Parzen Rosenblatt KDE Bandwidth Analysis on Ill Conditioned Sample

Jacob Richards

2024-10-10

```r
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]  0.1831131 -0.2990057 -2.3165653 -0.3314470  0.2224101  1.6815151
##  [7] -0.3518467  1.5666341  0.1428357 -2.2505130
##
## Slot "h":
## [1] 0.2773091
##
## Slot "input":
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## Slot "Probs":
##  [1]  3.077407e-02  1.168176e-01  2.001954e-06  9.807285e-17  1.160502e-32
##  [6]  3.144798e-54  1.926275e-81 2.659029e-114 8.266354e-153 5.786626e-197
```

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

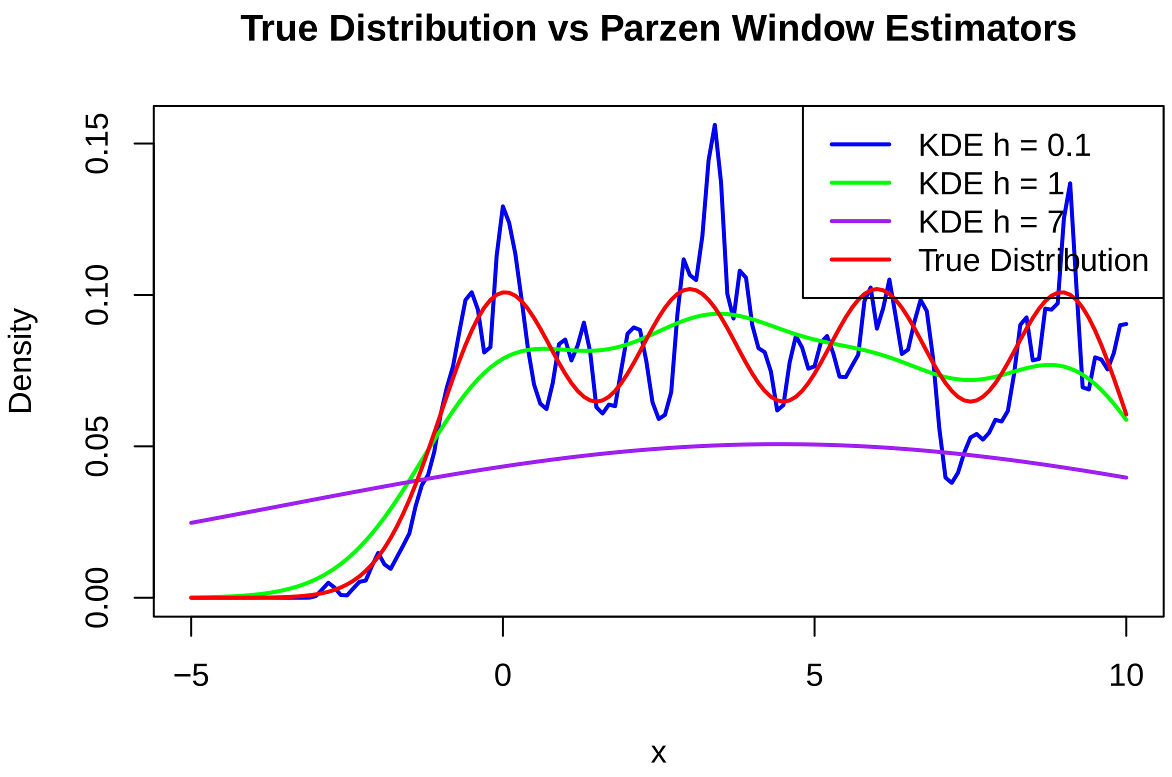
```
## Creating a new generic function for 'show' in the global environment
```

```
## [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
```

```r
# h = 0.1
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.

```r
# 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]
}
```

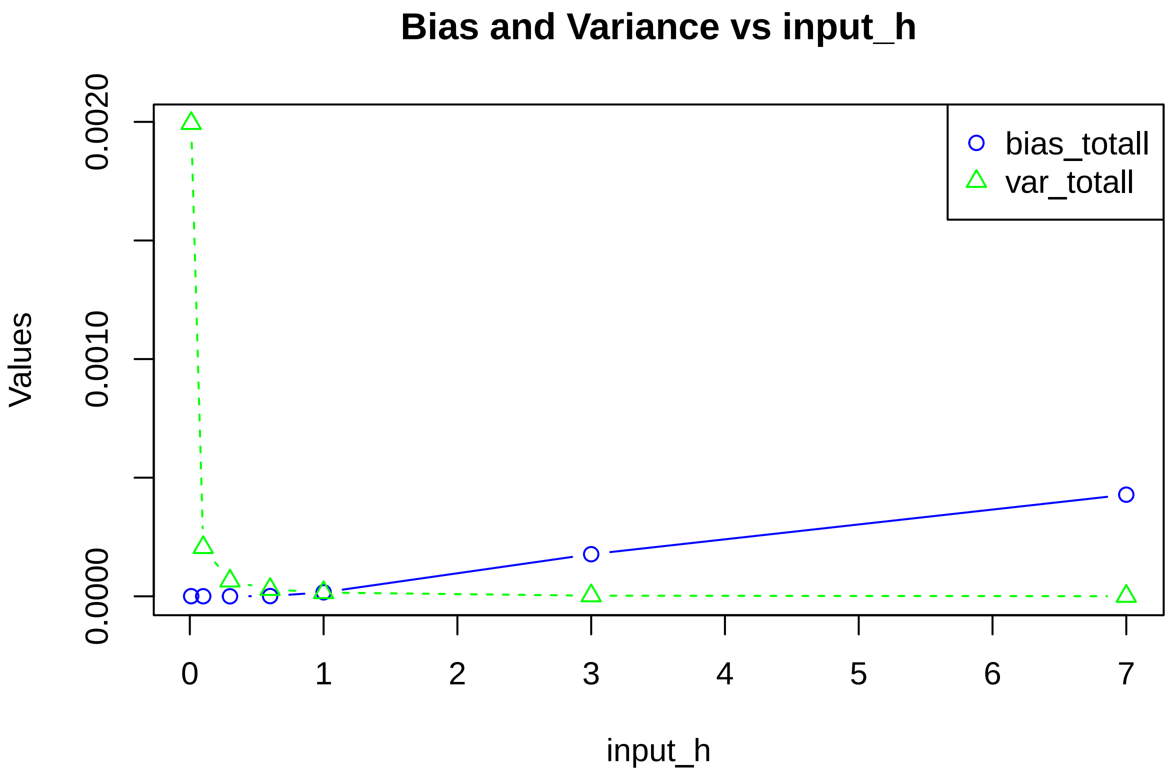
```r
Bias_variance_analysis <- function(input_h){

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

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


Master_evaluated <- table_of_evaluations_at_trials

column_names <- seq(-5, 10, by = 0.1)

colnames(Master_evaluated) <- column_names

rows <- seq(1:150)

rownames(Master_evaluated) <- rows

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


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_pdf_values_out <- true_pdf(Input)

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


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


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

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

```r
h_values <- c(.01, .1, .3, .6, 1, 3, 7)
results <- lapply(h_values, function(h) Bias_variance_analysis(input_h = h))

results_matrix <- do.call(rbind, results)
```

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

results_matrix
```

```
##          input_h  bias_totall   var_totall
## h = 0.01    0.01 4.695590e-07 1.993582e-03
## h = 0.1     0.10 2.011973e-07 2.058623e-04
## h = 0.3     0.30 2.275656e-09 6.478870e-05
## h = 0.6     0.60 9.067484e-07 2.938561e-05
## h = 1       1.00 1.666403e-05 1.577116e-05
## h = 3       3.00 1.774369e-04 2.592528e-06
## h = 7       7.00 4.283717e-04 2.087384e-07
```

```r
matplot(results_matrix[, 1], results_matrix[, 2:3], type = "b", pch = 1:2, col = c("blue", "green"),
        xlab = "input_h", ylab = "Values", main = "Bias and Variance vs input_h")

legend("topright", legend = c("bias_totall", "var_totall"), col = c("blue", "green"), pch = 1:2)
```



```r
MSE <- results_matrix[,2] + results_matrix[,3]

(h_values[which.min(MSE)])
```

```
## [1] 0.6
```