lab3

Dean Tesler, Gil Shilo

8/6/2022

** Question 1 - Simulation

- 1.1 Implementing Kernel Regression
- 1.1.1

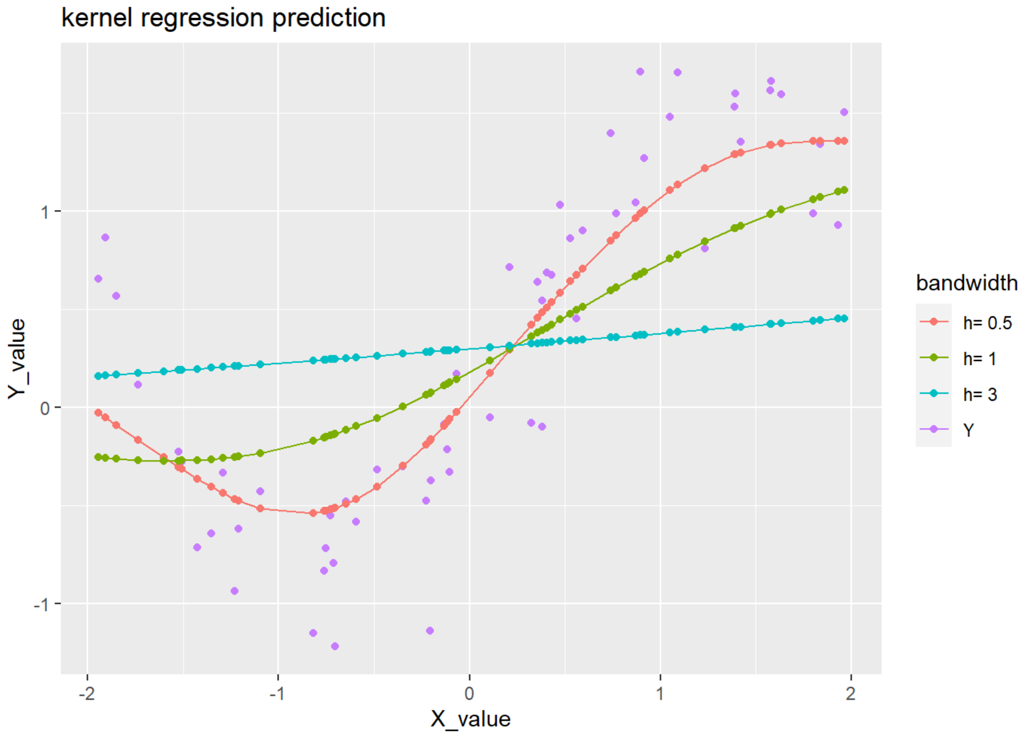
```
sample_f = function( n = 1, use_x= c(), lambda , sigma2 = 0.3){
  if (n == 0){
    return(NA)
    }
  if (length(use_x) > 0) {
    x_by_y <-  cbind(use_x, sapply(use_x, function(x)sin(lambda*x) + 0.3*x^2 + ((x - 0.4)/3)^3 + rnorm(1, 0, sigma2)))
    colnames(x_by_y) <- c("x_value", "y_value")
    return(x_by_y)
    }
  sample_f(n = n,use_x = runif(n = n,min = -2,max = 2),lambda = lambda,sigma2 = sigma2)
}
```

- 1.1.2

```
kernel_regression = function(train_x, train_y, h, test_x){
    kernel_k <- (1/h*sqrt(2*pi)) * exp(-0.5*((train_x - test_x)/h)^2)
    weight <- kernel_k/sum(kernel_k)
    Y <- train_y %*% weight
    return(list(Y = Y,weight = weight))}
```

- 1.1.3

```
n <- 60
lambda <- 1.5
plot_data <- data.frame(sample_f(n=n,lambda=lambda))
plot_data$band_0.5 <-  matrix(sapply(plot_data$x_value, function(x) kernel_regression(train_x = plot_data$x_value, train_
y = plot_data$y_value, h = 0.5, test_x = x)$Y),ncol = 1)
plot_data$band_1 <-  matrix(sapply(plot_data$x_value, function(x) kernel_regression(train_x = plot_data$x_value, train_y
 = plot_data$y_value, h = 1, test_x = x)$Y),ncol = 1)
plot_data$band_3 <-  matrix(sapply(plot_data$x_value, function(x) kernel_regression(train_x = plot_data$x_value, train_y
 = plot_data$y_value, h = 3, test_x = x)$Y),ncol = 1)


ggplot(data = plot_data, aes(x = x_value))  + geom_line(aes(y = band_0.5, color = "h= 0.5")) +  geom_line(aes(y = band_1,
color = "h= 1")) + geom_line(aes(y = band_3, color = "h= 3"))+ labs(col = 'bandwidth')+
  geom_point(aes(y = y_value, color = "Y")) +
  geom_point(aes(y = band_0.5, color = "h= 0.5")) +
  geom_point(aes(y = band_1, color = "h= 1")) +
  geom_point(aes(y = band_3, color = "h= 3"))+ ggtitle("kernel regression prediction") + xlab("X_value") + ylab("Y_value"
)
```



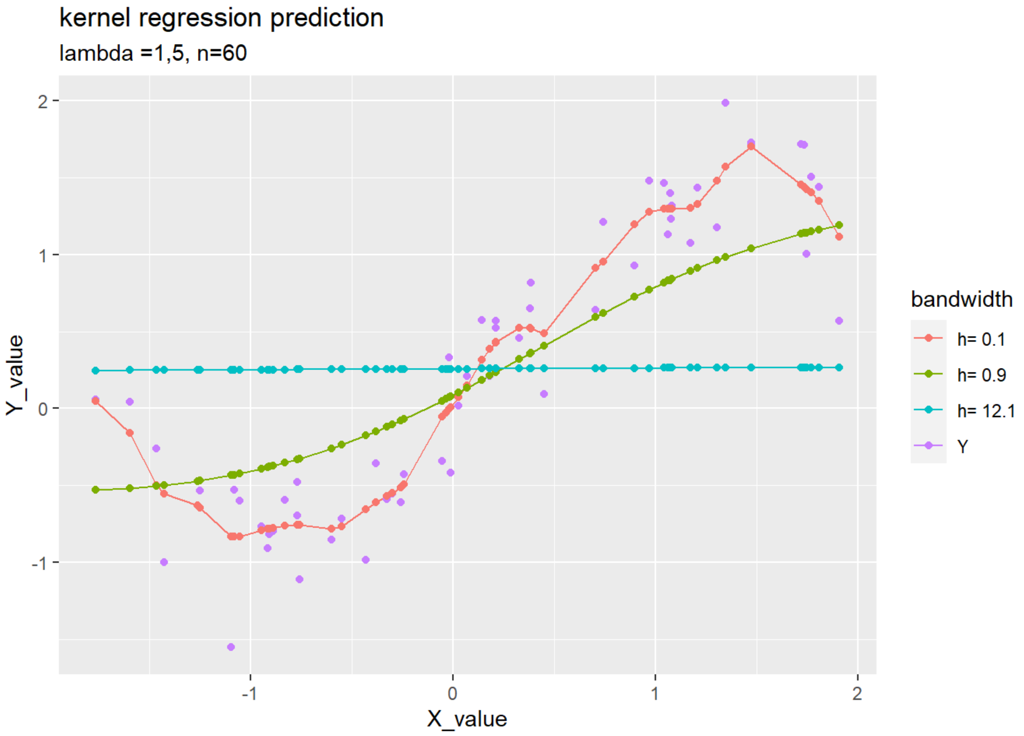
from the graph we can see that

the smallest the 'h' is, it is closer to the real y value and fits it better. on the other hand we see that as 'h' is higher it gets more like a straight line, and that means it dosent have lots of 'noise'.

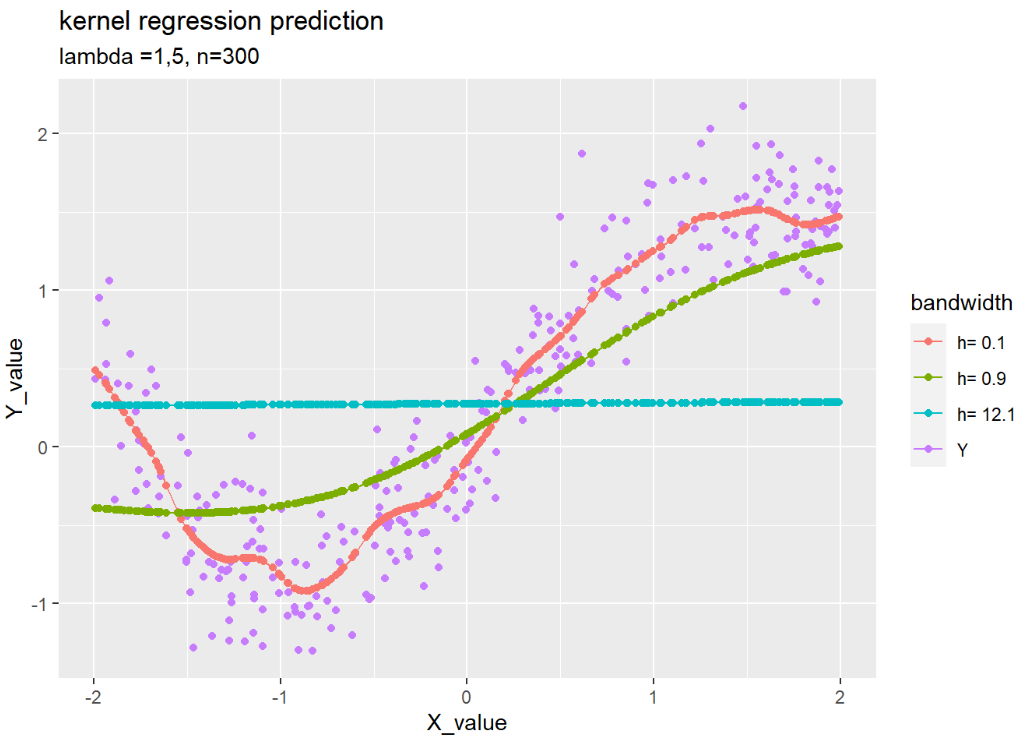
- 1.2 Regression errors for Kernel Regression
- 1.2.1

```
sample_run = function(sim_data){
  for (h in seq(0.1,12.1,0.4)) {
    run <-  matrix(sapply(sim_data$x_value, function(x) kernel_regression(train_x = sim_data$x_value,     train_y = sim_d
ata$y_value, h = h, test_x = x)$Y))
    colnames(run) <- paste0('h_',h)
    sim_data <- cbind(sim_data,run)
    }
  return(sim_data)
}

sample_lambda1.5_n60 <- data.frame(sample_f(n = 60, lambda = 1.5))
sample_lambda1.5_n60_run <- sample_run(sample_lambda1.5_n60)
sample_lambda1.5_n300 <- data.frame(sample_f(n = 300, lambda = 1.5))
sample_lambda1.5_n300_run <- sample_run(sample_lambda1.5_n300)

sample_lambda5_n60 <- data.frame(sample_f(n = 60, lambda = 5))
sample_lambda5_n60_run <- sample_run(sample_lambda5_n60)
sample_lambda5_n300<- data.frame(sample_f(n = 300, lambda = 5))
sample_lambda5_n300_run <- sample_run(sample_lambda5_n300)
```

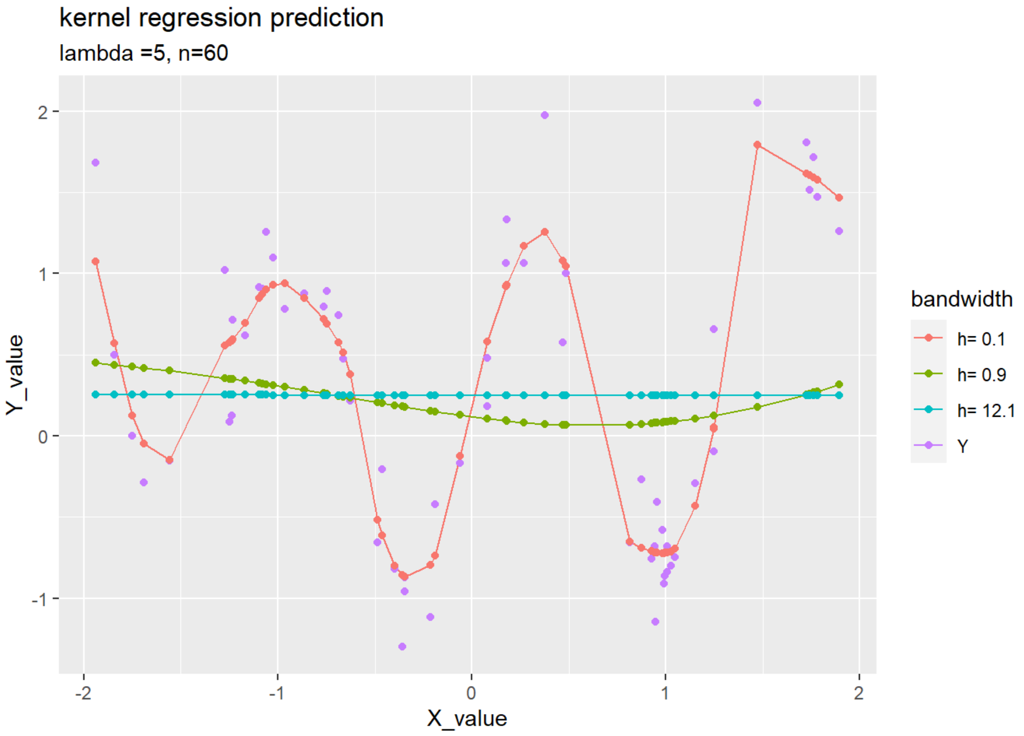
```
ggplot(data = sample_lambda1.5_n60_run, aes(x = x_value))  + geom_line(aes(y =h_0.1 , color = "h= 0.1")) +  geom_line(aes
(y = h_0.9, color = "h= 0.9")) + geom_line(aes(y = h_12.1, color = "h= 12.1"))+ labs(col = 'bandwidth')+
  geom_point(aes(y = y_value, color = "Y")) +
  geom_point(aes(y = h_0.1, color = "h= 0.1")) +
  geom_point(aes(y = h_0.9, color = "h= 0.9")) +
  geom_point(aes(y = h_12.1, color = "h= 12.1"))+ ggtitle("kernel regression prediction", subtitle = 'lambda =1,5, n=60')
+ xlab("X_value") + ylab("Y_value")
```



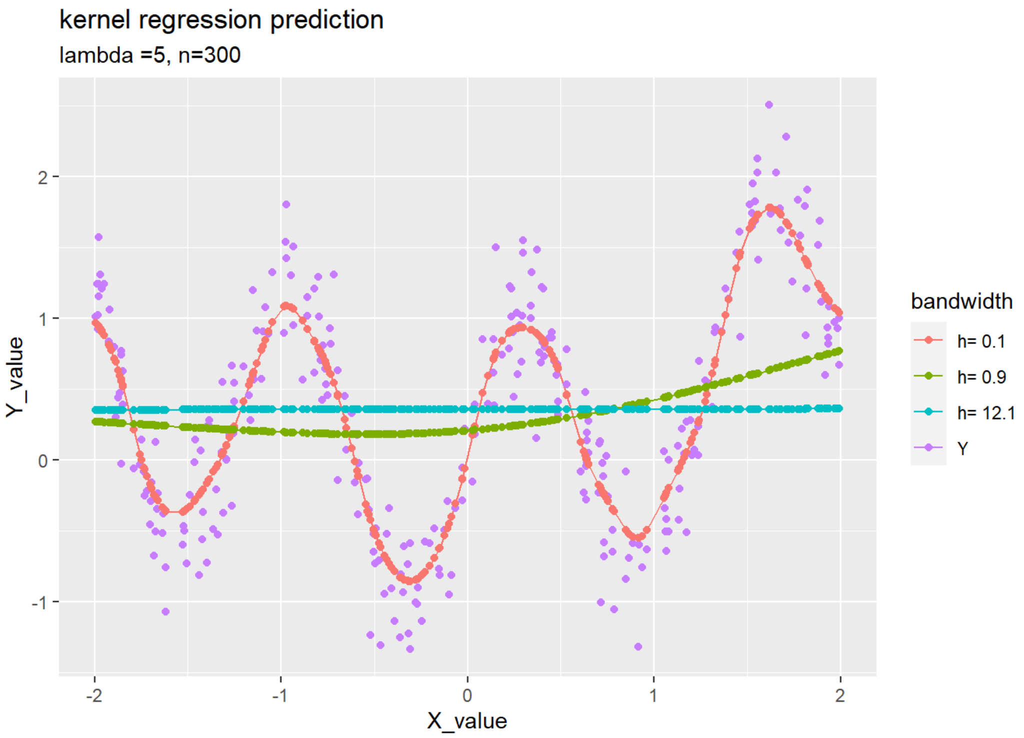

```
ggplot(data = sample_lambda1.5_n300_run, aes(x = x_value))  + geom_line(aes(y =h_0.1 , color = "h= 0.1")) +  geom_line(ae
s(y = h_0.9, color = "h= 0.9")) + geom_line(aes(y = h_12.1, color = "h= 12.1"))+ labs(col = 'bandwidth')+
  geom_point(aes(y = y_value, color = "Y")) +
  geom_point(aes(y = h_0.1, color = "h= 0.1")) +
  geom_point(aes(y = h_0.9, color = "h= 0.9")) +
  geom_point(aes(y = h_12.1, color = "h= 12.1"))+ ggtitle("kernel regression prediction", subtitle = 'lambda =1,5, n=300'
) + xlab("X_value") + ylab("Y_value")
```




```
ggplot(data = sample_lambda5_n60_run, aes(x = x_value))  + geom_line(aes(y =h_0.1 , color = "h= 0.1")) +  geom_line(aes(y
= h_0.9, color = "h= 0.9")) + geom_line(aes(y = h_12.1, color = "h= 12.1"))+ labs(col = 'bandwidth')+
  geom_point(aes(y = y_value, color = "Y")) +
  geom_point(aes(y = h_0.1, color = "h= 0.1")) +
  geom_point(aes(y = h_0.9, color = "h= 0.9")) +
  geom_point(aes(y = h_12.1, color = "h= 12.1"))+ ggtitle("kernel regression prediction", subtitle = 'lambda =5, n=60') +
xlab("X_value") + ylab("Y_value")
```




```
ggplot(data = sample_lambda5_n300_run, aes(x = x_value))  + geom_line(aes(y =h_0.1 , color = "h= 0.1")) +  geom_line(aes
(y = h_0.9, color = "h= 0.9")) + geom_line(aes(y = h_12.1, color = "h= 12.1"))+ labs(col = 'bandwidth')+
  geom_point(aes(y = y_value, color = "Y")) +
  geom_point(aes(y = h_0.1, color = "h= 0.1")) +
  geom_point(aes(y = h_0.9, color = "h= 0.9")) +
  geom_point(aes(y = h_12.1, color = "h= 12.1"))+ ggtitle("kernel regression prediction", subtitle = 'lambda =5, n=300')
 + xlab("X_value") + ylab("Y_value")
```



we can say as we saw beofre, the lower 'h' value is it is closer to the y value. and we can see that as lambda is higher it is more cyclical.

- 1.2.1 kernel regresion
- a - err

```
err_calculate <- function(data){
  err <- c()
  for (col in 3:33){
    err <- c(err,mean((data[,col] - data[,2])^2))}
  err <- data.frame(err)
  err$h <- seq(0.1,12.1,0.4)
  return(err)
}

err_lambda1.5_n60  <- err_calculate(sample_lambda1.5_n60_run)
err_lambda1.5_n300  <- err_calculate(sample_lambda1.5_n300_run)
err_lambda5_n60  <- err_calculate(sample_lambda5_n60_run)
err_lambda5_n300  <- err_calculate(sample_lambda5_n300_run)

type_lambda_n <- rep(c('lambda=1.5, n=60','lambda=5, n=60','lambda=1.5, n=300','lambda=5, n=300'),times = c(31,31,31,31))
err_table <- rbind(err_lambda1.5_n60,err_lambda1.5_n300  ,err_lambda5_n60,err_lambda5_n300 )
err_table <- cbind(type_lambda_n,err_table,rep('err',124))
colnames(err_table) <- c('lambda_n','variable','h','variable_name')
err_table
```

```
##            lambda_n   variable    h variable_name
## 1    lambda=1.5, n=60 0.05862827  0.1           err
## 2    lambda=1.5, n=60 0.11019713  0.5           err
## 3    lambda=1.5, n=60 0.21464437  0.9           err
## 4    lambda=1.5, n=60 0.37312204  1.3           err
## 5    lambda=1.5, n=60 0.50938555  1.7           err
## 6    lambda=1.5, n=60 0.60459845  2.1           err
## 7    lambda=1.5, n=60 0.66853128  2.5           err
## 8    lambda=1.5, n=60 0.71208332  2.9           err
## 9    lambda=1.5, n=60 0.74260715  3.3           err
## 10   lambda=1.5, n=60 0.76464663  3.7           err
## 11   lambda=1.5, n=60 0.78100246  4.1           err
## 12   lambda=1.5, n=60 0.79343834  4.5           err
## 13   lambda=1.5, n=60 0.80309622  4.9           err
## 14   lambda=1.5, n=60 0.81073654  5.3           err
## 15   lambda=1.5, n=60 0.81687925  5.7           err
## 16   lambda=1.5, n=60 0.82188856  6.1           err
## 17   lambda=1.5, n=60 0.82602519  6.5           err
## 18   lambda=1.5, n=60 0.82947948  6.9           err
## 19   lambda=1.5, n=60 0.83239289  7.3           err
## 20   lambda=1.5, n=60 0.83487220  7.7           err
## 21   lambda=1.5, n=60 0.83699920  8.1           err
## 22   lambda=1.5, n=60 0.83883737  8.5           err
## 23   lambda=1.5, n=60 0.84043656  8.9           err
## 24   lambda=1.5, n=60 0.84183635  9.3           err
## 25   lambda=1.5, n=60 0.84306847  9.7           err
## 26   lambda=1.5, n=60 0.84415859 10.1           err
## 27   lambda=1.5, n=60 0.84512769 10.5           err
## 28   lambda=1.5, n=60 0.84599298 10.9           err
## 29   lambda=1.5, n=60 0.84676877 11.3           err
## 30   lambda=1.5, n=60 0.84746695 11.7           err
## 31   lambda=1.5, n=60 0.84809752 12.1           err
## 32     lambda=5, n=60 0.08224677  0.1           err
## 33     lambda=5, n=60 0.13107042  0.5           err
## 34     lambda=5, n=60 0.22061819  0.9           err
## 35     lambda=5, n=60 0.35192544  1.3           err
## 36     lambda=5, n=60 0.48028909  1.7           err
## 37     lambda=5, n=60 0.57849775  2.1           err
## 38     lambda=5, n=60 0.64790427  2.5           err
## 39     lambda=5, n=60 0.69658061  2.9           err
## 40     lambda=5, n=60 0.73129746  3.3           err
## 41     lambda=5, n=60 0.75664541  3.7           err
## 42     lambda=5, n=60 0.77559787  4.1           err
## 43     lambda=5, n=60 0.79008398  4.5           err
## 44     lambda=5, n=60 0.80137721  4.9           err
## 45     lambda=5, n=60 0.81033693  5.3           err
## 46     lambda=5, n=60 0.81755635  5.7           err
## 47     lambda=5, n=60 0.82345392  6.1           err
## 48     lambda=5, n=60 0.82833085  6.5           err
## 49     lambda=5, n=60 0.83240794  6.9           err
## 50     lambda=5, n=60 0.83584983  7.3           err
## 51     lambda=5, n=60 0.83878116  7.7           err
## 52     lambda=5, n=60 0.84129760  8.1           err
## 53     lambda=5, n=60 0.84347354  8.5           err
## 54     lambda=5, n=60 0.84536750  8.9           err
## 55     lambda=5, n=60 0.84702599  9.3           err
## 56     lambda=5, n=60 0.84848635  9.7           err
## 57     lambda=5, n=60 0.84977883 10.1           err
## 58     lambda=5, n=60 0.85092813 10.5           err
## 59     lambda=5, n=60 0.85195459 10.9           err
## 60     lambda=5, n=60 0.85287506 11.3           err
## 61     lambda=5, n=60 0.85370362 11.7           err
## 62     lambda=5, n=60 0.85445208 12.1           err
## 63  lambda=1.5, n=300 0.07185750  0.1           err
## 64  lambda=1.5, n=300 0.59193587  0.5           err
## 65  lambda=1.5, n=300 0.75493792  0.9           err
## 66  lambda=1.5, n=300 0.79221342  1.3           err
## 67  lambda=1.5, n=300 0.80777176  1.7           err
## 68  lambda=1.5, n=300 0.81414782  2.1           err
## 69  lambda=1.5, n=300 0.81694799  2.5           err
## 70  lambda=1.5, n=300 0.81830128  2.9           err
```

```
## 71  lambda=1.5, n=300 0.81901661  3.3           err
## 72  lambda=1.5, n=300 0.81942477  3.7           err
## 73  lambda=1.5, n=300 0.81967299  4.1           err
## 74  lambda=1.5, n=300 0.81983216  4.5           err
## 75  lambda=1.5, n=300 0.81993888  4.9           err
## 76  lambda=1.5, n=300 0.82001318  5.3           err
## 77  lambda=1.5, n=300 0.82006659  5.7           err
## 78  lambda=1.5, n=300 0.82010606  6.1           err
## 79  lambda=1.5, n=300 0.82013594  6.5           err
## 80  lambda=1.5, n=300 0.82015903  6.9           err
## 81  lambda=1.5, n=300 0.82017720  7.3           err
## 82  lambda=1.5, n=300 0.82019175  7.7           err
## 83  lambda=1.5, n=300 0.82020355  8.1           err
## 84  lambda=1.5, n=300 0.82021325  8.5           err
## 85  lambda=1.5, n=300 0.82022132  8.9           err
## 86  lambda=1.5, n=300 0.82022810  9.3           err
## 87  lambda=1.5, n=300 0.82023385  9.7           err
## 88  lambda=1.5, n=300 0.82023877 10.1           err
## 89  lambda=1.5, n=300 0.82024301 10.5           err
## 90  lambda=1.5, n=300 0.82024669 10.9           err
## 91  lambda=1.5, n=300 0.82024991 11.3           err
## 92  lambda=1.5, n=300 0.82025273 11.7           err
## 93  lambda=1.5, n=300 0.82025523 12.1           err
## 94    lambda=5, n=300 0.08946936  0.1           err
## 95    lambda=5, n=300 0.50033491  0.5           err
## 96    lambda=5, n=300 0.58150668  0.9           err
## 97    lambda=5, n=300 0.61355852  1.3           err
## 98    lambda=5, n=300 0.63297850  1.7           err
## 99    lambda=5, n=300 0.64411575  2.1           err
## 100   lambda=5, n=300 0.65076641  2.5           err
## 101   lambda=5, n=300 0.65499477  2.9           err
## 102   lambda=5, n=300 0.65783788  3.3           err
## 103   lambda=5, n=300 0.65983822  3.7           err
## 104   lambda=5, n=300 0.66129783  4.1           err
## 105   lambda=5, n=300 0.66239496  4.5           err
## 106   lambda=5, n=300 0.66324014  4.9           err
## 107   lambda=5, n=300 0.66390484  5.3           err
## 108   lambda=5, n=300 0.66443690  5.7           err
## 109   lambda=5, n=300 0.66486933  6.1           err
## 110   lambda=5, n=300 0.66522548  6.5           err
## 111   lambda=5, n=300 0.66552226  6.9           err
## 112   lambda=5, n=300 0.66577215  7.3           err
## 113   lambda=5, n=300 0.66598450  7.7           err
## 114   lambda=5, n=300 0.66616647  8.1           err
## 115   lambda=5, n=300 0.66632357  8.5           err
## 116   lambda=5, n=300 0.66646013  8.9           err
## 117   lambda=5, n=300 0.66657959  9.3           err
## 118   lambda=5, n=300 0.66668466  9.7           err
## 119   lambda=5, n=300 0.66677758 10.1           err
## 120   lambda=5, n=300 0.66686015 10.5           err
## 121   lambda=5, n=300 0.66693384 10.9           err
## 122   lambda=5, n=300 0.66699989 11.3           err
## 123   lambda=5, n=300 0.66705930 11.7           err
## 124   lambda=5, n=300 0.66711295 12.1           err
```

we used the formulas that we learnt in class.

- b Eop

```
eop_calculate = function(data, n, sigma= 0.3){
  for (h in seq(0.1,12.1,0.4)) {
    tr_weight <-  as.data.frame(sum(diag(as.matrix(sapply(data$x_value, function(x) kernel_regression(train_x = data$x_va
lue, train_y = data$y_value, h = h, test_x = x)$weight)))))
    data <- cbind(data,tr_weight)}
  eop <- c()
  for (col in 3:33) {
    eop <- c(eop, (2*sigma)/n*data[1,col])}
  eop <- data.frame(eop)
  eop$h <- seq(0.1,12.1,0.4)
  return(eop)
}

eop_lambda1.5_n60 <- eop_calculate(sample_lambda1.5_n60, 60)
eop_lambda5_n60  <- eop_calculate(sample_lambda1.5_n300, 60)
eop_lambda1.5_n300  <- eop_calculate(sample_lambda5_n60, 300)
eop_lambda5_n300  <- eop_calculate(sample_lambda5_n300, 300)



eop_table <- rbind(eop_lambda1.5_n60,eop_lambda1.5_n300  ,eop_lambda5_n60,eop_lambda5_n300 )
eop_table <- cbind(type_lambda_n,eop_table,rep('eop',124))
colnames(eop_table) <- c('lambda_n','variable','h','variable_name')
eop_table
```

```
##           lambda_n    variable    h variable_name
## 1    lambda=1.5, n=60 0.144318000  0.1           eop
## 2    lambda=1.5, n=60 0.033310186  0.5           eop
## 3    lambda=1.5, n=60 0.020357166  0.9           eop
## 4    lambda=1.5, n=60 0.015637316  1.3           eop
## 5    lambda=1.5, n=60 0.013443585  1.7           eop
## 6    lambda=1.5, n=60 0.012293709  2.1           eop
## 7    lambda=1.5, n=60 0.011629570  2.5           eop
## 8    lambda=1.5, n=60 0.011214946  2.9           eop
## 9    lambda=1.5, n=60 0.010939824  3.3           eop
## 10   lambda=1.5, n=60 0.010748292  3.7           eop
## 11   lambda=1.5, n=60 0.010609737  4.1           eop
## 12   lambda=1.5, n=60 0.010506327  4.5           eop
## 13   lambda=1.5, n=60 0.010427128  4.9           eop
## 14   lambda=1.5, n=60 0.010365141  5.3           eop
## 15   lambda=1.5, n=60 0.010315723  5.7           eop
## 16   lambda=1.5, n=60 0.010275694  6.1           eop
## 17   lambda=1.5, n=60 0.010242819  6.5           eop
## 18   lambda=1.5, n=60 0.010215490  6.9           eop
## 19   lambda=1.5, n=60 0.010192527  7.3           eop
## 20   lambda=1.5, n=60 0.010173047  7.7           eop
## 21   lambda=1.5, n=60 0.010156381  8.1           eop
## 22   lambda=1.5, n=60 0.010142011  8.5           eop
## 23   lambda=1.5, n=60 0.010129534  8.9           eop
## 24   lambda=1.5, n=60 0.010118632  9.3           eop
## 25   lambda=1.5, n=60 0.010109050  9.7           eop
## 26   lambda=1.5, n=60 0.010100584 10.1           eop
## 27   lambda=1.5, n=60 0.010093067 10.5           eop
## 28   lambda=1.5, n=60 0.010086362 10.9           eop
## 29   lambda=1.5, n=60 0.010080356 11.3           eop
## 30   lambda=1.5, n=60 0.010074956 11.7           eop
## 31   lambda=1.5, n=60 0.010070082 12.1           eop
## 32     lambda=5, n=60 0.028242304  0.1           eop
## 33     lambda=5, n=60 0.006918862  0.5           eop
## 34     lambda=5, n=60 0.004216524  0.9           eop
## 35     lambda=5, n=60 0.003229344  1.3           eop
## 36     lambda=5, n=60 0.002757215  1.7           eop
## 37     lambda=5, n=60 0.002506059  2.1           eop
## 38     lambda=5, n=60 0.002360051  2.5           eop
## 39     lambda=5, n=60 0.002268625  2.9           eop
## 40     lambda=5, n=60 0.002207869  3.3           eop
## 41     lambda=5, n=60 0.002165539  3.7           eop
## 42     lambda=5, n=60 0.002134903  4.1           eop
## 43     lambda=5, n=60 0.002112032  4.5           eop
## 44     lambda=5, n=60 0.002094512  4.9           eop
## 45     lambda=5, n=60 0.002080799  5.3           eop
## 46     lambda=5, n=60 0.002069865  5.7           eop
## 47     lambda=5, n=60 0.002061008  6.1           eop
## 48     lambda=5, n=60 0.002053734  6.5           eop
## 49     lambda=5, n=60 0.002047686  6.9           eop
## 50     lambda=5, n=60 0.002042605  7.3           eop
## 51     lambda=5, n=60 0.002038295  7.7           eop
## 52     lambda=5, n=60 0.002034606  8.1           eop
## 53     lambda=5, n=60 0.002031427  8.5           eop
## 54     lambda=5, n=60 0.002028665  8.9           eop
## 55     lambda=5, n=60 0.002026253  9.3           eop
## 56     lambda=5, n=60 0.002024133  9.7           eop
## 57     lambda=5, n=60 0.002022259 10.1           eop
## 58     lambda=5, n=60 0.002020596 10.5           eop
## 59     lambda=5, n=60 0.002019112 10.9           eop
## 60     lambda=5, n=60 0.002017783 11.3           eop
## 61     lambda=5, n=60 0.002016588 11.7           eop
## 62     lambda=5, n=60 0.002015509 12.1           eop
## 63  lambda=1.5, n=300 0.161613629  0.1           eop
## 64  lambda=1.5, n=300 0.036867969  0.5           eop
## 65  lambda=1.5, n=300 0.022809864  0.9           eop
## 66  lambda=1.5, n=300 0.017330664  1.3           eop
## 67  lambda=1.5, n=300 0.014600434  1.7           eop
## 68  lambda=1.5, n=300 0.013102373  2.1           eop
## 69  lambda=1.5, n=300 0.012216738  2.5           eop
## 70  lambda=1.5, n=300 0.011657392  2.9           eop
```

```
## 71  lambda=1.5, n=300 0.011283999  3.3          eop
## 72  lambda=1.5, n=300 0.011023189  3.7          eop
## 73  lambda=1.5, n=300 0.010834152  4.1          eop
## 74  lambda=1.5, n=300 0.010692899  4.5          eop
## 75  lambda=1.5, n=300 0.010584635  4.9          eop
## 76  lambda=1.5, n=300 0.010499859  5.3          eop
## 77  lambda=1.5, n=300 0.010432249  5.7          eop
## 78  lambda=1.5, n=300 0.010377470  6.1          eop
## 79  lambda=1.5, n=300 0.010332475  6.5          eop
## 80  lambda=1.5, n=300 0.010295066  6.9          eop
## 81  lambda=1.5, n=300 0.010263630  7.3          eop
## 82  lambda=1.5, n=300 0.010236961  7.7          eop
## 83  lambda=1.5, n=300 0.010214143  8.1          eop
## 84  lambda=1.5, n=300 0.010194467  8.5          eop
## 85  lambda=1.5, n=300 0.010177383  8.9          eop
## 86  lambda=1.5, n=300 0.010162455  9.3          eop
## 87  lambda=1.5, n=300 0.010149335  9.7          eop
## 88  lambda=1.5, n=300 0.010137742 10.1          eop
## 89  lambda=1.5, n=300 0.010127448 10.5          eop
## 90  lambda=1.5, n=300 0.010118267 10.9          eop
## 91  lambda=1.5, n=300 0.010110043 11.3          eop
## 92  lambda=1.5, n=300 0.010102648 11.7          eop
## 93  lambda=1.5, n=300 0.010095974 12.1          eop
## 94    lambda=5, n=300 0.032697235  0.1          eop
## 95    lambda=5, n=300 0.007487069  0.5          eop
## 96    lambda=5, n=300 0.004563370  0.9          eop
## 97    lambda=5, n=300 0.003454206  1.3          eop
## 98    lambda=5, n=300 0.002909319  1.7          eop
## 99    lambda=5, n=300 0.002612328  2.1          eop
## 100   lambda=5, n=300 0.002437254  2.5          eop
## 101   lambda=5, n=300 0.002326826  2.9          eop
## 102   lambda=5, n=300 0.002253157  3.3          eop
## 103   lambda=5, n=300 0.002201718  3.7          eop
## 104   lambda=5, n=300 0.002164442  4.1          eop
## 105   lambda=5, n=300 0.002136591  4.5          eop
## 106   lambda=5, n=300 0.002115247  4.9          eop
## 107   lambda=5, n=300 0.002098534  5.3          eop
## 108   lambda=5, n=300 0.002085206  5.7          eop
## 109   lambda=5, n=300 0.002074407  6.1          eop
## 110   lambda=5, n=300 0.002065537  6.5          eop
## 111   lambda=5, n=300 0.002058163  6.9          eop
## 112   lambda=5, n=300 0.002051966  7.3          eop
## 113   lambda=5, n=300 0.002046709  7.7          eop
## 114   lambda=5, n=300 0.002042211  8.1          eop
## 115   lambda=5, n=300 0.002038333  8.5          eop
## 116   lambda=5, n=300 0.002034965  8.9          eop
## 117   lambda=5, n=300 0.002032023  9.3          eop
## 118   lambda=5, n=300 0.002029436  9.7          eop
## 119   lambda=5, n=300 0.002027151 10.1          eop
## 120   lambda=5, n=300 0.002025122 10.5          eop
## 121   lambda=5, n=300 0.002023312 10.9          eop
## 122   lambda=5, n=300 0.002021691 11.3          eop
## 123   lambda=5, n=300 0.002020234 11.7          eop
## 124   lambda=5, n=300 0.002018918 12.1          eop
```

sigma^2 is given to us. and w is the weigthed from the values of the diagonal matrix.

- c - accuracy - 5 fold cross validation

```
sort_func <- function(x, y) {
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}
cross_func <- function(data,h, k = 5){
  accuracy_vec <- c()
  for (i in 1:k) {
        test <- data[sample(length(data$x_value),length(data$x_value)/k),]
        train <- data.frame(x_value = sort_func(test$x_value, data$x_value), y_value = sort_func(test$y_value, data$y_val
ue))
        y_hat <- sapply(test$x_value, function(x) kernel_regression(train_x = train$x_value,train_y = train$y_value,h =
 h,test_x = x)$Y)
  accuracy_vec <- c(accuracy_vec, mean((y_hat - test$y_value)^2))}
  return(mean(accuracy_vec))
}

accuracy_func = function(data){
  accuracy <- c()
  for (h in seq(0.1,12.1,0.4)) {
    accuracy_sample <- cross_func(data, h)
    accuracy <- c(accuracy, accuracy_sample)}
  accuracy <- data.frame(accuracy)
  accuracy$h <- seq(0.1,12.1,0.4)
  return(accuracy)
}
accuracy_lambda1.5_n60 <- accuracy_func(sample_lambda1.5_n60)
accuracy_lambda1.5_n300 <- accuracy_func(sample_lambda1.5_n300)
accuracy_lambda5_n60 <- accuracy_func(sample_lambda5_n60)
accuracy_lambda5_n300  <- accuracy_func(sample_lambda5_n300)
accuracy_table <- rbind(accuracy_lambda1.5_n60,accuracy_lambda1.5_n300,accuracy_lambda5_n60,accuracy_lambda5_n300 )
accuracy_table <- cbind(type_lambda_n,accuracy_table,rep('accuracy',124))
colnames(accuracy_table) <- c('lambda_n','variable','h','variable_name')
accuracy_table
```

```
##             lambda_n  variable    h variable_name
## 1    lambda=1.5, n=60 0.1572637  0.1      accuracy
## 2    lambda=1.5, n=60 0.2178471  0.5      accuracy
## 3    lambda=1.5, n=60 0.1841231  0.9      accuracy
## 4    lambda=1.5, n=60 0.4356151  1.3      accuracy
## 5    lambda=1.5, n=60 0.4763658  1.7      accuracy
## 6    lambda=1.5, n=60 0.6152363  2.1      accuracy
## 7    lambda=1.5, n=60 0.6744869  2.5      accuracy
## 8    lambda=1.5, n=60 0.6679692  2.9      accuracy
## 9    lambda=1.5, n=60 0.6987636  3.3      accuracy
## 10   lambda=1.5, n=60 0.6739122  3.7      accuracy
## 11   lambda=1.5, n=60 0.7697318  4.1      accuracy
## 12   lambda=1.5, n=60 1.0170868  4.5      accuracy
## 13   lambda=1.5, n=60 0.7711157  4.9      accuracy
## 14   lambda=1.5, n=60 0.8693050  5.3      accuracy
## 15   lambda=1.5, n=60 0.8502850  5.7      accuracy
## 16   lambda=1.5, n=60 0.9092769  6.1      accuracy
## 17   lambda=1.5, n=60 0.8323830  6.5      accuracy
## 18   lambda=1.5, n=60 0.7564630  6.9      accuracy
## 19   lambda=1.5, n=60 0.8693179  7.3      accuracy
## 20   lambda=1.5, n=60 0.8338725  7.7      accuracy
## 21   lambda=1.5, n=60 0.7054842  8.1      accuracy
## 22   lambda=1.5, n=60 0.7774978  8.5      accuracy
## 23   lambda=1.5, n=60 0.7856783  8.9      accuracy
## 24   lambda=1.5, n=60 0.8451477  9.3      accuracy
## 25   lambda=1.5, n=60 0.9766117  9.7      accuracy
## 26   lambda=1.5, n=60 0.8410078 10.1      accuracy
## 27   lambda=1.5, n=60 0.7001720 10.5      accuracy
## 28   lambda=1.5, n=60 0.8340807 10.9      accuracy
## 29   lambda=1.5, n=60 0.7165308 11.3      accuracy
## 30   lambda=1.5, n=60 1.0237684 11.7      accuracy
## 31   lambda=1.5, n=60 0.9220188 12.1      accuracy
## 32     lambda=5, n=60 0.2231475  0.1      accuracy
## 33     lambda=5, n=60 0.2305778  0.5      accuracy
## 34     lambda=5, n=60 0.2824700  0.9      accuracy
## 35     lambda=5, n=60 0.3374443  1.3      accuracy
## 36     lambda=5, n=60 0.4595032  1.7      accuracy
## 37     lambda=5, n=60 0.5390466  2.1      accuracy
## 38     lambda=5, n=60 0.6168466  2.5      accuracy
## 39     lambda=5, n=60 0.7092142  2.9      accuracy
## 40     lambda=5, n=60 0.7520468  3.3      accuracy
## 41     lambda=5, n=60 0.7222653  3.7      accuracy
## 42     lambda=5, n=60 0.6971109  4.1      accuracy
## 43     lambda=5, n=60 0.7353066  4.5      accuracy
## 44     lambda=5, n=60 0.8345960  4.9      accuracy
## 45     lambda=5, n=60 0.7429365  5.3      accuracy
## 46     lambda=5, n=60 0.7390875  5.7      accuracy
## 47     lambda=5, n=60 0.7739134  6.1      accuracy
## 48     lambda=5, n=60 0.8353072  6.5      accuracy
## 49     lambda=5, n=60 0.8392348  6.9      accuracy
## 50     lambda=5, n=60 0.7729405  7.3      accuracy
## 51     lambda=5, n=60 0.8229741  7.7      accuracy
## 52     lambda=5, n=60 0.8261619  8.1      accuracy
## 53     lambda=5, n=60 0.8344161  8.5      accuracy
## 54     lambda=5, n=60 0.9268398  8.9      accuracy
## 55     lambda=5, n=60 0.8672005  9.3      accuracy
## 56     lambda=5, n=60 0.7919014  9.7      accuracy
## 57     lambda=5, n=60 0.9139082 10.1      accuracy
## 58     lambda=5, n=60 0.8786409 10.5      accuracy
## 59     lambda=5, n=60 0.8646662 10.9      accuracy
## 60     lambda=5, n=60 0.8758589 11.3      accuracy
## 61     lambda=5, n=60 0.8787534 11.7      accuracy
## 62     lambda=5, n=60 0.8323666 12.1      accuracy
## 63  lambda=1.5, n=300 1.7258237  0.1      accuracy
## 64  lambda=1.5, n=300 1.6663115  0.5      accuracy
## 65  lambda=1.5, n=300 1.4906725  0.9      accuracy
## 66  lambda=1.5, n=300 1.0520015  1.3      accuracy
## 67  lambda=1.5, n=300 1.0545260  1.7      accuracy
## 68  lambda=1.5, n=300 0.7654812  2.1      accuracy
## 69  lambda=1.5, n=300 0.9941353  2.5      accuracy
## 70  lambda=1.5, n=300 0.8579364  2.9      accuracy
```

```
## 71  lambda=1.5, n=300 0.9051252  3.3      accuracy
## 72  lambda=1.5, n=300 0.7133373  3.7      accuracy
## 73  lambda=1.5, n=300 0.8253217  4.1      accuracy
## 74  lambda=1.5, n=300 0.8007190  4.5      accuracy
## 75  lambda=1.5, n=300 0.7495059  4.9      accuracy
## 76  lambda=1.5, n=300 0.9363602  5.3      accuracy
## 77  lambda=1.5, n=300 0.7620593  5.7      accuracy
## 78  lambda=1.5, n=300 0.8982898  6.1      accuracy
## 79  lambda=1.5, n=300 1.1075357  6.5      accuracy
## 80  lambda=1.5, n=300 0.7917880  6.9      accuracy
## 81  lambda=1.5, n=300 0.9091576  7.3      accuracy
## 82  lambda=1.5, n=300 0.8841317  7.7      accuracy
## 83  lambda=1.5, n=300 0.7946156  8.1      accuracy
## 84  lambda=1.5, n=300 0.8830536  8.5      accuracy
## 85  lambda=1.5, n=300 0.8619301  8.9      accuracy
## 86  lambda=1.5, n=300 0.7312832  9.3      accuracy
## 87  lambda=1.5, n=300 0.7630633  9.7      accuracy
## 88  lambda=1.5, n=300 0.8893450 10.1      accuracy
## 89  lambda=1.5, n=300 0.9336503 10.5      accuracy
## 90  lambda=1.5, n=300 0.6981388 10.9      accuracy
## 91  lambda=1.5, n=300 0.7300429 11.3      accuracy
## 92  lambda=1.5, n=300 0.9618196 11.7      accuracy
## 93  lambda=1.5, n=300 0.8415244 12.1      accuracy
## 94    lambda=5, n=300 1.0418331  0.1      accuracy
## 95    lambda=5, n=300 0.9323402  0.5      accuracy
## 96    lambda=5, n=300 0.8344756  0.9      accuracy
## 97    lambda=5, n=300 0.7285636  1.3      accuracy
## 98    lambda=5, n=300 0.5750707  1.7      accuracy
## 99    lambda=5, n=300 0.7161844  2.1      accuracy
## 100   lambda=5, n=300 0.6124456  2.5      accuracy
## 101   lambda=5, n=300 0.6412174  2.9      accuracy
## 102   lambda=5, n=300 0.6407543  3.3      accuracy
## 103   lambda=5, n=300 0.6653878  3.7      accuracy
## 104   lambda=5, n=300 0.6311576  4.1      accuracy
## 105   lambda=5, n=300 0.6577095  4.5      accuracy
## 106   lambda=5, n=300 0.6488115  4.9      accuracy
## 107   lambda=5, n=300 0.6706192  5.3      accuracy
## 108   lambda=5, n=300 0.6877738  5.7      accuracy
## 109   lambda=5, n=300 0.5993079  6.1      accuracy
## 110   lambda=5, n=300 0.6385288  6.5      accuracy
## 111   lambda=5, n=300 0.6489655  6.9      accuracy
## 112   lambda=5, n=300 0.6789554  7.3      accuracy
## 113   lambda=5, n=300 0.7449235  7.7      accuracy
## 114   lambda=5, n=300 0.6304206  8.1      accuracy
## 115   lambda=5, n=300 0.7416109  8.5      accuracy
## 116   lambda=5, n=300 0.6516315  8.9      accuracy
## 117   lambda=5, n=300 0.6009812  9.3      accuracy
## 118   lambda=5, n=300 0.6783537  9.7      accuracy
## 119   lambda=5, n=300 0.6310215 10.1      accuracy
## 120   lambda=5, n=300 0.6994879 10.5      accuracy
## 121   lambda=5, n=300 0.6736706 10.9      accuracy
## 122   lambda=5, n=300 0.6660729 11.3      accuracy
## 123   lambda=5, n=300 0.6843652 11.7      accuracy
## 124   lambda=5, n=300 0.6395641 12.1      accuracy
```

- d- EPEin estimation

```
epe_in_func <- function(data, lambda){
  epe <- c()
    for (col in 3:33) {
      epe_in <- c()
      for (k in 1:100) {
        new_values <- sample_f(n = length(data[,1]), data[,1], lambda =  lambda)[,2]
        epe_in <- c(epe_in, mean((new_values - data[,col])^2))}
      epe <- c(epe,mean(epe_in))}
  epe <- data.frame(epe)
  epe$h <- seq(0.1,12.1,0.4)
  return(epe)
}
epe_in_lambda1.5_n60 <- epe_in_func(sample_lambda1.5_n60_run, 1.5)
epe_in_lambda1.5_n300 <- epe_in_func(sample_lambda1.5_n300_run, 1.5)
epe_in_lambda5_n60 <- epe_in_func(sample_lambda5_n60_run, 5)
epe_in_lambda5_n300 <- epe_in_func(sample_lambda5_n300_run, 5)
epe_in_table <- rbind(epe_in_lambda1.5_n60,epe_in_lambda1.5_n300,epe_in_lambda5_n60,epe_in_lambda5_n300)
epe_in_table <- cbind(type_lambda_n,epe_in_table,rep('epe_in',124))
colnames(epe_in_table) <- c('lambda_n','variable','h','variable_name')
epe_in_table
```

```
##             lambda_n   variable    h variable_name
## 1    lambda=1.5, n=60 0.10054717  0.1        epe_in
## 2    lambda=1.5, n=60 0.12313764  0.5        epe_in
## 3    lambda=1.5, n=60 0.22660316  0.9        epe_in
## 4    lambda=1.5, n=60 0.38832025  1.3        epe_in
## 5    lambda=1.5, n=60 0.52302386  1.7        epe_in
## 6    lambda=1.5, n=60 0.61660257  2.1        epe_in
## 7    lambda=1.5, n=60 0.70144449  2.5        epe_in
## 8    lambda=1.5, n=60 0.74184168  2.9        epe_in
## 9    lambda=1.5, n=60 0.76728541  3.3        epe_in
## 10   lambda=1.5, n=60 0.79399609  3.7        epe_in
## 11   lambda=1.5, n=60 0.82383814  4.1        epe_in
## 12   lambda=1.5, n=60 0.82950635  4.5        epe_in
## 13   lambda=1.5, n=60 0.83157687  4.9        epe_in
## 14   lambda=1.5, n=60 0.83250435  5.3        epe_in
## 15   lambda=1.5, n=60 0.85204876  5.7        epe_in
## 16   lambda=1.5, n=60 0.84964221  6.1        epe_in
## 17   lambda=1.5, n=60 0.85928762  6.5        epe_in
## 18   lambda=1.5, n=60 0.84857179  6.9        epe_in
## 19   lambda=1.5, n=60 0.87619792  7.3        epe_in
## 20   lambda=1.5, n=60 0.86747435  7.7        epe_in
## 21   lambda=1.5, n=60 0.87444404  8.1        epe_in
## 22   lambda=1.5, n=60 0.87495497  8.5        epe_in
## 23   lambda=1.5, n=60 0.87069126  8.9        epe_in
## 24   lambda=1.5, n=60 0.86720923  9.3        epe_in
## 25   lambda=1.5, n=60 0.87157331  9.7        epe_in
## 26   lambda=1.5, n=60 0.86166507 10.1        epe_in
## 27   lambda=1.5, n=60 0.87114126 10.5        epe_in
## 28   lambda=1.5, n=60 0.88004140 10.9        epe_in
## 29   lambda=1.5, n=60 0.88419831 11.3        epe_in
## 30   lambda=1.5, n=60 0.87306740 11.7        epe_in
## 31   lambda=1.5, n=60 0.87346800 12.1        epe_in
## 32     lambda=5, n=60 0.09448928  0.1        epe_in
## 33     lambda=5, n=60 0.12985091  0.5        epe_in
## 34     lambda=5, n=60 0.21798858  0.9        epe_in
## 35     lambda=5, n=60 0.34848926  1.3        epe_in
## 36     lambda=5, n=60 0.47928766  1.7        epe_in
## 37     lambda=5, n=60 0.57229682  2.1        epe_in
## 38     lambda=5, n=60 0.64319585  2.5        epe_in
## 39     lambda=5, n=60 0.68854902  2.9        epe_in
## 40     lambda=5, n=60 0.72191005  3.3        epe_in
## 41     lambda=5, n=60 0.74747945  3.7        epe_in
## 42     lambda=5, n=60 0.76802568  4.1        epe_in
## 43     lambda=5, n=60 0.78779154  4.5        epe_in
## 44     lambda=5, n=60 0.79688944  4.9        epe_in
## 45     lambda=5, n=60 0.80301925  5.3        epe_in
## 46     lambda=5, n=60 0.81003832  5.7        epe_in
## 47     lambda=5, n=60 0.81305627  6.1        epe_in
## 48     lambda=5, n=60 0.82247920  6.5        epe_in
## 49     lambda=5, n=60 0.82340562  6.9        epe_in
## 50     lambda=5, n=60 0.82815046  7.3        epe_in
## 51     lambda=5, n=60 0.83513092  7.7        epe_in
## 52     lambda=5, n=60 0.83414643  8.1        epe_in
## 53     lambda=5, n=60 0.83760149  8.5        epe_in
## 54     lambda=5, n=60 0.83511691  8.9        epe_in
## 55     lambda=5, n=60 0.83981024  9.3        epe_in
## 56     lambda=5, n=60 0.83981144  9.7        epe_in
## 57     lambda=5, n=60 0.84499432 10.1        epe_in
## 58     lambda=5, n=60 0.84428309 10.5        epe_in
## 59     lambda=5, n=60 0.84252380 10.9        epe_in
## 60     lambda=5, n=60 0.84243934 11.3        epe_in
## 61     lambda=5, n=60 0.83993385 11.7        epe_in
## 62     lambda=5, n=60 0.84774770 12.1        epe_in
## 63  lambda=1.5, n=300 0.12354338  0.1        epe_in
## 64  lambda=1.5, n=300 0.54211205  0.5        epe_in
## 65  lambda=1.5, n=300 0.69288566  0.9        epe_in
## 66  lambda=1.5, n=300 0.73228629  1.3        epe_in
## 67  lambda=1.5, n=300 0.73878672  1.7        epe_in
## 68  lambda=1.5, n=300 0.73950334  2.1        epe_in
## 69  lambda=1.5, n=300 0.74490733  2.5        epe_in
## 70  lambda=1.5, n=300 0.74889421  2.9        epe_in
```

```
## 71  lambda=1.5, n=300 0.75169304  3.3        epe_in
## 72  lambda=1.5, n=300 0.75372772  3.7        epe_in
## 73  lambda=1.5, n=300 0.75167856  4.1        epe_in
## 74  lambda=1.5, n=300 0.74101056  4.5        epe_in
## 75  lambda=1.5, n=300 0.75680122  4.9        epe_in
## 76  lambda=1.5, n=300 0.74975231  5.3        epe_in
## 77  lambda=1.5, n=300 0.75589524  5.7        epe_in
## 78  lambda=1.5, n=300 0.75729558  6.1        epe_in
## 79  lambda=1.5, n=300 0.75993449  6.5        epe_in
## 80  lambda=1.5, n=300 0.73608009  6.9        epe_in
## 81  lambda=1.5, n=300 0.75118768  7.3        epe_in
## 82  lambda=1.5, n=300 0.75295345  7.7        epe_in
## 83  lambda=1.5, n=300 0.75233801  8.1        epe_in
## 84  lambda=1.5, n=300 0.75687819  8.5        epe_in
## 85  lambda=1.5, n=300 0.74554892  8.9        epe_in
## 86  lambda=1.5, n=300 0.74766456  9.3        epe_in
## 87  lambda=1.5, n=300 0.74493743  9.7        epe_in
## 88  lambda=1.5, n=300 0.74928091 10.1        epe_in
## 89  lambda=1.5, n=300 0.75149022 10.5        epe_in
## 90  lambda=1.5, n=300 0.74334352 10.9        epe_in
## 91  lambda=1.5, n=300 0.74455942 11.3        epe_in
## 92  lambda=1.5, n=300 0.74395684 11.7        epe_in
## 93  lambda=1.5, n=300 0.75509917 12.1        epe_in
## 94    lambda=5, n=300 0.10141585  0.1        epe_in
## 95    lambda=5, n=300 0.50353899  0.5        epe_in
## 96    lambda=5, n=300 0.58692800  0.9        epe_in
## 97    lambda=5, n=300 0.61711533  1.3        epe_in
## 98    lambda=5, n=300 0.62931013  1.7        epe_in
## 99    lambda=5, n=300 0.64187486  2.1        epe_in
## 100   lambda=5, n=300 0.65442996  2.5        epe_in
## 101   lambda=5, n=300 0.65766498  2.9        epe_in
## 102   lambda=5, n=300 0.65982759  3.3        epe_in
## 103   lambda=5, n=300 0.66126672  3.7        epe_in
## 104   lambda=5, n=300 0.66691425  4.1        epe_in
## 105   lambda=5, n=300 0.66428291  4.5        epe_in
## 106   lambda=5, n=300 0.66555533  4.9        epe_in
## 107   lambda=5, n=300 0.66623080  5.3        epe_in
## 108   lambda=5, n=300 0.67237643  5.7        epe_in
## 109   lambda=5, n=300 0.66692176  6.1        epe_in
## 110   lambda=5, n=300 0.67116367  6.5        epe_in
## 111   lambda=5, n=300 0.66835640  6.9        epe_in
## 112   lambda=5, n=300 0.67120568  7.3        epe_in
## 113   lambda=5, n=300 0.66291050  7.7        epe_in
## 114   lambda=5, n=300 0.66774912  8.1        epe_in
## 115   lambda=5, n=300 0.66579304  8.5        epe_in
## 116   lambda=5, n=300 0.67117813  8.9        epe_in
## 117   lambda=5, n=300 0.67078879  9.3        epe_in
## 118   lambda=5, n=300 0.66884573  9.7        epe_in
## 119   lambda=5, n=300 0.66866208 10.1        epe_in
## 120   lambda=5, n=300 0.66702207 10.5        epe_in
## 121   lambda=5, n=300 0.67204252 10.9        epe_in
## 122   lambda=5, n=300 0.66164587 11.3        epe_in
## 123   lambda=5, n=300 0.66886013 11.7        epe_in
## 124   lambda=5, n=300 0.66914322 12.1        epe_in
```

- e - expected prediction error

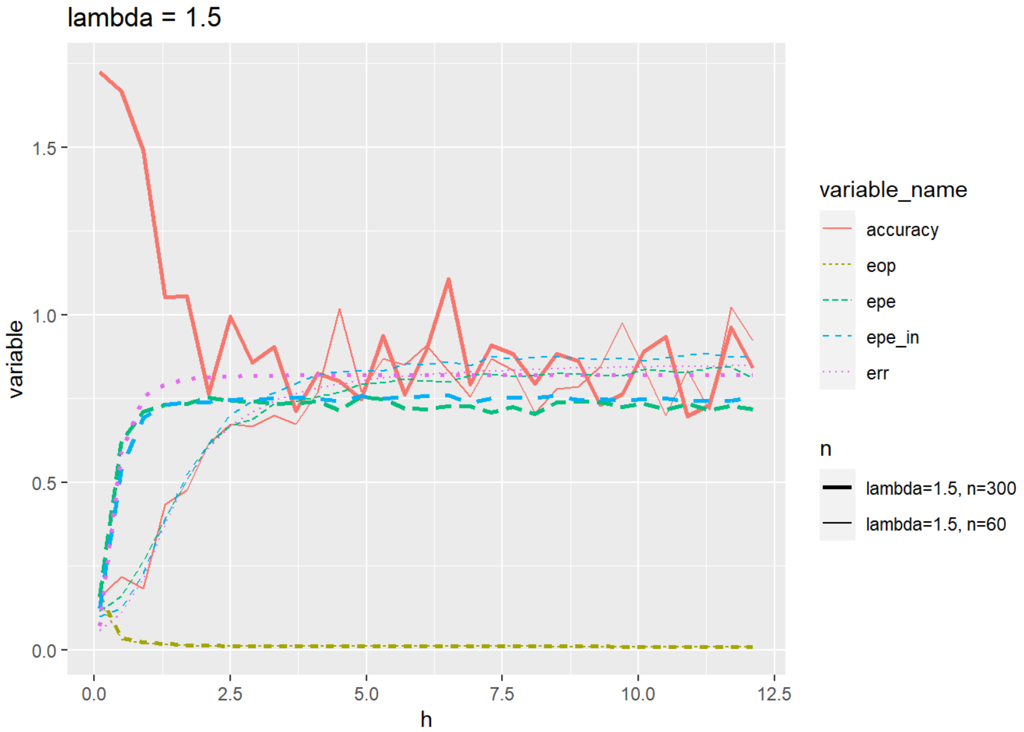
```
epe_func <- function(data, lambda){
  epe <- c()
    for (h in seq(0.1,12.1,0.4)) {
      epes <- c()
      for (k in 1:100) {
        new_data <- sample_f(n = length(data[,1]), lambda =  lambda)
        y_hat <- sapply(new_data[,1], function(x) kernel_regression(train_x = data[,1], train_y = data[,2], h = h, test_x
= x)$Y)
        epes <- c(epes, mean((y_hat - new_data[,2])^2))}
      epe <- c(epe,mean(epes))
      epes <- c()}
  epe <- data.frame(epe)
  epe$h <- seq(0.1,12.1,0.4)
  return(epe)
}

epe_lambda1.5_n60 <- epe_func(sample_lambda1.5_n60, 1.5)
epe_lambda1.5_n300 <- epe_func(sample_lambda1.5_n300, 1.5)
epe_lambda5_n60 <- epe_func(sample_lambda5_n60, 5)
epe_lambda5_n300 <- epe_func(sample_lambda5_n300, 5)
epe_table <- rbind(epe_lambda1.5_n60,epe_lambda1.5_n300,epe_lambda5_n60,epe_lambda5_n300)
epe_table <- cbind(type_lambda_n,epe_table,rep('epe',124))
colnames(epe_table) <- c('lambda_n','variable','h','variable_name')
epe_table
```

```
##            lambda_n   variable    h variable_name
## 1    lambda=1.5, n=60 0.11498739  0.1           epe
## 2    lambda=1.5, n=60 0.16077465  0.5           epe
## 3    lambda=1.5, n=60 0.26381933  0.9           epe
## 4    lambda=1.5, n=60 0.39292681  1.3           epe
## 5    lambda=1.5, n=60 0.50609422  1.7           epe
## 6    lambda=1.5, n=60 0.61327307  2.1           epe
## 7    lambda=1.5, n=60 0.66951327  2.5           epe
## 8    lambda=1.5, n=60 0.68733936  2.9           epe
## 9    lambda=1.5, n=60 0.73338214  3.3           epe
## 10   lambda=1.5, n=60 0.74090874  3.7           epe
## 11   lambda=1.5, n=60 0.75462441  4.1           epe
## 12   lambda=1.5, n=60 0.77021743  4.5           epe
## 13   lambda=1.5, n=60 0.79363574  4.9           epe
## 14   lambda=1.5, n=60 0.79811081  5.3           epe
## 15   lambda=1.5, n=60 0.80893716  5.7           epe
## 16   lambda=1.5, n=60 0.80119653  6.1           epe
## 17   lambda=1.5, n=60 0.80015148  6.5           epe
## 18   lambda=1.5, n=60 0.81961602  6.9           epe
## 19   lambda=1.5, n=60 0.82336855  7.3           epe
## 20   lambda=1.5, n=60 0.81599435  7.7           epe
## 21   lambda=1.5, n=60 0.81706428  8.1           epe
## 22   lambda=1.5, n=60 0.82481432  8.5           epe
## 23   lambda=1.5, n=60 0.82296927  8.9           epe
## 24   lambda=1.5, n=60 0.81885247  9.3           epe
## 25   lambda=1.5, n=60 0.81812176  9.7           epe
## 26   lambda=1.5, n=60 0.83571727 10.1           epe
## 27   lambda=1.5, n=60 0.83262477 10.5           epe
## 28   lambda=1.5, n=60 0.82820552 10.9           epe
## 29   lambda=1.5, n=60 0.84247313 11.3           epe
## 30   lambda=1.5, n=60 0.84265225 11.7           epe
## 31   lambda=1.5, n=60 0.81188201 12.1           epe
## 32     lambda=5, n=60 0.09359202  0.1           epe
## 33     lambda=5, n=60 0.13573431  0.5           epe
## 34     lambda=5, n=60 0.23069372  0.9           epe
## 35     lambda=5, n=60 0.36583708  1.3           epe
## 36     lambda=5, n=60 0.48412960  1.7           epe
## 37     lambda=5, n=60 0.57341106  2.1           epe
## 38     lambda=5, n=60 0.64295061  2.5           epe
## 39     lambda=5, n=60 0.68143223  2.9           epe
## 40     lambda=5, n=60 0.71176284  3.3           epe
## 41     lambda=5, n=60 0.74258557  3.7           epe
## 42     lambda=5, n=60 0.75613466  4.1           epe
## 43     lambda=5, n=60 0.77093982  4.5           epe
## 44     lambda=5, n=60 0.78135836  4.9           epe
## 45     lambda=5, n=60 0.78638446  5.3           epe
## 46     lambda=5, n=60 0.79199973  5.7           epe
## 47     lambda=5, n=60 0.80581914  6.1           epe
## 48     lambda=5, n=60 0.80671790  6.5           epe
## 49     lambda=5, n=60 0.81150835  6.9           epe
## 50     lambda=5, n=60 0.81556967  7.3           epe
## 51     lambda=5, n=60 0.81021300  7.7           epe
## 52     lambda=5, n=60 0.82239437  8.1           epe
## 53     lambda=5, n=60 0.81820402  8.5           epe
## 54     lambda=5, n=60 0.82055473  8.9           epe
## 55     lambda=5, n=60 0.82257509  9.3           epe
## 56     lambda=5, n=60 0.82126185  9.7           epe
## 57     lambda=5, n=60 0.82573955 10.1           epe
## 58     lambda=5, n=60 0.82161323 10.5           epe
## 59     lambda=5, n=60 0.82489254 10.9           epe
## 60     lambda=5, n=60 0.83432710 11.3           epe
## 61     lambda=5, n=60 0.82973468 11.7           epe
## 62     lambda=5, n=60 0.83351640 12.1           epe
## 63  lambda=1.5, n=300 0.15910479  0.1           epe
## 64  lambda=1.5, n=300 0.62181850  0.5           epe
## 65  lambda=1.5, n=300 0.71076758  0.9           epe
## 66  lambda=1.5, n=300 0.73205116  1.3           epe
## 67  lambda=1.5, n=300 0.73356959  1.7           epe
## 68  lambda=1.5, n=300 0.75394435  2.1           epe
## 69  lambda=1.5, n=300 0.74360971  2.5           epe
## 70  lambda=1.5, n=300 0.74067256  2.9           epe
```

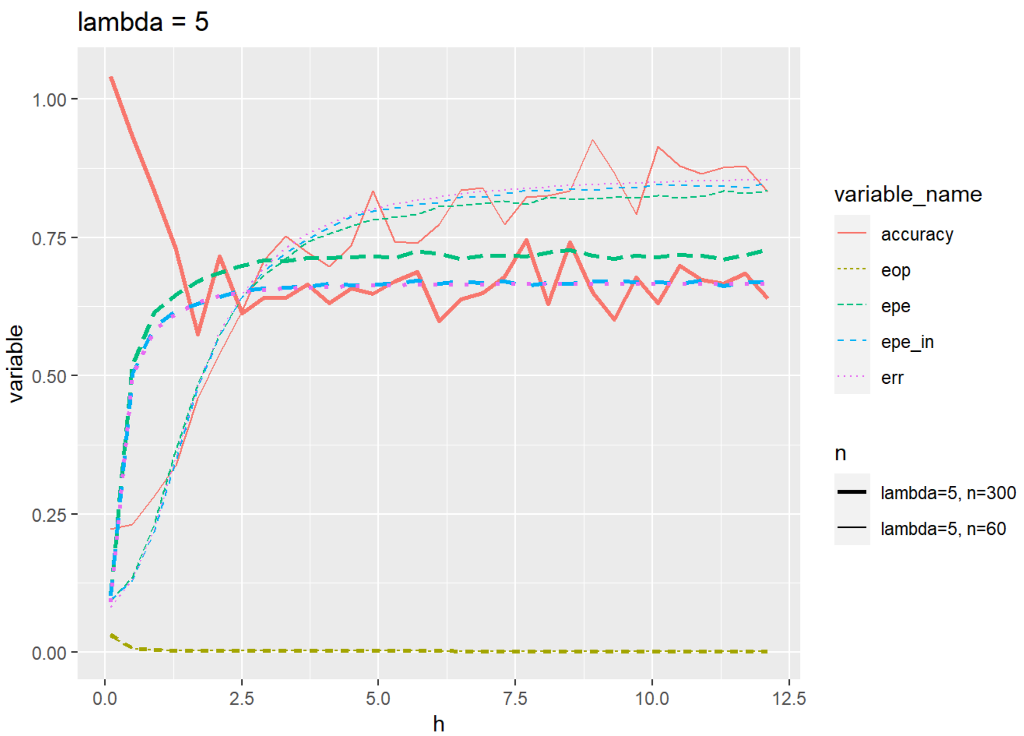
```
## 71  lambda=1.5, n=300 0.73416085  3.3           epe
## 72  lambda=1.5, n=300 0.73706849  3.7           epe
## 73  lambda=1.5, n=300 0.74135431  4.1           epe
## 74  lambda=1.5, n=300 0.71632724  4.5           epe
## 75  lambda=1.5, n=300 0.75586548  4.9           epe
## 76  lambda=1.5, n=300 0.74861242  5.3           epe
## 77  lambda=1.5, n=300 0.72091946  5.7           epe
## 78  lambda=1.5, n=300 0.71830316  6.1           epe
## 79  lambda=1.5, n=300 0.72847993  6.5           epe
## 80  lambda=1.5, n=300 0.72840407  6.9           epe
## 81  lambda=1.5, n=300 0.70922380  7.3           epe
## 82  lambda=1.5, n=300 0.72419170  7.7           epe
## 83  lambda=1.5, n=300 0.70509065  8.1           epe
## 84  lambda=1.5, n=300 0.73866229  8.5           epe
## 85  lambda=1.5, n=300 0.74080268  8.9           epe
## 86  lambda=1.5, n=300 0.73839123  9.3           epe
## 87  lambda=1.5, n=300 0.72618970  9.7           epe
## 88  lambda=1.5, n=300 0.73441033 10.1           epe
## 89  lambda=1.5, n=300 0.71915413 10.5           epe
## 90  lambda=1.5, n=300 0.73401016 10.9           epe
## 91  lambda=1.5, n=300 0.71593760 11.3           epe
## 92  lambda=1.5, n=300 0.72713909 11.7           epe
## 93  lambda=1.5, n=300 0.71908773 12.1           epe
## 94    lambda=5, n=300 0.10214788  0.1           epe
## 95    lambda=5, n=300 0.51922783  0.5           epe
## 96    lambda=5, n=300 0.61476115  0.9           epe
## 97    lambda=5, n=300 0.64714755  1.3           epe
## 98    lambda=5, n=300 0.67073425  1.7           epe
## 99    lambda=5, n=300 0.68645180  2.1           epe
## 100   lambda=5, n=300 0.69910833  2.5           epe
## 101   lambda=5, n=300 0.70819196  2.9           epe
## 102   lambda=5, n=300 0.70789859  3.3           epe
## 103   lambda=5, n=300 0.71269368  3.7           epe
## 104   lambda=5, n=300 0.71296969  4.1           epe
## 105   lambda=5, n=300 0.71401123  4.5           epe
## 106   lambda=5, n=300 0.71545395  4.9           epe
## 107   lambda=5, n=300 0.71293707  5.3           epe
## 108   lambda=5, n=300 0.72598706  5.7           epe
## 109   lambda=5, n=300 0.72007981  6.1           epe
## 110   lambda=5, n=300 0.71200001  6.5           epe
## 111   lambda=5, n=300 0.71694935  6.9           epe
## 112   lambda=5, n=300 0.71680669  7.3           epe
## 113   lambda=5, n=300 0.71578716  7.7           epe
## 114   lambda=5, n=300 0.72295244  8.1           epe
## 115   lambda=5, n=300 0.72642697  8.5           epe
## 116   lambda=5, n=300 0.71654074  8.9           epe
## 117   lambda=5, n=300 0.71135001  9.3           epe
## 118   lambda=5, n=300 0.71711913  9.7           epe
## 119   lambda=5, n=300 0.71399437 10.1           epe
## 120   lambda=5, n=300 0.71928208 10.5           epe
## 121   lambda=5, n=300 0.71721136 10.9           epe
## 122   lambda=5, n=300 0.70946871 11.3           epe
## 123   lambda=5, n=300 0.71813880 11.7           epe
## 124   lambda=5, n=300 0.72665432 12.1           epe
```

- ploting

```
preper_plot_data <- rbind(err_table,eop_table,accuracy_table,epe_in_table,epe_table)

data_lambda_is_1.5 <- preper_plot_data[preper_plot_data$lambda_n %in% c("lambda=1.5, n=60","lambda=1.5, n=300"),]
data_lambda_is_1.5$n <- factor(data_lambda_is_1.5$lambda_n)
ggplot(data_lambda_is_1.5,aes(x = h,y = variable)) + geom_line(aes(color = variable_name,linetype = variable_name, size =
n)) + scale_size_manual(values = c("lambda=1.5, n=60" = 0.5,  "lambda=1.5, n=300" = 1)) + ggtitle("lambda = 1.5")
```




```
data_lambda_is_5 <- preper_plot_data[preper_plot_data$lambda_n %in% c("lambda=5, n=60","lambda=5, n=300"),]
data_lambda_is_5$n <- factor(data_lambda_is_5$lambda_n)
ggplot(data_lambda_is_5,aes(x = h,y = variable)) + geom_line(aes(color = variable_name,linetype = variable_name, size =
 n)) + scale_size_manual(values = c("lambda=5, n=60" = 0.5,  "lambda=5, n=300" = 1)) + ggtitle("lambda = 5")
```



by looking at the graphs we can assume few things: as we have seen before, the lower the lambda is- the higher the accuracy is, and the err is higher. we can also see that for bigger n, the accuracy is higher as we could expect because it lowers the variance. for higher lambda, epein are lower and are more smoother and has less noise. also err are are lower for higher lambda. for both lambdas the eop converges to 0 as h grows but for highr lambda it is faster.

- 1.2.2 - quadratic regression
- a -err

```
quadratic_reg_func <- function(data){
  data$x_value_2 <- data[,1]^2
  return(lm(y_value ~ x_value + x_value_2, data = data))
}


quadratic_reg_lambda1.5_n60 <- mean((quadratic_reg_func(sample_lambda1.5_n60)$fitted.values - sample_lambda1.5_n60[,2])^2
)
quadratic_reg_lambda1.5_n300 <- mean((quadratic_reg_func(sample_lambda1.5_n300)$fitted.values - sample_lambda1.5_n300[,2
])^2)
quadratic_reg_lambda5_n60 <- mean((quadratic_reg_func(sample_lambda5_n60)$fitted.values - sample_lambda5_n60[,2])^2)
quadratic_reg_lambda5_n300 <- mean((quadratic_reg_func(sample_lambda5_n300)$fitted.values - sample_lambda5_n300[,2])^2)



quadratic_err_table <- cbind(quadratic_reg_lambda1.5_n60,quadratic_reg_lambda1.5_n300,quadratic_reg_lambda5_n60,quadratic
_reg_lambda5_n300)
row.names(quadratic_err_table) <- ("err")
knitr::kable(quadratic_err_table)
```

	quadratic_reg_lambda1.5_n60	quadratic_reg_lambda1.5_n300	quadratic_reg_lambda5_n60	quadratic_reg_lambda5_n300
err	0.1864899	0.2233602	0.6943747	0.5305154

- b- eop

```
eop_fun <- function(data){
  mat_x <- as.matrix(cbind(1,data[,1],data[,1]^2))
  mat_w <- mat_x %*% solve(t(mat_x) %*% mat_x) %*% t(mat_x)
  trace_mat_w <- sum(diag(mat_w))
  return(0.6/length(data[,1])*trace_mat_w)}

eop_qua_lambda1.5_n60 <- eop_fun(sample_lambda1.5_n60)
eop_qua_lambda5_n60 <- eop_fun(sample_lambda1.5_n300)
eop_qua_lambda1.5_n300 <- eop_fun(sample_lambda5_n60)
eop_qua_lambda5_n300 <- eop_fun(sample_lambda5_n300)
eop_qua_table <- cbind(eop_qua_lambda1.5_n60,eop_qua_lambda1.5_n300,eop_qua_lambda5_n60,eop_qua_lambda5_n300)
row.names(eop_qua_table) <- ("eop")
knitr::kable(eop_qua_table)
```

	eop_qua_lambda1.5_n60	eop_qua_lambda1.5_n300	eop_qua_lambda5_n60	eop_qua_lambda5_n300
eop	0.03	0.03	0.006	0.006

- c - accuracy- 5 fold cross validation

```
cross_func <- function(data,h, k = 5){
  accuracy_rate <- c()
  for (i in 1:k) {
        test_set <- data[sample(length(data$x_value),length(data$x_value)/k),]
        train_set <- data.frame(x.value = sort_func (test_set$x_value, data$x_value), y_value = sort_func (test_set$y_val
ue,  data$y_value))
        model <- lm(data[,2] ~ data[,1] + I(data[,1]^2))
        beta <- as.vector(model$coefficients)
        design <- as.matrix(cbind(1,test_set[,1], test_set[,1]^2))
        pred <- as.vector(design %*% beta)
  accuracy_rate <- c(accuracy_rate, mean((pred - test_set$y_value)^2))}
  return(mean(accuracy_rate))
}
accuracy_qua_lambda1.5_n60 <- cross_func((sample_lambda1.5_n60))
accuracy_qua_lambda1.5_n300  <- cross_func((sample_lambda1.5_n300))
accuracy_qua_lambda5_n60  <- cross_func((sample_lambda5_n60))
accuracy_qua_lambda5_n300   <- cross_func((sample_lambda5_n300))
accuracy_qua_table <- cbind(accuracy_qua_lambda1.5_n60,accuracy_qua_lambda1.5_n300,accuracy_qua_lambda5_n60,accuracy_qua_
lambda5_n300)
row.names(accuracy_qua_table) <- ("accuracy")
knitr::kable(accuracy_qua_table)
```

	accuracy_qua_lambda1.5_n60	accuracy_qua_lambda1.5_n300	accuracy_qua_lambda5_n60	accuracy_qua_lambda5_n300

	accuracy_qua_lambda1.5_n60	accuracy_qua_lambda1.5_n300	accuracy_qua_lambda5_n60	accuracy_qua_lambda5_n300
accuracy	0.2193751	0.2317732	0.6992124	0.5028954

- d- EPE in

```
epe_in_func  <- function(data, lambda){
  y_hat <- quadratic_reg_func(data)$fitted.values
  epe_in <- c()
      for (k in 1:100) {
        new_values <- sample_f(n = length(data[,1]), data[,1], lambda =  lambda)[,2]
        epe_in <- c(epe_in, mean((new_values - y_hat)^2))}
  return(mean(epe_in))
}

epe_in_qua_lambda1.5_n60 <- epe_in_func (sample_lambda1.5_n60,1.5)
epe_in_qua_lambda1.5_n300 <- epe_in_func (sample_lambda1.5_n300,1.5)
epe_in_qua_lambda5_n60 <- epe_in_func (sample_lambda5_n60,5)
epe_in_qua_lambda5_n300 <- epe_in_func (sample_lambda5_n300,5)
epe_in_qua_table <- cbind(epe_in_qua_lambda1.5_n60,epe_in_qua_lambda1.5_n300,epe_in_qua_lambda5_n60,epe_in_qua_lambda5_n3
00)
row.names(epe_in_qua_table) <- ("epe_in")
knitr::kable(epe_in_qua_table)
```

	epe_in_qua_lambda1.5_n60	epe_in_qua_lambda1.5_n300	epe_in_qua_lambda5_n60	epe_in_qua_lambda5_n300
epe_in	0.1789791	0.2162008	0.6372732	0.5366454

- e - epe

```
epe_func  <- function(data, lambda){
  epe <- c()
      for (k in 1:100) {
        new_values <- data.frame(sample_f(n = length(data[,1]), lambda =  lambda))
        y_hat <- quadratic_reg_func(new_values)$fitted.values
        epe <- c(epe, mean((y_hat - new_values[,2])^2))}
  return(mean(epe))
}
epe_qua_lambda1.5_n60 <- epe_func (sample_lambda1.5_n60,1.5)
eepe_qua_lambda1.5_n300 <- epe_func (sample_lambda1.5_n300,1.5)
epe_qua_lambda5_n60 <- epe_func (sample_lambda5_n60,5)
epe_qua_lambda5_n300 <- epe_func (sample_lambda5_n300,5)
epe_qua_table <- cbind(epe_qua_lambda1.5_n60,eepe_qua_lambda1.5_n300,epe_qua_lambda5_n60,epe_qua_lambda5_n300)
row.names(epe_qua_table) <- ("epe")
knitr::kable(epe_qua_table)
```

	epe_qua_lambda1.5_n60	eepe_qua_lambda1.5_n300	epe_qua_lambda5_n60	epe_qua_lambda5_n300
epe	0.2060396	0.2180181	0.5443906	0.5508114

```
all_table <- data.frame(rbind(quadratic_err_table,eop_qua_table,accuracy_qua_table,epe_in_qua_table,epe_qua_table))
knitr::kable(all_table)
```

	quadratic_reg_lambda1.5_n60	quadratic_reg_lambda1.5_n300	quadratic_reg_lambda5_n60	quadratic_reg_lambda5_n300
err	0.1864899	0.2233602	0.6943747	0.5305154
eop	0.0300000	0.0300000	0.0060000	0.0060000
accuracy	0.2193751	0.2317732	0.6992124	0.5028954
epe_in	0.1789791	0.2162008	0.6372732	0.5366454
epe	0.2060396	0.2180181	0.5443906	0.5508114

we can see that in the quadratic regression for higher lambda the err, eop, accuracy, epein and epe values are also higher. what we find surprising is that for lambda = 1.5 the accuracy is higher for lower n.

if we compare both regression we can say that the quadratic regression changes the values for bigger lambda much bigger, even more then double in some cases. for both regressions the eop are almost consistent. but in kernel regression the err, epein and epe are higher. the accuracy may change depends on the ‘h’ value so we cant determine what is better accuracy wise.

2

A functional magnetic resonance imaging (FMRI) is a type of magnetic resonance imaging. The FMRI are showing how the brain reacts to different levels of oxigen in blood. We want to build models that anticipate voxel reacts to natural images.

2.1 Prediction model

For each voxel, fit a linear model of the features. Because there are more features than responses, you will need to use penalised regression.

2.1.1 Model fitting

```
load("fMRI_data_22.Rdata")
```

create the matrix before filling them

```
set.seed(1)
mspe<-matrix(nrow=3, ncol=2)
colnames(mspe)<-c("ridge","lasso")
rownames(mspe)<-c("y=1","y=2","y=3")

rmspe<-matrix(nrow=3, ncol=2)
colnames(rmspe)<-c("ridge","lasso")
rownames(rmspe)<-c("y=1","y=2","y=3")

se_mat<- matrix(nrow=3, ncol=2)
colnames(se_mat)<-c("ridge","lasso")
rownames(se_mat)<-c("y=1","y=2","y=3")

cv_score<- matrix(nrow=3, ncol=2)
colnames(cv_score)<-c("ridge","lasso")
rownames(cv_score)<-c("y=1","y=2","y=3")

l_mat<- matrix(nrow=3, ncol=2)
colnames(l_mat)<-c("ridge","lasso")
rownames(l_mat)<-c("y=1","y=2","y=3")
```

Fit ridge regression and lasso regression models on training data

```
samp<-sample(1500, 300)
val_x<-feature_train[samp,]

val_y<-train_resp[samp,]

x_train<- feature_train[-samp,]
y_train<-train_resp[-samp,]

pred_m<-as.data.frame(matrix(nrow=250,ncol=6))
colnames(pred_m)<-c("0-1","0-2","0-3","1-1","1-2","1-3")
for(alpha in c(0,1)){
for(y in c(1,2,3)){
  cv.out = cv.glmnet(x_train, y_train[,y], alpha = alpha)

  l_mat[y,alpha+1] = cv.out$lambda.min
  pred<-predict(cv.out,val_x, gamma= "gamma.min")
  cv_score[y,alpha+1 ] <- min(cv.out$cvm)
  mspe[y,alpha+1]<-mean((val_y[,y]-pred)^2)
  rmspe[y,alpha+1]<-sqrt(mean((val_y[,y]-pred)^2))

  se_mat[y,alpha+1]<-sd((val_y[,y]-pred)^2)/sqrt(300)
  pred_m[,paste(alpha,y,sep="-")]<-predict(cv.out,feature_test, gamma= "gamma.min")
}

}
```

```
conf_mspe<- matrix(nrow=3, ncol=2)
conf_rmspe<- matrix(nrow=3, ncol=2)

for(i in c(1:3)){
  for(j in c(1:2)){
    CI_left_mspe<-round(mspe[i,j]-qt(p=0.95,df=300-2)*sqrt(se_mat[i,j]),3)
    CI_right_mspe<-round(mspe[i,j]+qt(p=0.95,df=300-2)*sqrt(se_mat[i,j]),3)

    CI_left_rmspe<-round(rmspe[i,j]-qt(p=0.95,df=300-2)*sqrt(se_mat[i,j]),3)
    CI_right_rmspe<-round(rmspe[i,j]+qt(p=0.95,df=300-2)*sqrt(se_mat[i,j]),3)

    conf_mspe[i,j]<- paste('[', as.character(as.numeric(CI_left_mspe)),' , ',  as.character(as.numeric(CI_right_mspe)),
']')

    conf_rmspe[i,j]<- paste('[', as.character(as.numeric(CI_left_rmspe)),' , ',  as.character(as.numeric(CI_right_rmsp
e)),']')

  }
}
```

confidence interval matrix MSPE

```
knitr::kable(conf_mspe, "simple")
```

[0.449 , 1.351]	[0.433 , 1.307]
[0.534 , 1.478]	[0.543 , 1.507]
[0.531 , 1.491]	[0.531 , 1.491]

confidence interval matrix RMSPE

```
knitr::kable(conf_rmspe, "simple")
```

[0.498 , 1.4]	[0.496 , 1.37]
[0.531 , 1.475]	[0.53 , 1.495]
[0.526 , 1.486]	[0.526 , 1.486]

2.1.2 Presenting results

Present the results for the three responses in a table, detailing for each response (a) the chosen model (ridge or lasso), (b) the chosen lambda, (c) the average cross-validation score (for best model), (d) the estimated MSPE from validation with a confidence interval, and (e) the estimated RMSPE with a confidence interval.

```
res_mat<- as.data.frame(matrix(nrow=7, ncol=3))
colnames(res_mat)=c("Y=1","Y=2","Y=3")
rownames(res_mat)=c("chosen model","lambda","acvs","estimated
MSPE", "CI of MSPE", "RMSPE", "CI of RMSPE")
```

```
for(m in 1:3){
  choosen=1
  if (mspe[m,2]>mspe[m,1]){choosen=2}
  res_mat[1,m]=colnames(mspe)[choosen]
  res_mat[2,m]=l_mat[m,choosen]
  res_mat[3,m]=cv_score[m,choosen]
  res_mat[4,m]=mspe[m,choosen]
  res_mat[5,m]=conf_mspe[m,choosen]
  res_mat[6,m]=rmspe[m,choosen]
  res_mat[7,m]=conf_rmspe[m,choosen]

}
rm(mspe,l_mat,cv_score,conf_mspe,conf_rmspe)
res_mat["estimated\nMSPE",] <- as.numeric(res_mat["estimated\nMSPE",])
```

```
knitr::kable(res_mat)
```

	Y=1	Y=2	Y=3
chosen model	ridge	lasso	ridge
lambda	6.31015553788765	0.0582699481186562	111.769928068285
acvs	0.705983085716746	0.821667429112097	0.992514006382222
estimated			
MSPE	0.900083655323798	1.02492910702545	1.01137693824279
CI of MSPE	[0.449 , 1.351]	[0.543 , 1.507]	[0.531 , 1.491]
RMSPE	0.948727387252944	1.01238782441585	1.00567238116734
CI of RMSPE	[0.498 , 1.4]	[0.53 , 1.495]	[0.526 , 1.486]

- We see a difference in the prediction accuracy for the three responses. We can see that the MSPE for Y=1 is the lowest with lambda of 6.3, therefore this is the best prediction. Further more the CI of Y=1 is the is better than the other two predictions,as we can see the lower and upper bounds are lower than the others.Another parameter we can examine is the average cross-validation score, and the results are the same for this parameter too, Y=1 got the best score. By the results we got Y=1 with lambda 6.31 is the best prediction, but we need to consider that if we will sample the data again (differently) we might get different results.

2.2 Interpreting the results

Linearity of response

```
load("feature_pyramid.Rdata")
load("train_stim_1_250.Rdata")
load("train_stim_251_500.Rdata")
load("train_stim_501_750.Rdata")
load("train_stim_751_1000.Rdata")
load("train_stim_1001_1250.Rdata")
```

```
assign("file-1", train_stim_1_250)
assign("file-2", train_stim_251_500)
assign("file-3", train_stim_501_750)
assign("file-4", train_stim_751_1000)
assign("file-5", train_stim_1001_1250)
```

fitting the chosen model

```
lambda = res_mat$`Y=1`[2]
reg = glmnet(x=feature_train,y=train_resp[,1],lambda = lambda,a = 0)
beta <-as.matrix(predict(reg, type = "coefficients", s = lambda))
```

The most important feature

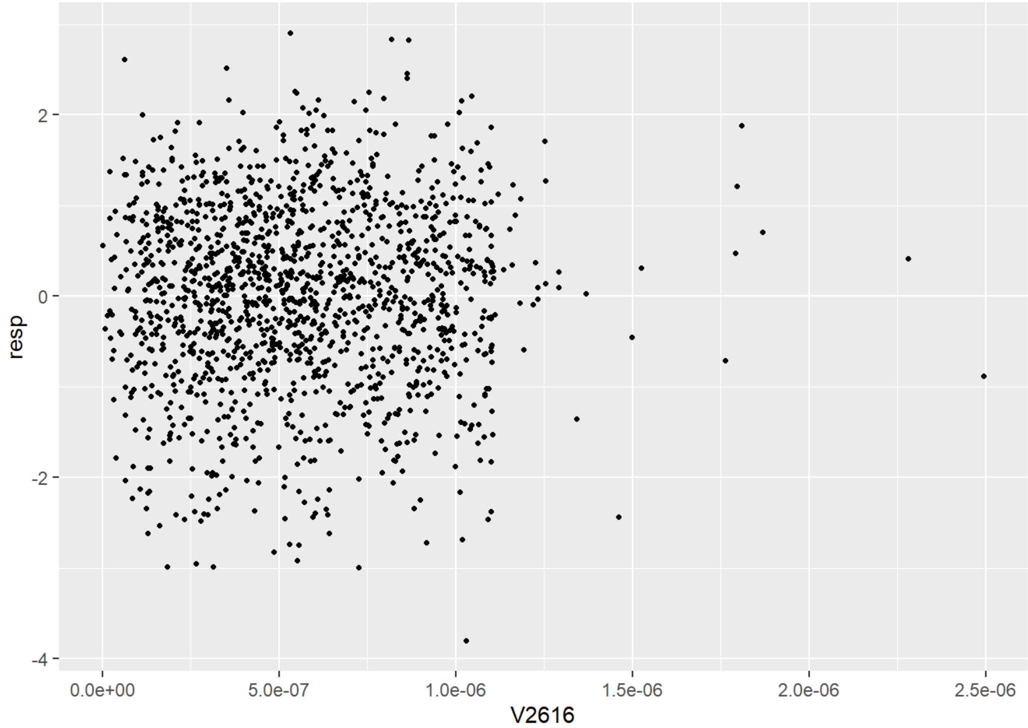
```
knitr::kable(rownames(beta)[beta==max(beta)])
```

x
V2616

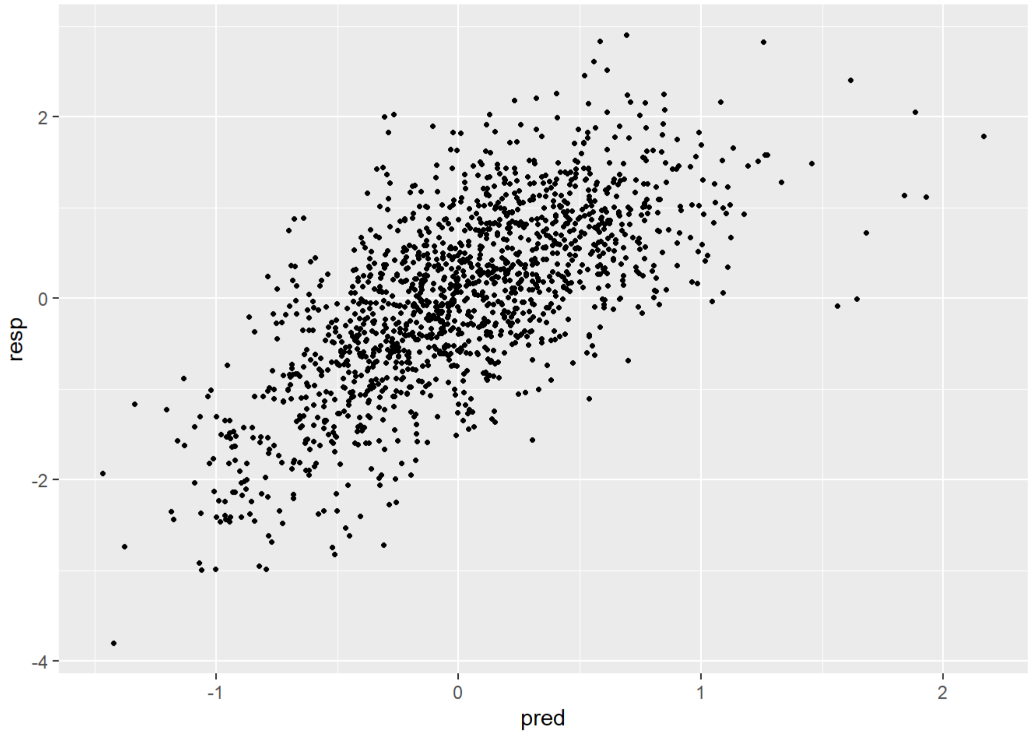
Select the most important feature by weight

```
important <- as.data.frame(feature_train)
important$resp <- train_resp[,1]
important$pred <- predict(reg,newx=feature_train)[,1]
important <- important[c("V2616","resp","pred")]
```

```
ggplot(important,aes(x=V2616,y=resp))+geom_point(shape=20)
```




```
ggplot(important,aes(x=pred,y=resp))+geom_point(shape=20)
```



In the first graph we don’t see a clear linear connection between the most important value and the response values. In the second graph we can see a linear connection between the predicted values and the response values. From this information we can understand that the most important value by weight is not enough to predict the response values and we need more variables in order to find the linear connection. It comes from a high number of dimensions.

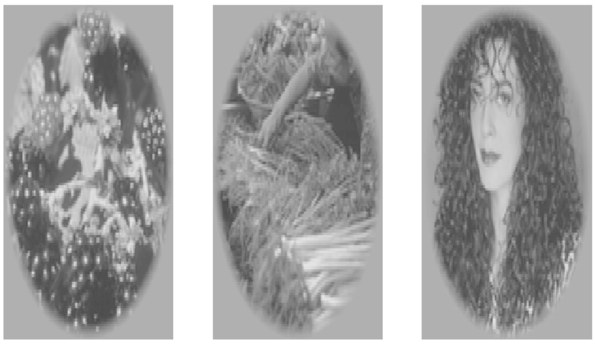
The example domain

```
important$diff <- important$pred-important$resp
important$nrow <- rownames(important)
bottom5 <- top_n(important,5,wt=abs(diff))$nrow
top5 <- top_n(important,-5,wt=abs(diff))$nrow
```

```
train_lst <- list(train_0=train_stim_1_250,train_1=train_stim_251_500,train_2=train_stim_501_750,train_3=train_stim_751_1
000,train_4=train_stim_1001_1250)
best <- matrix(nrow=5,ncol=16384)
j=1
for (i in as.numeric(top5)){
  file_index <- i%/%250+1
  if (file_index<6){
    tmp <- as.data.frame(train_lst[file_index])
    best[j,] <- as.numeric(tmp[rownames(tmp)==i,])
    j=j+1
  }
}

lowest <- matrix(nrow=5,ncol=16384)

j=1
for (i in as.numeric(bottom5)){
  file_index <- i%/%250+1
  if (file_index<6){
    tmp <- as.data.frame(train_lst[file_index])
    lowest[j,] <- as.numeric(tmp[rownames(tmp)==i,])
    j=j+1
  }
}
```

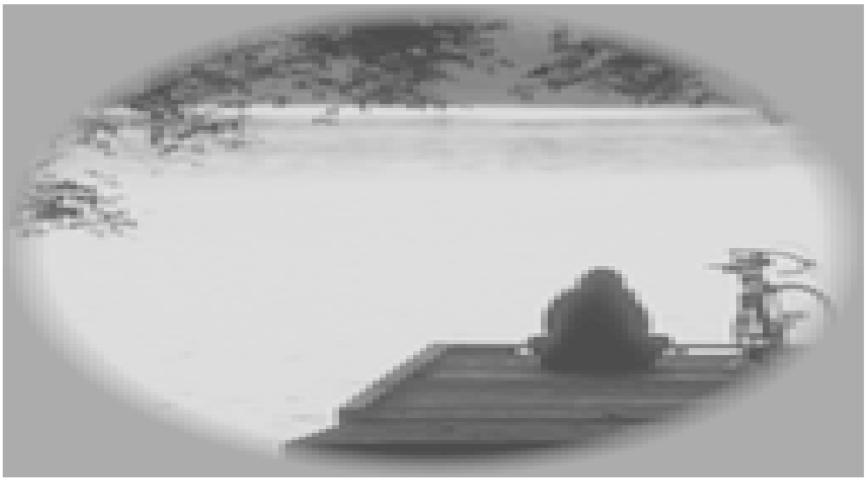
the best by mse

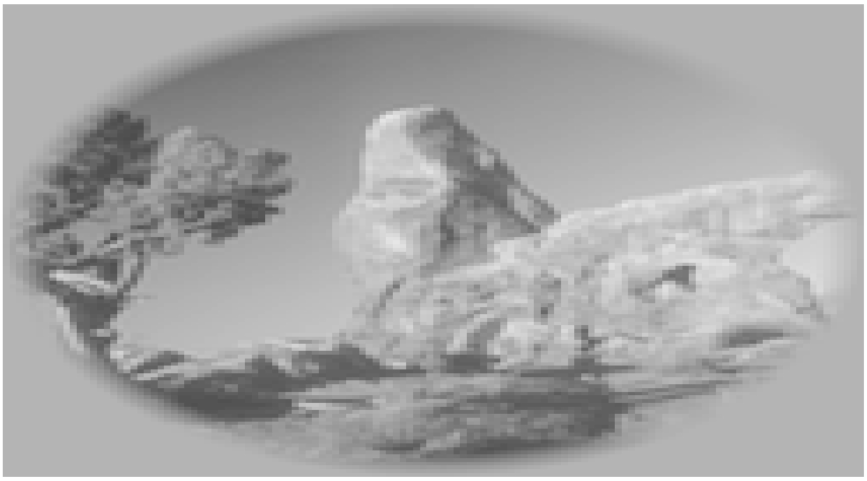
```
par(mar=c(1, 1, 1, 1),mfrow=c(2,5))
for (i in 1:5){image(t(matrix(best[i,],nrow=128)[128:1,]),col=grey.colors(100),axes=F)}
```



the lowest by mse

```
for (i in 1:5){image(t(matrix(lowest[i,],nrow=128)[128:1,]),col=grey.colors(100),axes=F)}
```





After examin the best and the worst predicted pictures we can understand that the pictures that the most elemnts where in the background were the worst predicted and the pictues that the front was the most detailed had the best prediction.

** question 3

```
df <- read_csv("Israel_covid19_newdetections.csv")
```

```
## New names:
## * `` -> ...2
```

```
## Rows: 832 Columns: 2-- Column specification ------------------------------------------------------------
## Delimiter: ","
## chr (2): 2... ,מאומתים חדשים - יומי
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
names(df) <- c('date', 'cases_this_day')
df <- df[-1,]
vec <- c()
for(i in 1:nrow(df)){
  date_1<-str_split(df$date[i], pattern = "-")[[1]]
  date_1<- as.vector(date_1)%>% rev()
  new_vec <- date_1 %>% glue_collapse( sep = "-") %>% as.Date()
  vec <- append(vec,new_vec)
  vec<- as.Date(vec)
  }
df$date <- vec
```

```
df$reg_1 <- ksmooth(x=c(1:831), y= df$cases_this_day,kernel = "normal",bandwidth = 1)$y
df$reg_2 <- ksmooth(x=c(1:831), y= df$cases_this_day,kernel = "normal",bandwidth = 10)$y
df$reg_3 <- ksmooth(x=c(1:831), y= df$cases_this_day,kernel = "normal",bandwidth = 50)$y
head(df)
```

```
## # A tibble: 6 x 5
##   date       cases_this_day reg_1  reg_2 reg_3
##   <date>     <chr>          <dbl>  <dbl> <dbl>
## 1 2020-02-12 0                  0 0.0112  24.8
## 2 2020-02-13 0                  0 0.0181  26.8
## 3 2020-02-14 0                  0 0.0285  28.9
## 4 2020-02-15 0                  0 0.0434  31.3
## 5 2020-02-16 0                  0 0.0641  33.7
## 6 2020-02-17 0                  0 0.0919  36.4
```

```
df_vec <- c(as.Date(rep(df$date, each = 3)))
reg_data <- c(rbind(df$reg_1,df$reg_2,df$reg_3))
order_vec<-c(1,2,3)
order_df = data.frame(date =df_vec)
order_df$order_num <- order_vec
order_df$reg_val <- reg_data
head(order_df)
```

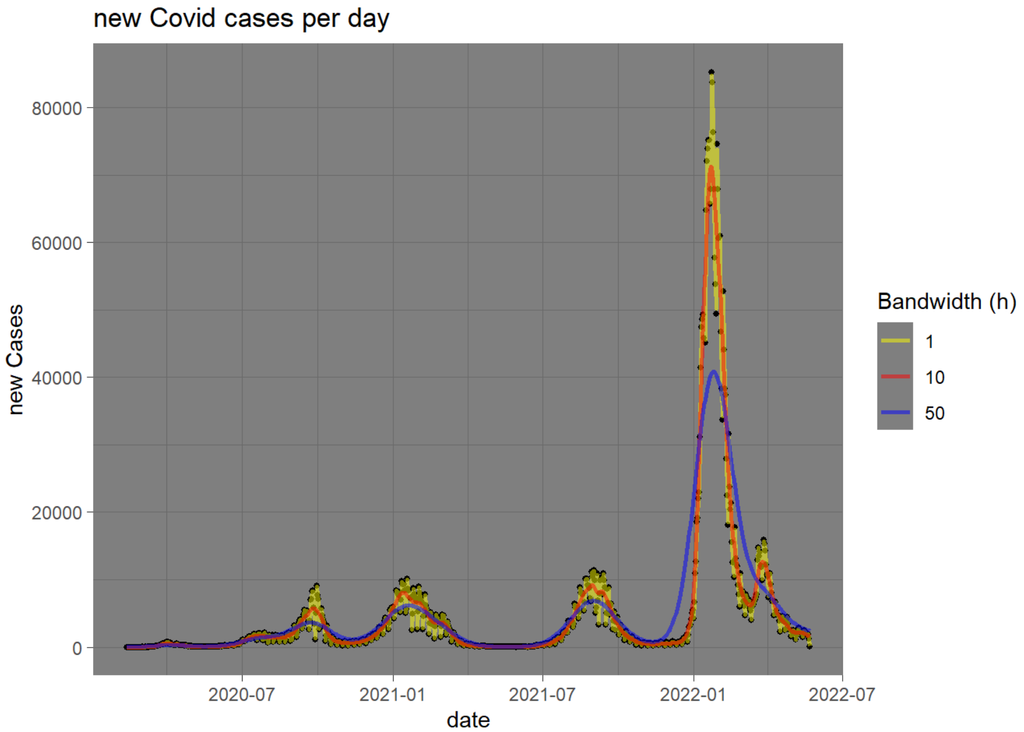
```
##         date order_num     reg_val
## 1 2020-02-12         1  0.00000000
## 2 2020-02-12         2  0.01122268
## 3 2020-02-12         3 24.76990153
## 4 2020-02-13         1  0.00000000
## 5 2020-02-13         2  0.01807014
## 6 2020-02-13         3 26.78088895
```

```
order_df$new_cases <- rep(df$cases_this_day, each = 3)
order_df[order_df$order_num != 1,"new_cases"] <- NA
order_df[order_df$order_num == 1, "color"]<- "blue"
order_df[order_df$order_num == 2, "color"]<- "red"
order_df[order_df$order_num == 3, "color"]<- "yellow"
head(order_df)
```

```
##         date order_num     reg_val new_cases  color
## 1 2020-02-12         1  0.00000000         0   blue
## 2 2020-02-12         2  0.01122268      <NA>    red
## 3 2020-02-12         3 24.76990153      <NA> yellow
## 4 2020-02-13         1  0.00000000         0   blue
## 5 2020-02-13         2  0.01807014      <NA>    red
## 6 2020-02-13         3 26.78088895      <NA> yellow
```

```
order_df$new_cases<-as.numeric(as.character(order_df$new_cases))
ggplot(data = order_df, aes(x = date, y = reg_val, group = order_num, color = color)) +
  labs(col = "Bandwidth (h)")+
  geom_point(aes(y= new_cases), size = 1, color = "black")+
  geom_line(size = 1, alpha = 0.5)+
  ggtitle("new Covid cases per day")+ylab("new Cases") +
  scale_color_manual(values=c("yellow","red","blue"),
                     labels=c("1","10","50") )+ theme_dark()
```

```
## Warning: Removed 1662 rows containing missing values (geom_point).
```



we choose for this question to use the kernel regression that we have seen before in this lab. we can use different 'h'(bandwith) to see the graphs in a smoother way so it wiil be easier to see the ups and downs for the covid data.

as we have said before, we have choose 3 different random 'h' values. the higher the value the smoother is the graph and easy to read, but on the other hand the lower the value the more is it close to the real data. from the graph we can see that there 4 times when the graph rises and we can see that it consistent. at the end of the 3rd quarter of 2020 and 2021. this time of the year in israel it is after the jewish holiday where people gather alot together then it makes sense that the will be a peak of corona. and at the beginning of 2021 and 2022. we can assume that it happens after new years eve where lots of people celebrate new years eve at parties.

```
df$rate1 <- df$reg_1 - lag(df$reg_1)
df$rate2 <- df$reg_2 - lag(df$reg_2)
df$rate3 <- df$reg_3 - lag(df$reg_3)
df$orig_rate <- as.numeric(df$cases_this_day) - lag(as.numeric(df$cases_this_day))

df[1,c("rate1","rate2","rate3","orig_rate")]<- 0

head(df)
```

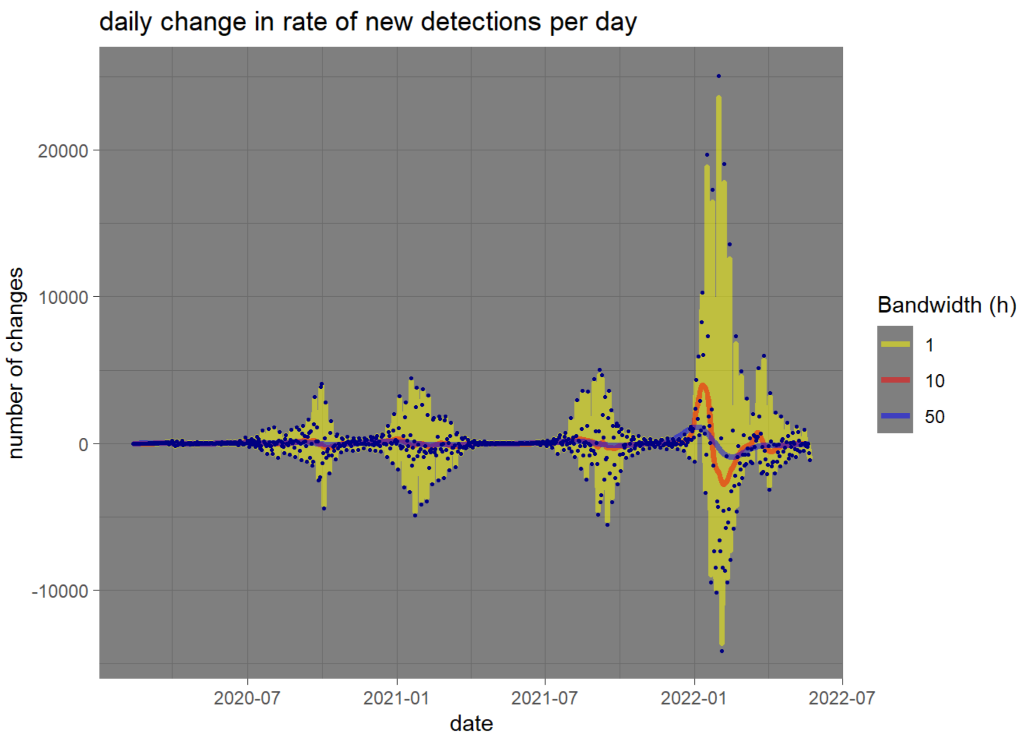
```
## # A tibble: 6 x 9
##   date       cases_this_day reg_1  reg_2 reg_3 rate1   rate2 rate3 orig_rate
##   <date>     <chr>          <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl>     <dbl>
## 1 2020-02-12 0                  0 0.0112  24.8     0 0        0            0
## 2 2020-02-13 0                  0 0.0181  26.8     0 0.00685  2.01         0
## 3 2020-02-14 0                  0 0.0285  28.9     0 0.0104   2.16         0
## 4 2020-02-15 0                  0 0.0434  31.3     0 0.0149   2.32         0
## 5 2020-02-16 0                  0 0.0641  33.7     0 0.0207   2.48         0
## 6 2020-02-17 0                  0 0.0919  36.4     0 0.0279   2.65         0
```

```
vec_df <- rbind(df$rate1,df$rate2,df$rate3)
order_df$rate_type <- c(vec_df)

order_df$orig_rate <- rep(df$orig_rate, each = 3)
order_df[order_df$order_num != 1,"orig_rate"] <- NA

head(order_df)
```

```
##         date order_num     reg_val new_cases  color   rate_type orig_rate
## 1 2020-02-12         1  0.00000000         0   blue 0.000000000         0
## 2 2020-02-12         2  0.01122268        NA    red 0.000000000        NA
## 3 2020-02-12         3 24.76990153        NA yellow 0.000000000        NA
## 4 2020-02-13         1  0.00000000         0   blue 0.000000000         0
## 5 2020-02-13         2  0.01807014        NA    red 0.006847461        NA
## 6 2020-02-13         3 26.78088895        NA yellow 2.010987422        NA
```

```
ggplot(data = order_df, aes(x = date, y= rate_type, group = order_num, color = color))+
  labs(col = "Bandwidth (h)")+
  geom_line(alpha = 0.5, lwd = 1.4)+
  geom_point(aes(y= orig_rate), size = 0.5, color = "navy")+
  ggtitle("daily change in rate of new detections per day")+
  scale_color_manual(values=c("yellow","red","blue"),
                     labels=c("1","10","50") )+ ylab('number of changes')+ theme_dark()
```

```
## Warning: Removed 1662 rows containing missing values (geom_point).
```



this graphs shows us the changes per day of new detections of covid. we can see from this graph at the same eras that we so from the last graph the eras that have lots of covid examinations. again as we said the lower the ‘h’ value is the bigger the swing is and it easier to see the peaks.