June 7, 2023

Bias-variance trade-off with infinite-dimensional nuisance parameter

Anders Munch

Introduction

In this exercise we consider the following statistical problem and estimation strategy. We are interested in estimating the parameter

$$\Psi(P) = P(X \leq -2),$$

when we assume that P belongs to a model $\mathcal{P}$ such that all measure in $\mathcal{P}$ has a density with respect to Lebesgue measure. Under this assumption, the target parameter can be written as

$$P(X \leq -2) = \int_{-\infty}^{-2} f(x)\,\mathrm{d}x,$$

where f is the density of P (with respect to Lebesgue measure).

We now choose to first estimate the density f using a kernel-based density estimator. For any $x \in \mathbb{R}$ this estimator is given as

$$\hat{f}_h(x) = \mathbb{P}_n[k_h(x, \cdot)] = \frac{1}{n}\sum_{i=1}^{n} k_h(x, X_i),$$

for some kernel function k_h that depends on the tuning parameter h which is called the bandwidth. We use the Gaussian kernel which means that

$$k_h(x, y) = \frac{1}{h}k\left(\frac{x-y}{h}\right), \quad \text{with} \quad k(z) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2}.$$

Plugging this estimator into the expression above we obtain our target estimator

$$\hat{\Psi}_n = \int_{-\infty}^{-2} \hat{f}_h(x)\,\mathrm{d}x.$$

In the following exercises we consider how the bandwidth tuning parameter h influences both our nuisance parameter estimator $\hat{f}_h$ and our target estimator $\hat{\Psi}_n$.

We use the following libraries

```
library(data.table)
library(ggplot2)
```

1 Kernel estimator

We start by defining the true nuisance and target parameter

```r
f0 <- function(x) dnorm(x)/2 + dnorm(x, sd=3, mean=3)/2
F0 <- function(x) pnorm(x)/2 + pnorm(x, sd=3, mean=3)/2
Psi0 <- F0(-2)
```

We can simulate data that has `f0` as density as follows.

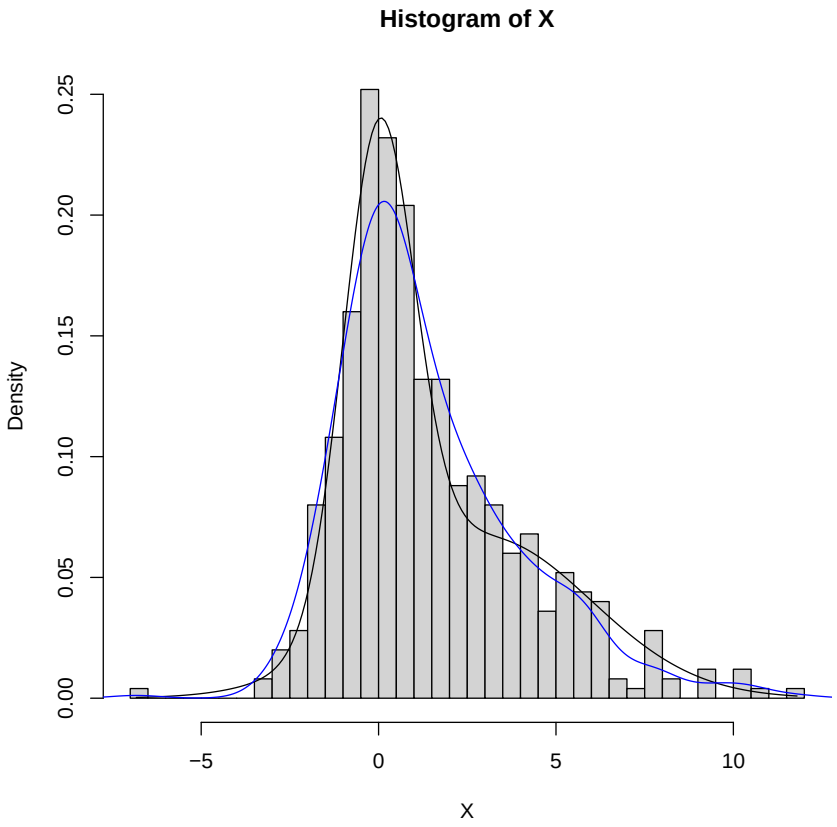
```r
sim_data <- function(n){
  hh <- 1*(runif(n)<.5)
  xx <- hh*rnorm(n) + (1-hh)*rnorm(n, sd=3, mean=3)
  return(xx)
}
```

```r
X <- sim_data(n=500)
head(X)
```

```
[1] -0.23865413 -0.02579588  2.76576294  1.42927874  0.05944578 -0.81904552
```

We can use the R-function `density` to estimate the density using a kernel-based estimator.

```r
## Plot the data and the true density
xseq <- seq(min(X), max(X), length.out=200)
hist(X, breaks=50, probability = 1, ylim = c(0,0.25))
lines(xseq, f0(xseq))

## Estimate and plot the density
bandwidth <- 0.7
f_hat <- density(X, kernel="gaussian", bw = bandwidth)
lines(f_hat, col="blue")
```



Try changing the `bandwidth` tuning parameter to see the effect. You can also try to change the number of observations generated in the simulation by supplying a different `n` to the function `sim_data`.

2 Plug-in estimator

The integral of a kernel-density estimator based on the Guassian kernel can be written as

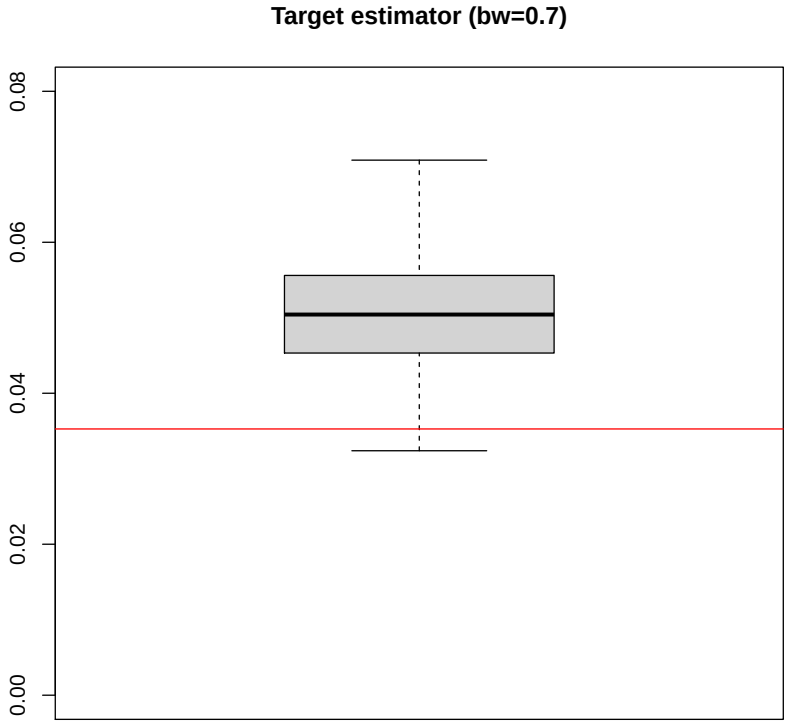
$$\begin{aligned}
\hat{\Psi}_n &= \int_{-\infty}^{x} \hat{f}_h(u)\,\mathrm{d}u \\
&= \int_{-\infty}^{x} \frac{1}{n}\sum_{i=1}^{n} k_h(X_i, u)\,\mathrm{d}u \\
&= \frac{1}{n}\sum_{i=1}^{n} \int_{-\infty}^{x} k_h(X_i, u)\,\mathrm{d}u \\
&= \frac{1}{n}\sum_{i=1}^{n} \int_{-\infty}^{x} \frac{1}{h} k\left(\frac{u - X_i}{h}\right)\mathrm{d}u \quad \text{with } k \text{ the density of } \mathcal{N}(0,1) \\
&= \frac{1}{n}\sum_{i=1}^{n} \int_{-\infty}^{\frac{x-X_i}{h}} k(z)\,\mathrm{d}z \\
&= \frac{1}{n}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \quad \text{with } K \text{ the CDF of } \mathcal{N}(0,1).
\end{aligned}$$

We can thus implement this estimator as

```
target_estimator <- function(data, bw, x = -2){
  mean(pnorm((x-data)/bw))
}
```

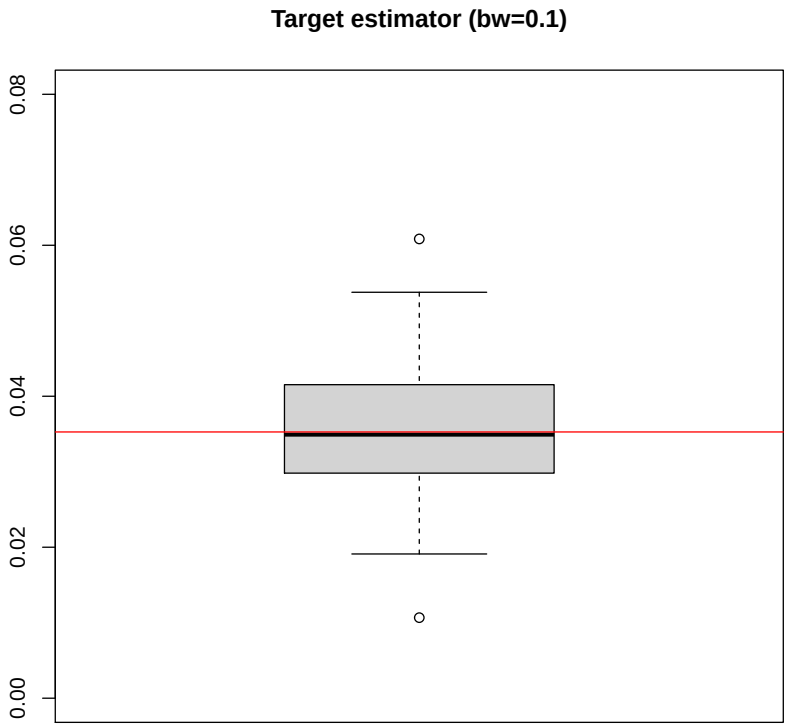
To evaluate this estimator's performance we need to simulate data repeatedly:

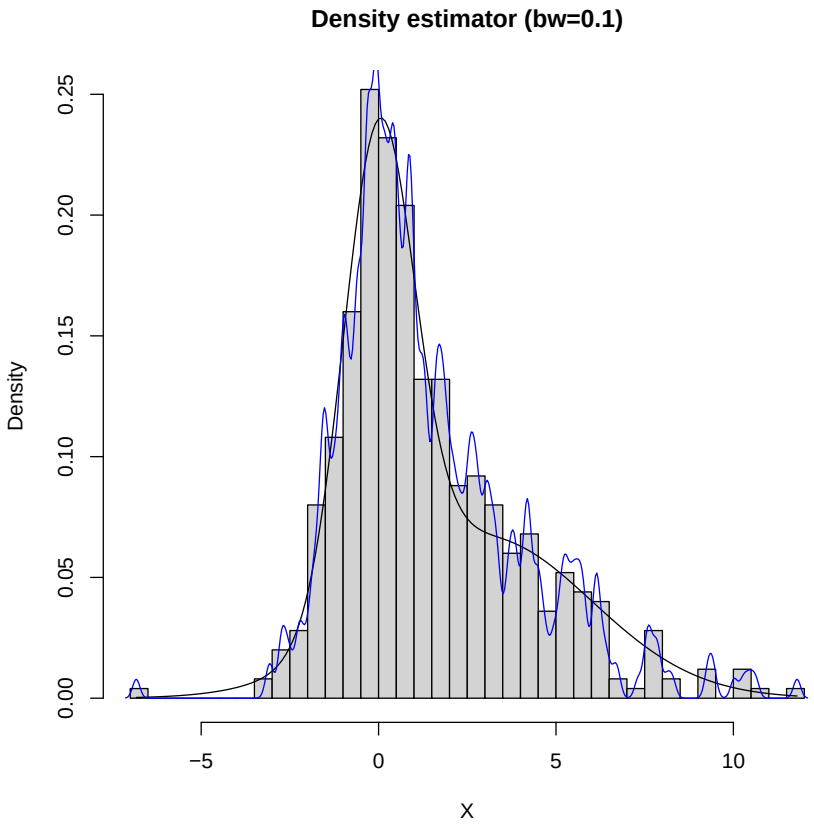
```
## Simulate 200 instance of the target estimator
target_est0 <- sapply(1:200, function(x){
  X0 = sim_data(n = 500)
  target_estimator(data = X0, bw = 0.7)
})

## Visualize and compare to true target parameter
boxplot(target_est0, main = "Target estimator (bw=0.7)", ylim=c(0,0.08))
abline(h = Psi0, col = "red")
```



1. Try changing the bandwidth tuning parameter (by supplying a different value for `bw`) to examine how this affects the performance of the nuisance estimator.

2. For a choice of `bw` that leads to a good performance for the estimator of the target parameter, try to see how the density estimator using the same `bw` performs by supplying this value to the code in the previous exercise.


Target estimator (bw=0.1)
0.00
0.02
0.04
0.06
0.08



3 Bias/variance trade-off

In this exercise we conduct a more systematic investigation of the performance of the estimators of the the nuisance parameter and the target parameter.

We start by simulating the density estimator on a dataset of size $\texttt{n} = 500$ for 25 different bandwidth values. We repeat this 100 times and calculate the (integrated) squared bias, variance and mean squared error (MSE). The MSE is the sum of the squared bias and the variance, and thus we typically use this as a measure that balances bias and variance.

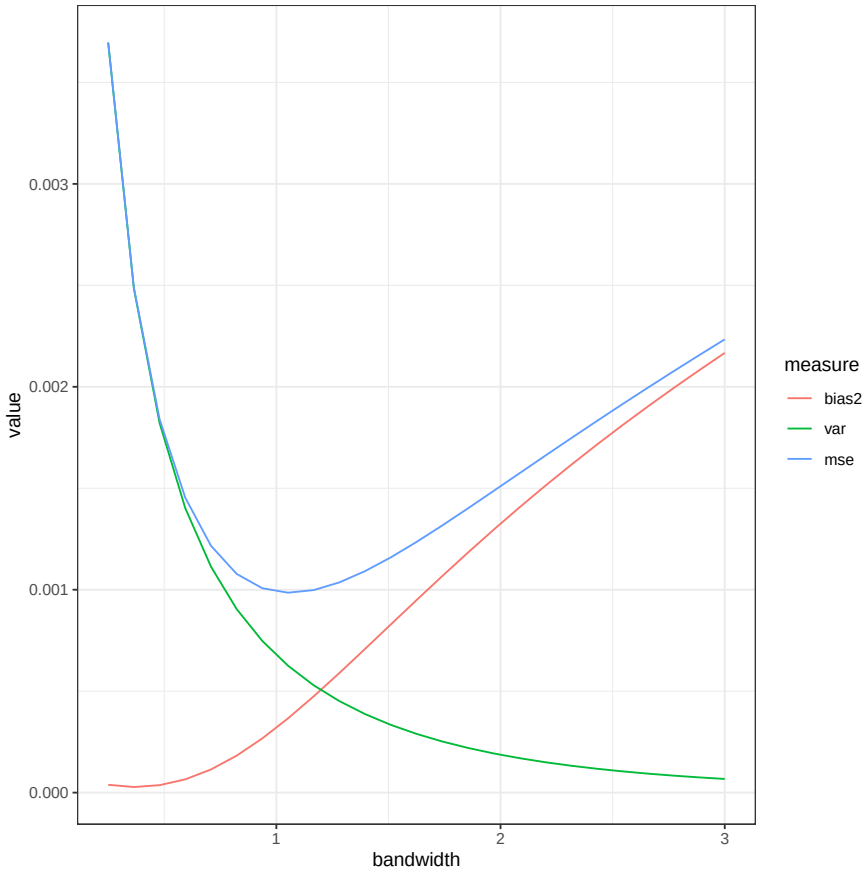
Run the following code and try to understand what happens.

```
## Function to simulate density estimator
sim_density_estimator <- function(n = 500,
                  bws = seq(0.25, 3, length.out = 25)){
  X0 = sim_data(n = n)
  out = do.call(rbind, lapply(bws, function(bw){
    f_hat = density(X0, kernel="gaussian", bw = bw,
            from = -4, to = 11, n = 512)
    x_grid = seq(-4, 11, length.out = 512)
    data.table(bandwidth = bw,
```

```
            x = x_grid,
            f0 = f0(x_grid),
            f_hat = f_hat$y)
  }))
  return(out[])
}

## Repeaet estimation 100 times and collect results
point_est <- do.call(rbind, lapply(1:100, sim_density_estimator))
point_est <- point_est[, .(bias2 = (mean(f_hat-f0))^2,
                var = var(f_hat),
                mse = mean((f_hat-f0)^2)),
                by = .(bandwidth, x)]
int_est <- melt(point_est[, .(bias2 = mean(bias2),
                   var = mean(var),
                   mse = mean(mse)),
                by = bandwidth],
         id.vars = "bandwidth",
         variable.name = "measure")
```

Visualize the results with the following code and describe the bias-variance trade-off as a function of the bandwidth.

```
ggplot(int_est, aes(x = bandwidth, y = value, col = measure)) +
  geom_line() +
  theme_bw()
```

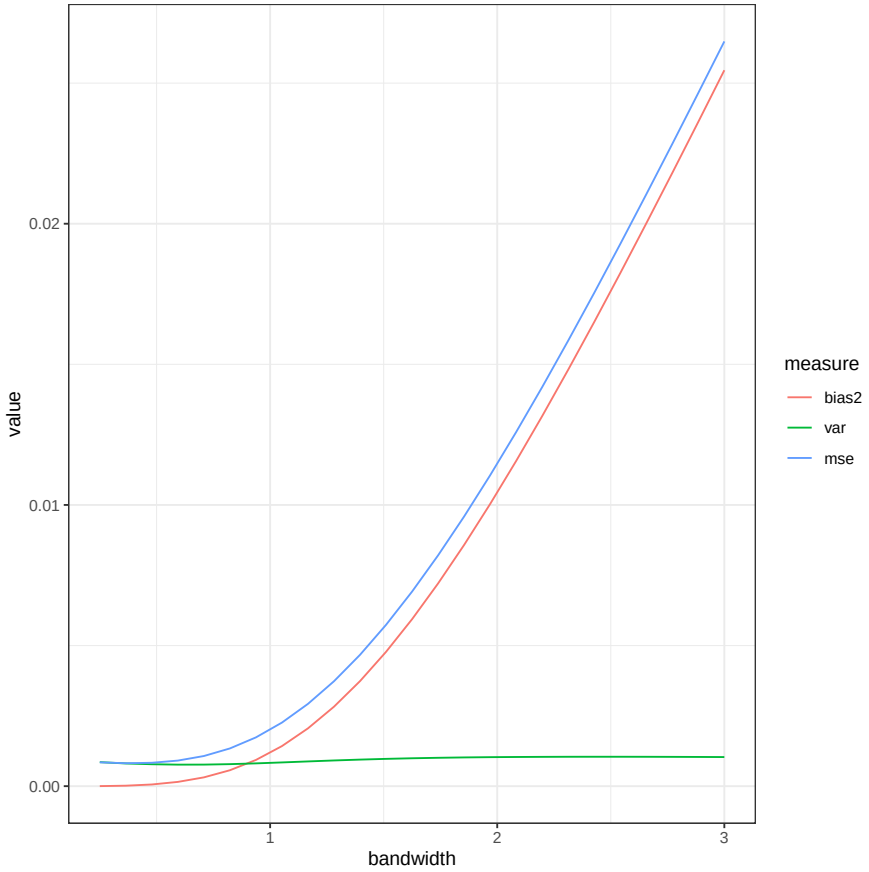


We do the same thing for the estimator of the target parameter

```
## Function to simulate target parameter with different bandwidths
sim_target_estimator <- function(n = 500,
                  bws = seq(0.25, 3, length.out = 25)){
  X0 = sim_data(n = n)
  out = do.call(rbind, lapply(bws, function(bw){
    Psi_hat = target_estimator(data = X0, bw = bw)
    data.table(bandwidth = bw, Psi0 = Psi0, Psi_hat = Psi_hat)
  }))
  return(out[])
}

## Repeat 100 times and collect results
tar_est <- do.call(rbind, lapply(1:100, sim_target_estimator))
tar_est <- tar_est[, .(bias2 = (mean(Psi_hat-Psi0))^2,
                var = var(Psi_hat),
                mse = mean((Psi_hat-Psi0)^2)),
            by = .(bandwidth)]
tar_est <- melt(tar_est,id.vars = "bandwidth",variable.name = "measure")
```

Visualize the results with the following code and compare with the plot for the estimator of the density.

```
ggplot(tar_est, aes(x = bandwidth, y = value, col = measure)) +
  geom_line() +
  theme_bw()
```



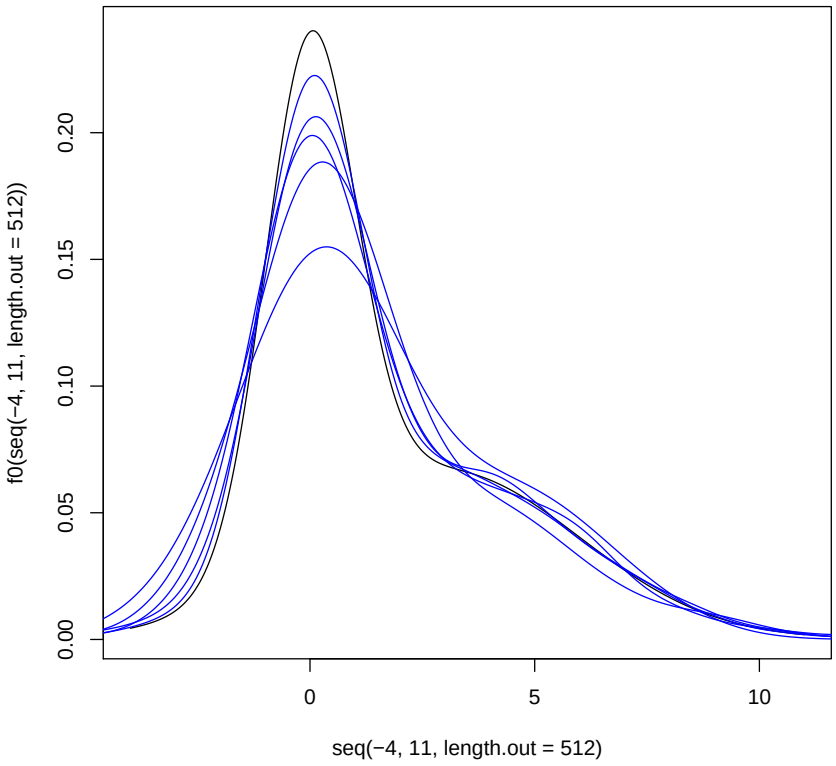
4 Inference and asymptotic normality

When n increases, we can let the bandwidth decrease because the data is more dense. It is known that, under suitable smoothness assumptions, the optimal performance for kernel-based density is given when the bandwidth is chosen as $\texttt{bw} = Cn^{-1/5}$ for some constant that depends on the true density (see for instance chapter 25 in van der Vaart [2000]). In exercise 3 we saw that a good choice for the bandwidth was around 1 when $n = 500$. Hence we can estimate C to be $C = 500^{1/5} \approx 3.4$.

We use this to construct the function `optimal_bw` which returns the (approximately) optimal value for the bandwidth as a function of n. We can verify that this looks reasonable by plotting the obtained density with increasing n.

```
## Function giving the optimal bandwidth as a function of n
optimal_bw <- function(n) 3.4*n^{-1/5}
```

```
## Verify that this looks reasonable:
## Plot the true density
plot(seq(-4, 11, length.out = 512),
     f0(seq(-4, 11, length.out = 512)),
     type = "l")

## Plot the estimated densities for increasing n
lapply(c(100, 500, 1000, 5000, 20000), function(n){
  X0 = sim_data(n)
  bw_n = optimal_bw(n)
  f_hat_n = density(X0, kernel = "gaussian", bw = bw_n)
  Sys.sleep(0.5)
  lines(f_hat_n, col = "blue")
})
```

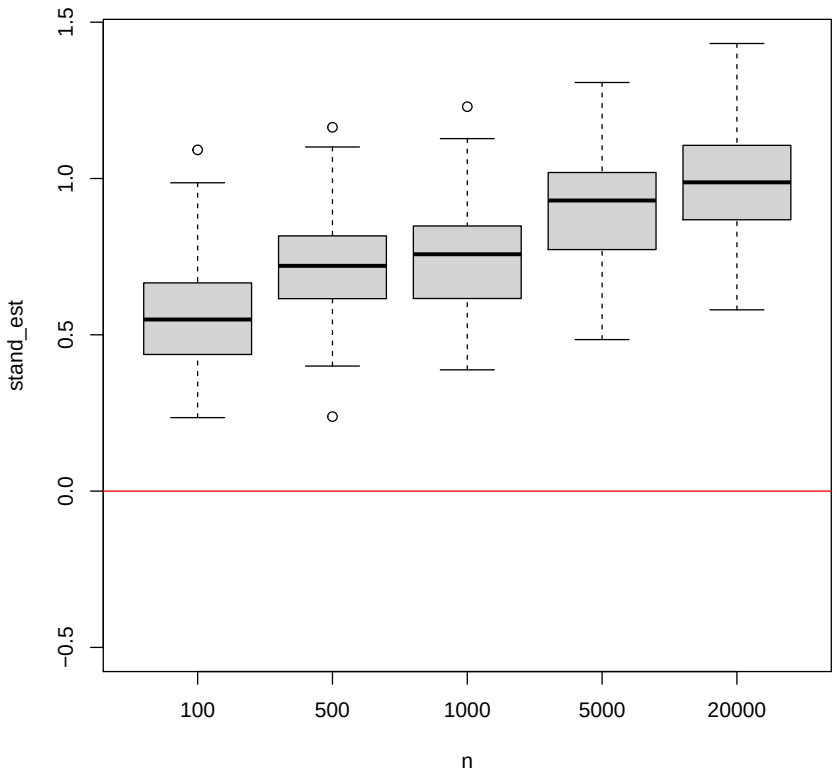


Let us now use the same automatic bandwidth selector for our target estimator. When we want to do inference for the parameter Ψ we will typically appeal to the central limit theorem to argue that

$$\sqrt{n}(\hat{\Psi}_n - \Psi) \sim \mathcal{N}(0, \sigma^2),$$

when n is large. Let us investigate if this seems correct when we use `optimal_bw` to select the bandwidth for our target estimator.

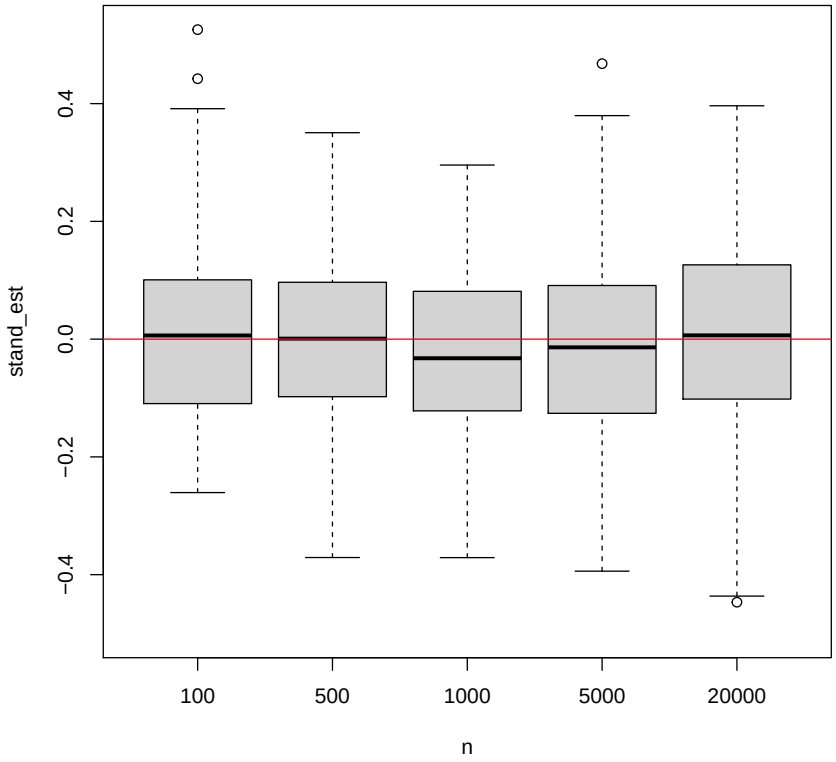
```
target_est_auto <- do.call(rbind, lapply(1:100, function(x){
  do.call(rbind, lapply(c(100, 500, 1000, 5000, 20000), function(n){
    X0 = sim_data(n)
    bw_n = optimal_bw(n)
    est = target_estimator(data = X0, bw = bw_n)
    stand_est = sqrt(n)*(est-Psi0)
    out = data.table(n = n,bw = bw_n,est = est, stand_est)
    return(out)
  }))
}))
target_est_auto[,{
  boxplot(stand_est~n, ylim = c(-0.5,max(stand_est)))
  abline(h = 0, col = "red")
}]
```



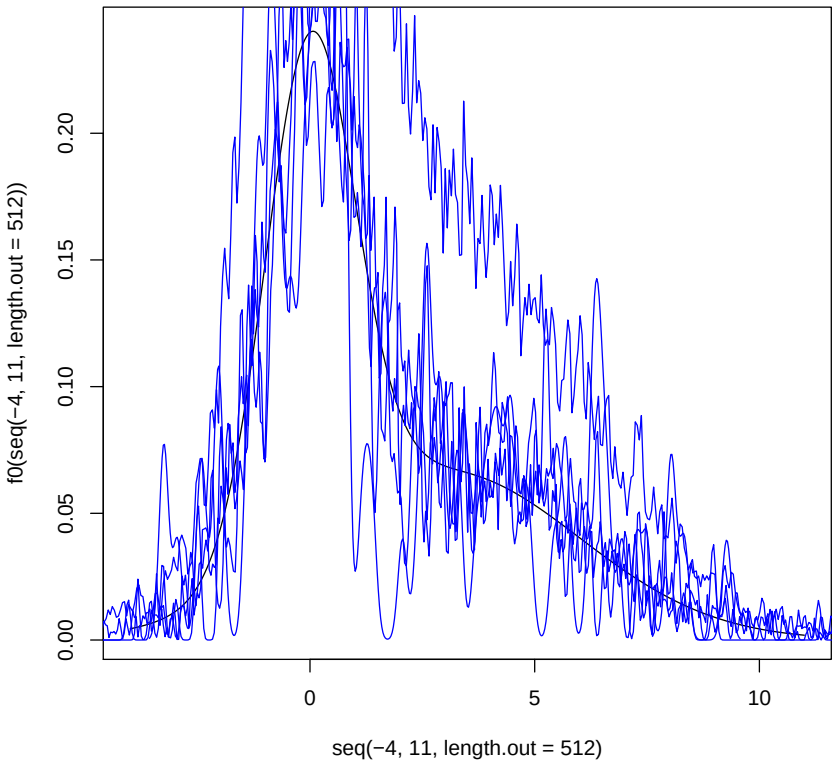
Try to see if the estimator improves if we use a different function to select the bandwidth, for instance by using the function

```
undersmooth_bw <- function(n) n^{-1/2}
```

Reuse the code snippet we used to construct `target_est_auto` but replace `optimal_bw` with `undersmooth_bw` (or your own function). Does this improve performance? Can you explain this using the plots of the bias-variance trade-off we made in exercise 3?



You can also try to rerun the code that generated the densities for increasing n, but use `undersmooth_bw` instead of `optimal_bw`.



5 References

A. W. van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.