Parzen Rosenblatt KDE Bandwidth Analysis on Ill Conditioned Sample

Jacob Richards

2024-10-26

```
setClass("Parzen.smoother",representation(Data = "numeric",
                                          h = "numeric",
                                          input = "numeric",
                                          Probs = "numeric"),
                                          prototype = list(h = NA_real_, input = NA_real_))

setMethod("initialize", "Parzen.smoother",
          function(.Object, Data, h = NA_real_, Probs, input = NA_real_) {

  Data <- as.numeric(Data)
  Data <- na.omit(Data)
  if (length(Data) == 0) { stop("Data set is empty")}

  Probs <- numeric(length(Data))

  if (all(is.na(input))) { input <- Data }

   if (is.na(h)) {
    s <- sd(Data)
    inter_q <- IQR(Data)
    n <- length(Data)
    h <- 1.06 * min(s, inter_q / 1.349) * n^(-1/5)
   }

  if (!is.na(h) && h == 0) { stop(" h cannot be zero ") }

  .Object <- callNextMethod(.Object, Data = Data, h = h, Probs = Probs, input = input)

  return(.Object)
})
```

```r
setGeneric("Parzen.creator", function(object) {standadardGeneric("Parzen.creator")})
```

```
## [1] "Parzen.creator"
```

```r
setMethod("Parzen.creator", signature= "Parzen.smoother", function(object){

  n <- length(object@Data)
  k <- length(object@input)
  Probs <- numeric(k)

  # Initialize Probs with the correct length
  object@Probs <- numeric(k)

  for (j in 1:k) {

        Kernel_sum <- 0

        for (i in 1:n) {
          Kernel_sum <- Kernel_sum + dnorm((object@Data[i] - object@input[j]) / object@h)
        }

  object@Probs[j] <- 1 / (object@h * n) * Kernel_sum
  }

  return(object)
})
```

```r
object <- new("Parzen.smoother", Data=rnorm(10), input=seq(1:10))
Parzen.creator(object)
```

```
## An object of class "Parzen.smoother"
## Slot "Data":
##  [1]  0.2940419  0.1605270  0.8496686  1.3795754  0.3633199  0.4028317
##  [7] -0.5852237 -0.4329306  2.1569999 -0.1649829
##
## Slot "h":
## [1] 0.40732
##
## Slot "input":
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## Slot "Probs":
##  [1] 2.543937e-01 1.235436e-01 1.154011e-02 3.510381e-06 2.583271e-12
##  [6] 4.585009e-21 1.962696e-32 2.026321e-46 5.045521e-63 3.030024e-82
```

```r
setGeneric("show", function(object) {standardGeneric("show")})
```

```
## Creating a new generic function for 'show' in the global environment
```

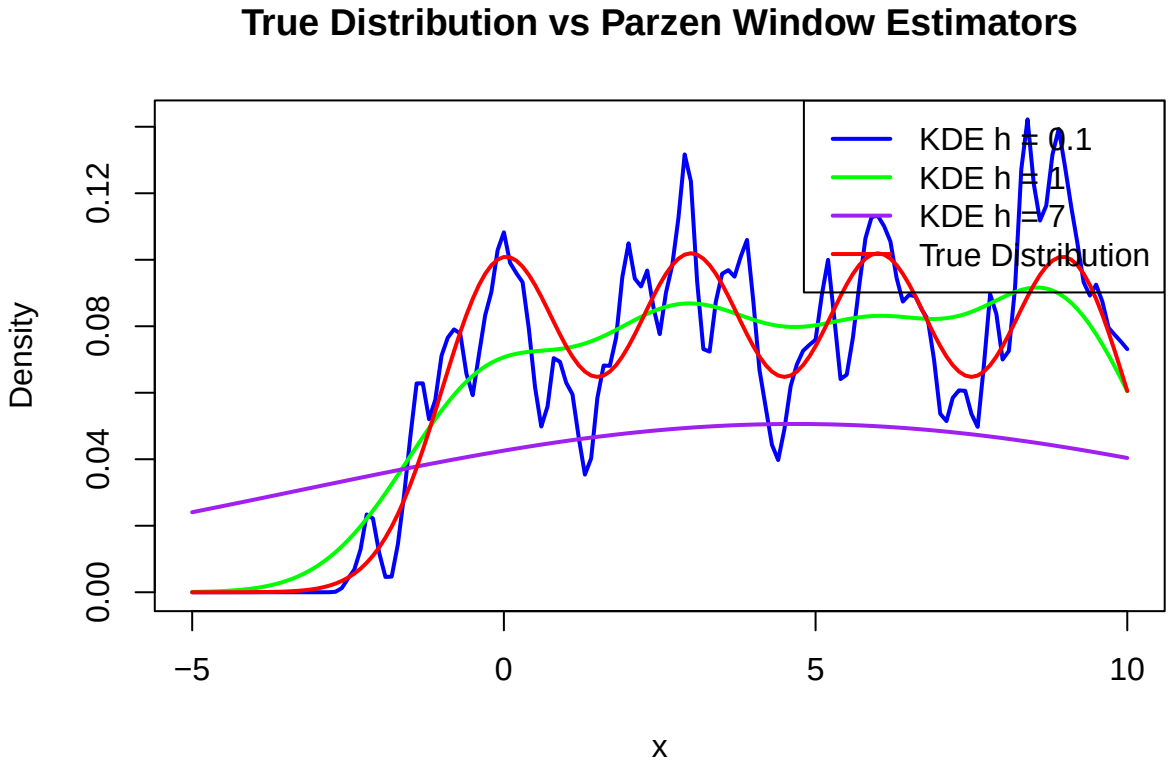
```
## [1] "show"
```

```r
setMethod("show", signature = "Parzen.smoother", function(object) {

  precision <- 8

  print_in_chunks <- function(vec, precision) {
```

```
    vec <- format(round(vec, precision), nsmall = precision)
    for (i in seq(1, length(vec), by = 6)) {
      cat(vec[i:min(i+5, length(vec))], "\n")
    }
  }
  cat("Data:\n")
  print_in_chunks(object@Data, precision)
  cat("Probs:\n")
  print_in_chunks(object@Probs, precision)
  cat("Size:\n", length(object@Probs), "\n")
  cat("Bandwidth (h):\n", format(round(object@h, precision), nsmall = precision), "\n")
})
```

```
# Defining a function for the true PDF
true_pdf <- function(x) {
  0.25 * dnorm(x, mean = 0, sd = 1) +
  0.25 * dnorm(x, mean = 3, sd = 1) +
  0.25 * dnorm(x, mean = 6, sd = 1) +
  0.25 * dnorm(x, mean = 9, sd = 1)
}

  #generating random uniform variable to calculate a Multimodal distribution such that there is a 25% c
  x <- matrix(0, nrow = 800, ncol = 2)

  #first column has the random uniform
  x[, 1] <- runif(800, 0, 1)

  #for each element of the first column of the matrix that is between 0 and .25: impute a Gaussian rand
  x[x[,1] <= 0.25, 2] <- rnorm(sum(x[,1] <= 0.25), mean = 0, sd = 1)

  #same thing for the rest of the distributions
  x[x[,1] > 0.25 & x[,1] <= 0.5, 2] <- rnorm(sum(x[,1] > 0.25 & x[,1] <= 0.5), mean = 3, sd = 1)

  x[x[,1] > 0.5 & x[,1] <= 0.75, 2] <- rnorm(sum(x[,1] > 0.5 & x[,1] <= 0.75), mean = 6, sd = 1)

  x[x[,1] > 0.75 & x[,1] <= 1, 2] <- rnorm(sum(x[,1] > 0.75 & x[,1] <= 1), mean = 9, sd = 1)

  #now our data set for the estimator is the vector of our multi-modal distribution
  data <- x[,2]


#we will be evaluating the true pdf function on the same domain as we will evaluate the estimator
#estimator Input vector, each descrete step will be it's own alpha, alpha = 5, alpha = 4.9 ...
Input <- seq(-5, 10, by = 0.1)

#initiating output vector
true_pdf_values_out <- numeric(length(Input))

#evaluating true pdf function at the same input arguments that we will evaluate our estimator
true_pdf_values_out <- true_pdf(Input)

#Here we will evaluate the estimator at each of the h step sizes as a preliminary investigation into th

# h = 0.1
```

```r
vessel_0.1 <- new("Parzen.smoother", Data = data, Probs = numeric(), input = Input, h = 0.1)
vessel_0.1_out <- Parzen.creator(vessel_0.1)

# h = 1
vessel_1 <- new("Parzen.smoother", Data = data, Probs = numeric(), input = Input, h = 1)
vessel_1_out <- Parzen.creator(vessel_1)

# h = 7
vessel_7 <- new("Parzen.smoother", Data = data, Probs = numeric(), input = Input, h = 7)
vessel_7_out <- Parzen.creator(vessel_7)

# Plot object@input against object@Probs for different bandwidths and true distribution on the same gra

# Plot for h = 0.1
plot(vessel_0.1_out@input, vessel_0.1_out@Probs, type = "l", col = "blue", lwd = 2,
     ylim = range(0, max(true_pdf_values_out, vessel_0.1_out@Probs, vessel_1_out@Probs, vessel_7_out@Pr
     xlim = range(vessel_0.1_out@input),
     main = "True Distribution vs Parzen Window Estimators",
     xlab = "x", ylab = "Density")

# Add lines for h = 1 and h = 7
lines(vessel_1_out@input, vessel_1_out@Probs, col = "green", lwd = 2)
lines(vessel_7_out@input, vessel_7_out@Probs, col = "purple", lwd = 2)

# Add the true PDF line
lines(Input, true_pdf_values_out, col = "red", lwd = 2)

# Add legend
legend("topright", legend = c("KDE h = 0.1", "KDE h = 1", "KDE h = 7", "True Distribution"),
       col = c("blue", "green", "purple", "red"), lwd = 2)
```



The preliminary investigation indicates that step size h=1 is the best choice. From here we will demonstrate this more rigorously.

```
# initialize our matrix to store all 150 trials containing 800 samples
Master <- matrix(0, nrow = 150, ncol = 800)

for (trial in 1:150) {

  #this is the same thing as before, just that we do it 150 times and store each trial in it's own row

  #reinitialize the individual trial each iteration
  x <- matrix(0, nrow = 800, ncol = 2)

  x[, 1] <- runif(800, 0, 1)

  x[x[,1] <= 0.25, 2] <- rnorm(sum(x[,1] <= 0.25), mean = 0, sd = 1)
  x[x[,1] > 0.25 & x[,1] <= 0.5, 2] <- rnorm(sum(x[,1] > 0.25 & x[,1] <= 0.5), mean = 3, sd = 1)
  x[x[,1] > 0.5 & x[,1] <= 0.75, 2] <- rnorm(sum(x[,1] > 0.5 & x[,1] <= 0.75), mean = 6, sd = 1)
  x[x[,1] > 0.75 & x[,1] <= 1, 2] <- rnorm(sum(x[,1] > 0.75 & x[,1] <= 1), mean = 9, sd = 1)

  #storing the completed trial in each row for each iteration
  Master[trial,] <- x[, 2]
}
```

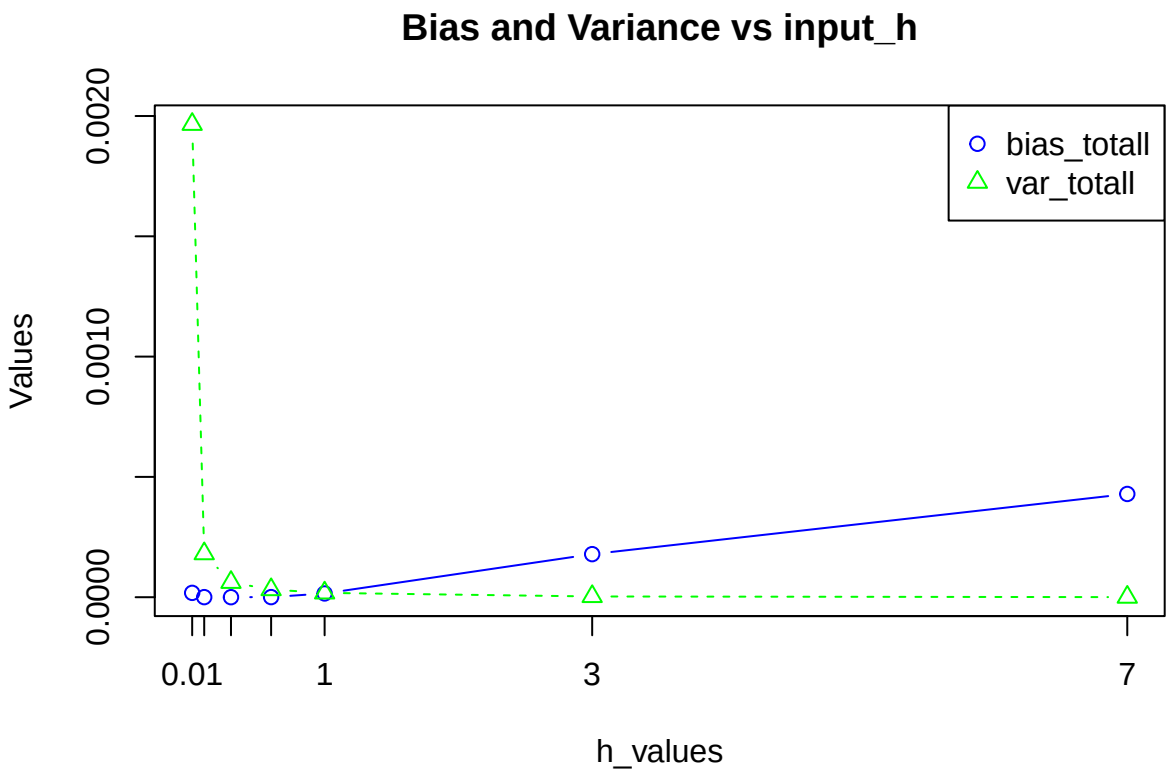
```
Bias_variance_analysis <- function(input_h){

Master_evaluated <- matrix(0,nrow=150,ncol=151)

#evaluating the function with h as input argument on the rows of master matrix
#the input data set being master and output data set being master evaluated
for (i in 1:150) {
```

```
trial_evaluated <- new("Parzen.smoother", Data = Master[i,], Probs = numeric(), input = Input, h = inpu
trial_evaluated <- Parzen.creator(trial_evaluated)
Master_evaluated[i,] <- trial_evaluated@Probs
}

#evaluate the mean for each column of the matrix, which is the mean of the each alpha value across each
mean_for_each_alpha <- matrix(0, nrow = 1, ncol = 151)
mean_for_each_alpha[1, ] <- colMeans(Master_evaluated[, seq(1:151)])

#variance
variance_for_each_alpha <- matrix(0,nrow=1,ncol=151)
for(i in 1:151){
    variance_for_each_alpha[1,i] <- mean((Master_evaluated[seq(1:150),i]-mean_for_each_alpha[1,i])^2)
}

#true distribution value for bias calculation
true_pdf_values_out <- true_pdf(Input)

#bias
bias <- matrix(0,nrow=1,ncol=151)
for (i in 1:151) {
  bias[1,i] <- mean_for_each_alpha[1,i] - true_pdf_values_out[i]
}

#MSE
MSE <- numeric(151)
for (i in 1:151) {
  MSE[i] <- variance_for_each_alpha[1,i] + (bias[i])^2
}

#total bias
bias_totall <- numeric(1)
for (i in 1:151)
bias_totall <- mean((bias[i])^2)

#total variance
var_totall <- numeric(1)
for (i in 1:151)
var_totall <- mean((variance_for_each_alpha[i]))

totall_MSE <- bias_totall + var_totall

return(c(input_h, bias_totall, var_totall, totall_MSE))
}
```

```
choose_your_h <- function(h_values) {

results <- lapply(h_values, function(h) Bias_variance_analysis(input_h = h))

results_matrix <- do.call(rbind, results)

rownames(results_matrix) <- paste("h =", h_values)
colnames(results_matrix) <- c("input_h", "bias_totall", "var_totall", "MSE")
```

```
(matplot(h_values, results_matrix[, 2:3], type = "b", pch = 1:2, col = c("blue", "green"),
         xlab = "h_values", ylab = "Values", main = "Bias and Variance vs input_h", xaxt = "n"))
    (axis(1, at = h_values, labels = h_values))
(legend("topright", legend = c("bias_totall", "var_totall"), col = c("blue", "green"), pch = 1:2))

print(results_matrix)
cat("Optimal h value:", h_values[which.min(results_matrix[,4])])
}
```

```
#since from the preliminary investigation 1 was the best option, let's look deeper
choose_your_h(c(.01,.1,.3,.6,1,3,7))
```

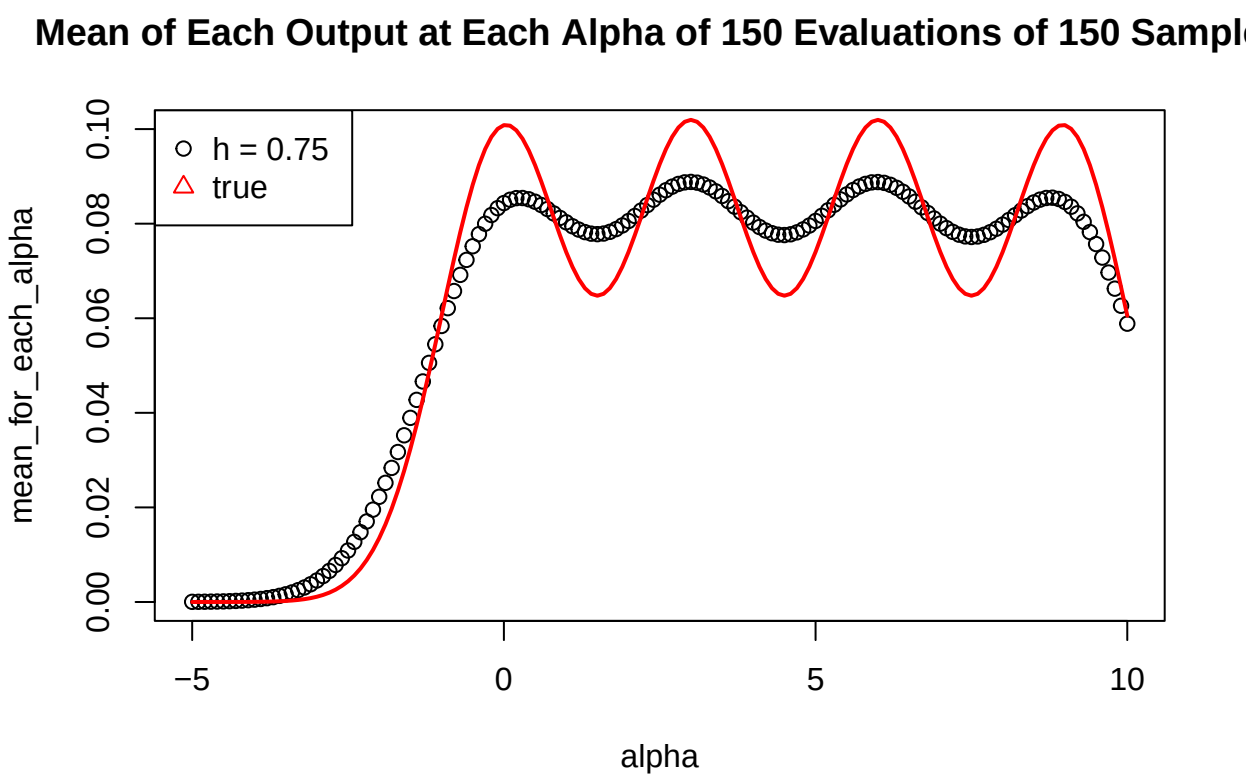



```
##          input_h  bias_totall   var_totall          MSE
## h = 0.01    0.01 1.798236e-05 1.965550e-03 1.983532e-03
## h = 0.1     0.10 2.868236e-07 1.809180e-04 1.812048e-04
## h = 0.3     0.30 1.046046e-07 6.186621e-05 6.197081e-05
## h = 0.6     0.60 3.699415e-07 3.166282e-05 3.203276e-05
## h = 1       1.00 1.553985e-05 1.793238e-05 3.347223e-05
## h = 3       3.00 1.789675e-04 3.075434e-06 1.820429e-04
## h = 7       7.00 4.290853e-04 2.612371e-07 4.293465e-04
## Optimal h value: 0.6
```

```
plot_mean_for_each_output <- function(input_h) {

  Master_evaluated <- matrix(0,nrow=150,ncol=151)

for (i in 1:150) {
trial_evaluated <- new("Parzen.smoother", Data = Master[i,], Probs = numeric(), input = Input, h = input
trial_evaluated <- Parzen.creator(trial_evaluated)
Master_evaluated[i,] <- trial_evaluated@Probs
```

```
}


mean_for_each_alpha <- matrix(0, nrow = 1, ncol = 151)
mean_for_each_alpha[1, ] <- colMeans(Master_evaluated[, seq(1:151)])

alpha = seq(-5,10,by=0.1)
plot(alpha, mean_for_each_alpha, ylim = c(0,0.1))
lines(Input, true_pdf_values_out, col = "red", lwd = 2)
title("Mean of Each Output at Each Alpha of 150 Evaluations of 150 Samples")
(legend("topleft", legend = c("h = 0.75 ", "true"), col = c("black", "red"), pch = 1:2))
}
```

```
plot_mean_for_each_output(input_h=.75)
```




```
## $rect
## $rect$w
## [1] 3.16711
##
## $rect$h
## [1] 0.0243609
##
## $rect$left
## [1] -5.6
##
## $rect$top
## [1] 0.104
##
##
## $text
```

```
## $text$x
## [1] -4.676046 -4.676046
##
## $text$y
## [1] 0.0958797 0.0877594
```

```
#this is a remarkably good approximation
```