_inv <- solve(diag(20) + lambda * L)
  list(L=L, L_inv=L_inv)
}
S_value <- function(lambda){
  S <- matrix(0, nrow=20, ncol=20)
  S[1, 1:3] <- c(1, -2, 1)
  S[2, 1:4] <- c(-2, 5, -4, 1)
  for (i in 3:18){
    S[i, (i-2):(i+2)] <- c(1, -4, 6, -4, 1)
  }
  S[19, 17:20] <- c(-2, 5, -4, 1)
  S[20, 18:20] <- c(1, -2, 1)
  S_inv <- solve(diag(20) + lambda * S)
  list(S=S, S_inv=S_inv)
}
par(mfrow=c(1,3),mar=c(0,0,1.2,0))
# L
for (lambda in c(0.1, 1, 10)){
  LL <- L_value(lambda)
  persp(1:20, 1:20, LL$L, xlab = 'i', ylab='j', zlab='L',
        lab = c(3, 3, 3), theta = -30, phi = 20,
        ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6,
        main=paste0('lambda = ', lambda))
}
```



```
# (I + lambda L)^-1
for (lambda in c(0.1, 1, 10)){
  LL <- L_value(lambda)
  persp(1:20, 1:20, LL$L_inv, xlab = 'i', ylab='j', zlab='(...L)^-1',
        lab = c(3, 3, 3), theta = -30, phi = 20,
        ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6,
        main=paste0('lambda = ', lambda))
}
```



```
# (I + lambda S)^-1
for (lambda in c(0.1, 1, 10)){
  SS <- S_value(lambda)
  persp(1:20, 1:20, SS$S_inv, xlab = 'i', ylab='j', zlab='(...S)^-1',
        lab = c(3, 3, 3), theta = -30, phi = 20,
        ticktype = 'detailed', nticks=3, cex.lab = 1, cex.axis = 0.6,
        main=paste0('lambda = ', lambda))
}
```

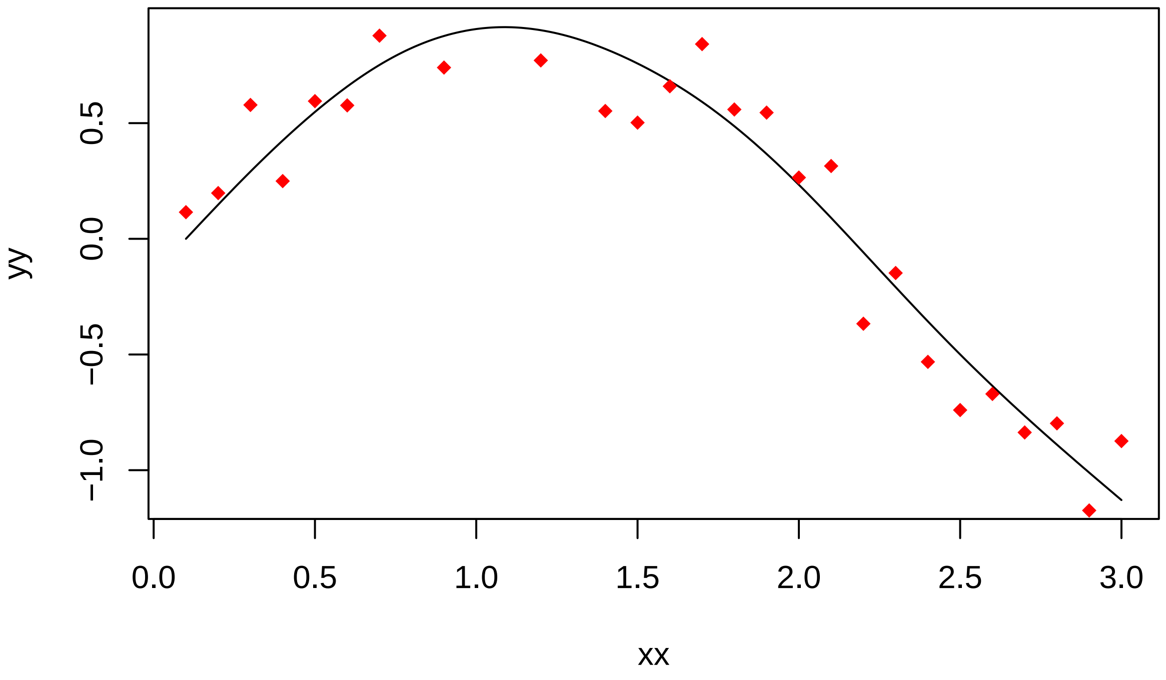


L을 사용한 경우, L[1,1]과 L[20,20]의 값이 같지만, S를 사용한 경우는 그렇지 않다.

5. Smoothing with B-spline

```
Smoothing_B_spline <- function(X, Y, lambda){
  intknot <- X[-c(1, length(X))]
  G <- bs(X, knots=intknot)
  P <- ncol(G)
  rng <- range(X)
  dx <- (rng[2] - rng[1]) / 200
```

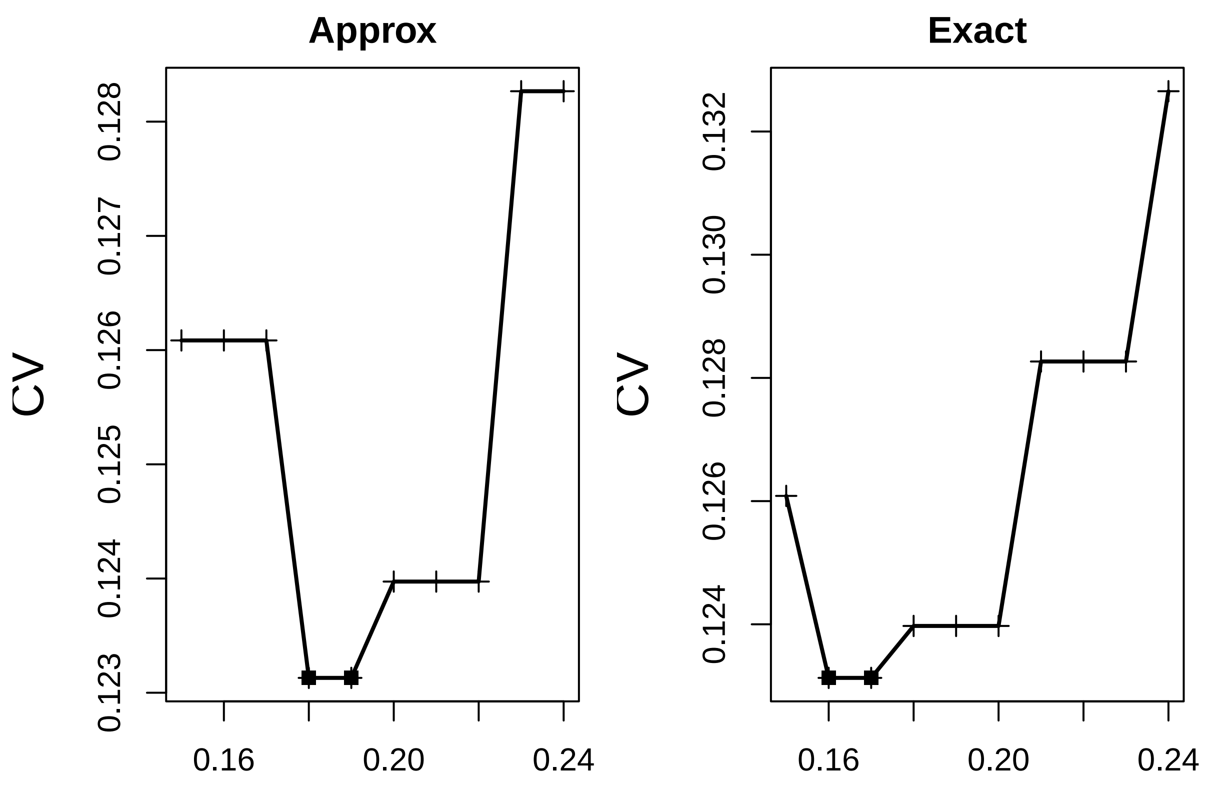
```r
  ddG <- dbs(seq(rng[1], rng[2], length=201), knots=intknot, derivs=2)
  K <- diag(P)
  for (i in 1:P){
    for (j in 1:P){
      K[i, j] <- sum(ddG[,i] * ddG[,j]) * dx
    }
  }
  beta <- solve(t(G) %*% G + lambda * K) %*% t(G) %*% Y
  H <- G %*% solve(t(G) %*% G + lambda * K) %*% t(G)
  list(beta=beta, H=H)
}

SS <- Smoothing_B_spline(X, Y, lambda = 0.1)
beta <- SS$beta
xx <- seq(0.1, 3, by=0.01)
gg <- bs(xx, knots=X[-c(1, length(X))])
yy <- gg %*% beta
plot(xx, yy, type="l")
points(X, Y, col="red", pch=18)
```



6. CV for LOESS

```r
locv2 <- function(xx, yy, nd, span1){
  locv <- function(sp, x1, y1){
    nd <- length(x1)
    s <- 0
    for (i in 1:nd){
      xx1 <- x1[-i]
      yy1 <- y1[-i]
      fit.lo <- loess(yy1 ~ xx1, span = sp, family = "gaussian",
                      degree = 1, surface = "direct")
      mhat <- predict(fit.lo, x1[i])
      s <- s + (y1[i] - mhat)^2
    }
    s/nd
  }
  sapply(span1, locv, x1=xx, y1=yy)
}
# Approximation
nd <- 40
xx <- seq(from = 1, by = 1, length = nd)^1.8
yy <- sin(0.004 * pi * xx) + rnorm(nd, mean = 0, sd = 0.3)
ntrial <- 10
span1 <- seq(from = 0.15, by = 0.01, length = ntrial)
output.lo <- locv1(xx, yy, nd, span1, ntrial)
cv <- output.lo$cv
par(mfrow = c(1, 2), mar = c(3, 4, 2, 1),
    oma=c(0.5,0.5,0.5,0.5), cex.lab=1.5)
plot(span1, cv, type = "n",
     xlab = "span", ylab = "CV", main="Approx")
points(span1, cv, pch = 3)
lines(span1, cv, lwd = 2)
pcvmin <- seq(along = cv)[cv == min(cv)]
spancv <- span1[pcvmin]
cvmin <- cv[pcvmin]
points(spancv, cvmin, cex = 1, pch = 15)

# real definition
cv <- locv2(xx, yy, nd, span1)
```

```r
plot(span1, cv, type = "n",
     xlab = "span", ylab = "CV", main="Exact")
points(span1, cv, pch = 3)
lines(span1, cv, lwd = 2)
pcvmin <- seq(along = cv)[cv == min(cv)]
spancv <- span1[pcvmin]
cvmin <- cv[pcvmin]
points(spancv, cvmin, cex = 1, pch = 15)
```

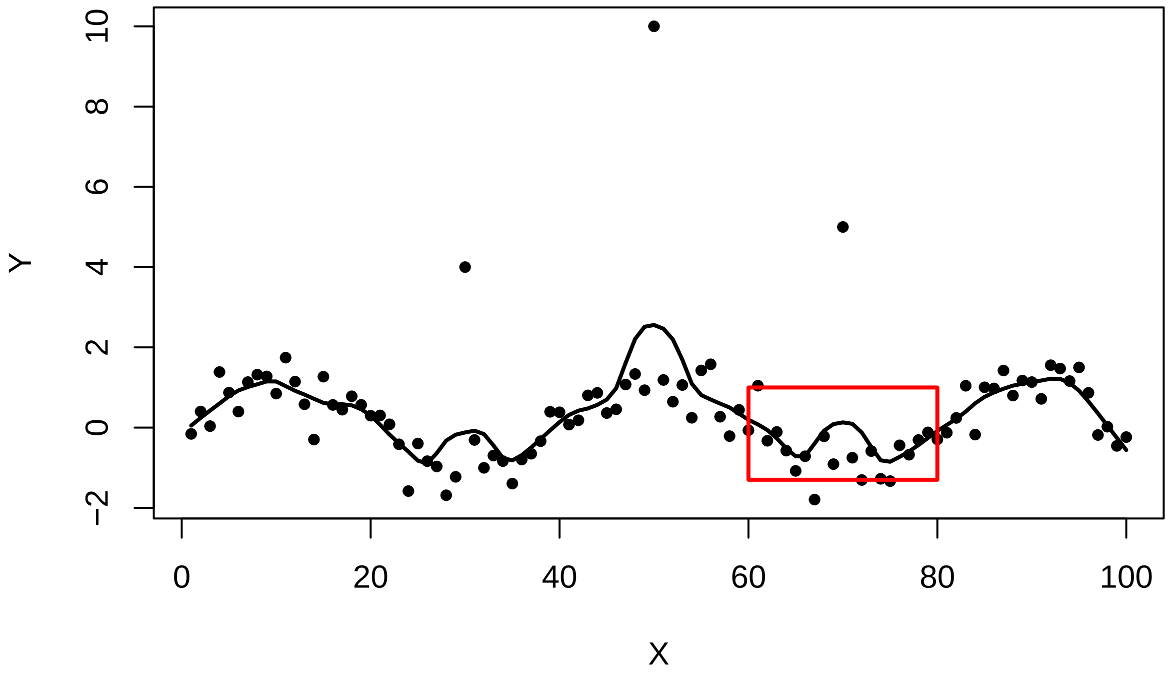


약간의 차이가 있긴 하지만, optimal한 span을 찾는 데에는 크게 차이가 나지 않으므로, approximation을 사용해도 괜찮다고 생각한다.

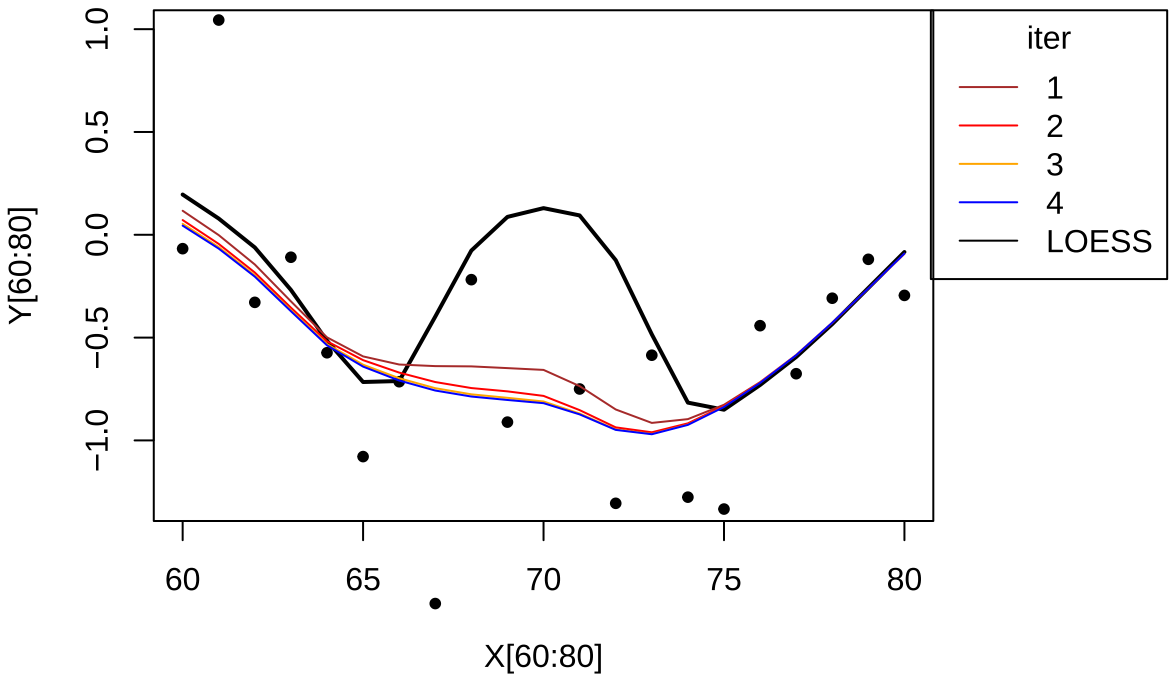
7. LOWESS

```r
B <- function(u) ifelse(abs(u)>1, 0, 15 * (1 - u^2)^2 / 16)
# LOESS
set.seed(1)
X <- 1:100
Y <- sin(0.05 * pi * X) + rnorm(100, sd=0.5)
Y[c(30, 50, 70)] <- c(4, 10, 5)
plot(X, Y, pch=20)
```

```
fit.lo <- loess(Y~ X, span = 0.1, family = "gaussian",
                degree = 1, surface = "direct")
lines(X, fit.lo$fitted, col='black', lwd=2)
rect(60, -1.3, 80, 1, border="red",lwd=2)
```



```
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
plot(X[60:80], Y[60:80], pch=20, ylim=c(-1.3,1))
lines(X[60:80], fit.lo$fitted[60:80], col="black", lwd=2)
R <- fit.lo$residuals
color <- c("brown", "red", "orange", "blue")
for (iter in 1:4){
  shat <- median(abs(R))
  sig <- B(R/6/shat)
  fit.lo <- loess(Y~ X, span = 0.1, weights=sig, family = "gaussian",
                  degree = 1, surface = "direct")
  lines(X[60:80], fit.lo$fitted[60:80], col=color[iter])
  R <- fit.lo$residuals
}
legend("topright", inset=c(-0.3,0), legend=c("1", "2", "3", "4", "LOESS"),
       col=c("brown", "red", "orange","blue", "black"), lty=1,
       title="iter")
```



iter
1
2
3
4
LOESS
Y[60:80]
X[60:80]
1.0
0.5
0.0
−0.5
−1.0
60
65
70
75
80