Smoothing: A. Density Estimation

Michaela Lukacova

2024-09-16

Implement a kernel density estimator using the Epanechnikov kernel, and implement one or more bandwidth selection algorithms using either AMISE plug-in methods or cross-validation methods. Test the implementation and compare the results with the results of using density() in R.

It may be a good idea to use real data to investigate how the bandwidth selection works, but for benchmarking and profiling it is best to use simulated data. Think about how to make your implementation as general and generic as possible.

Kernel density estimators

The kernel density estimators approximate the unknown density of data points with the function

$$\hat{f}(x) = \frac{1}{hN} \sum_{j=1}^{N} K\left(\frac{x - x_j}{h}\right)$$

For a given kernel $K : \mathbb{R} \to \mathbb{R}$ and bandwidth parameter h.

Naive density estimator

Implementation

We first implement a naive kernel density estimator. It computes density estimates along a grid of points. The number of gridpoints is given by m (default 512). We use the Epanechnikov kernel:

$$K(x) = \frac{3}{4}(1 - x^2)1_{[-1,1]}(x)$$

The naive kernal density estimator takes as input x and a bandwidth parameter h.

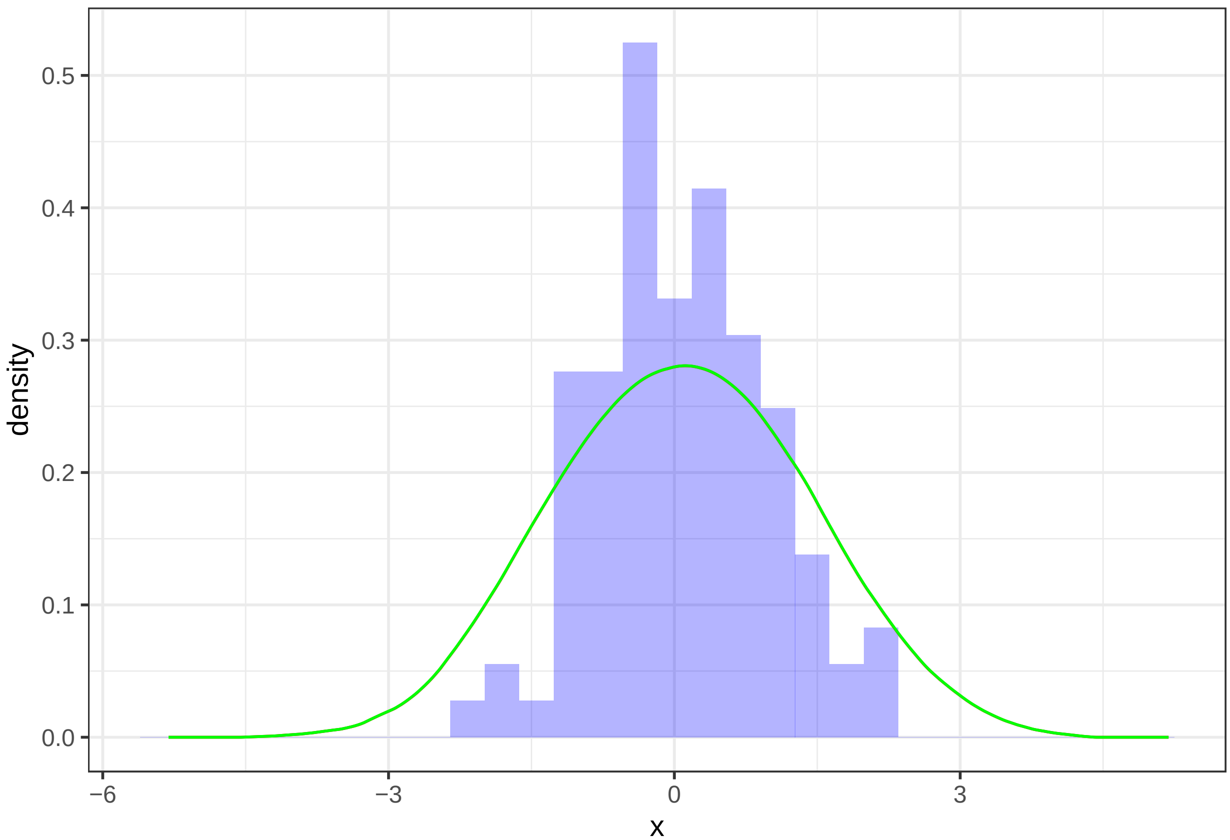
Evaluation

We check how the naive kernel density estimator approximates data with an arbitrary bandwidth parameter.

```r
# Simulate data
n <- 100
set.seed(123)
x <- rnorm(n)

# Compute kernel density estimates along grid
dens_est <- as.data.frame(kern_dens_naive(x, h = 1)[1:2])
dens_r <- as.data.frame(density(x, kernel = "epanechnikov", 1)[1:2])

# Plot along with kernel
```

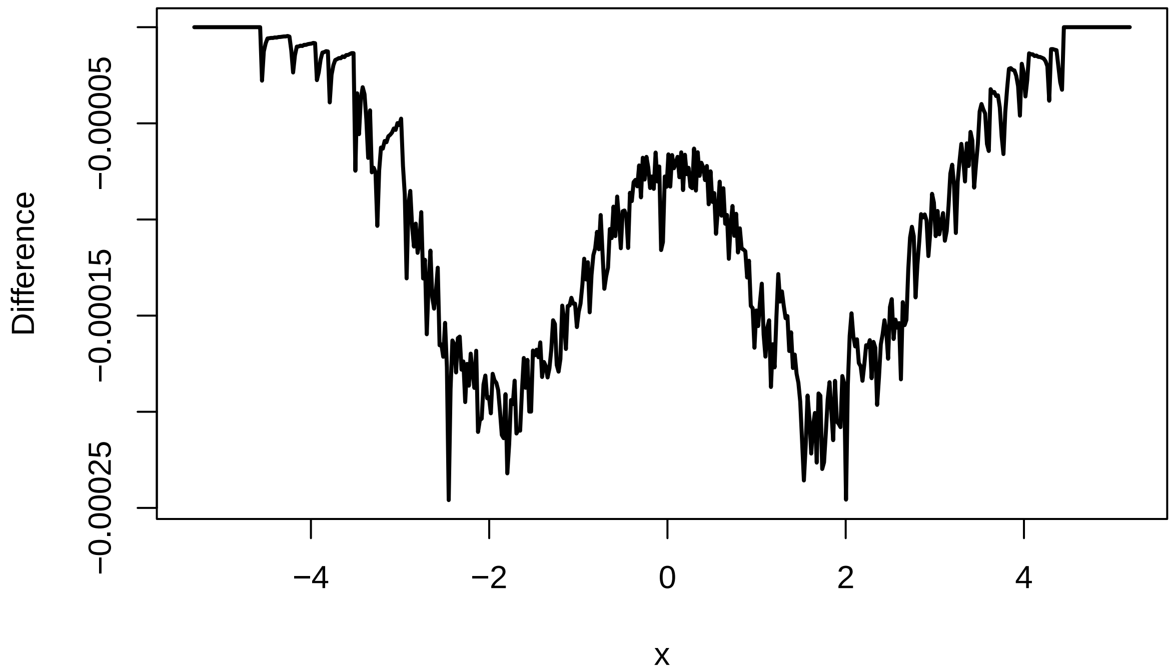
```r
ggplot(tibble(x = x), aes(x = x, y = ..density..)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.3) +
  geom_line(data = dens_est, aes(x = x, y = y), color = "red")+
  geom_line(data = dens_r, aes(x = x, y = y), color = "green")
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The approximation looks quite good. We check how it approximates the r density function

```r
# Det giver ikke mening fordi gridpoints'ene ikke er ens.
plot(
  dens_est$x,
  dens_est$y - dens_r$y,
  type = "l",
  lwd = 2,
  xlab = "x",
  ylab = "Difference"
)
```




```r
test_that("Our binned density implementation corresponds to the naive", {
  expect_equal(
    kern_dens_naive(x,1)$y,
    density(x, kernel = "epanechnikov", 1)$y,
    tolerance = 1e-3
  )
})
```

```
## Test passed
```

It approximates quite well. The error is in the magnitude of 10^{-4}.

We benchmark the two functions.

```r
# Generate random numbers
set.seed(19)
x <- rnorm(2^10)

dens_bench <- bench::press(
  k = 2^(5:10),
  {
    bench::mark(
      "r density" = density(x[1:k], 1),
      "Naive" = kern_dens_naive(x[1:k], 1),
      check = FALSE
    )
  }
)
```

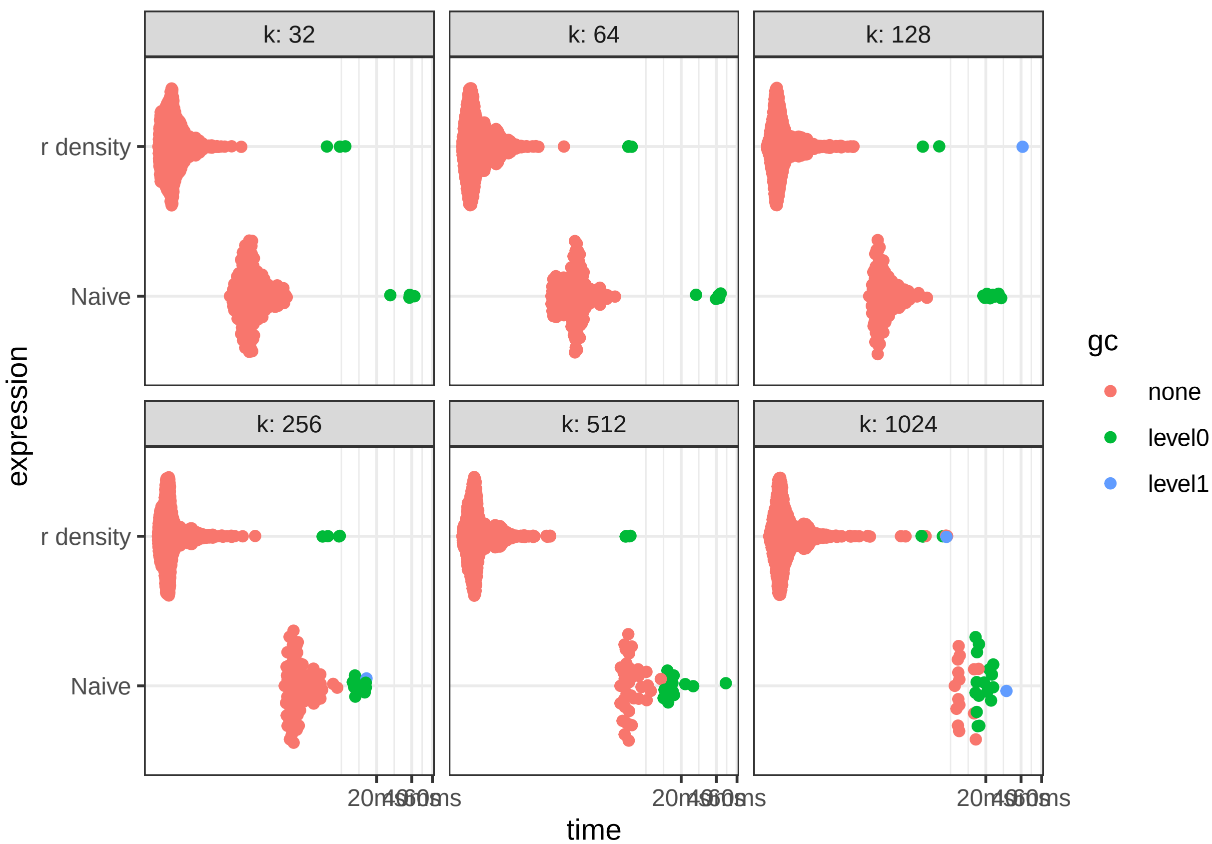
```
## Running with:
##        k
## 1    32
## 2    64
```

```
## 3   128
## 4   256
## 5   512
## 6  1024
```

```
dens_bench[c("expression", "k", "min", "median", "itr/sec", "n_gc")]
```

```
## # A tibble: 12 x 5
##    expression     k      min   median `itr/sec`
##    <bch:expr> <dbl> <bch:tm> <bch:tm>     <dbl>
##  1 r density     32  273.4us 357.42us     2593.
##  2 Naive         32   1.12ms   1.66ms      555.
##  3 r density     64    269us 348.02us     2510.
##  4 Naive         64   1.56ms   2.49ms      395.
##  5 r density    128 272.11us 345.67us     2527.
##  6 Naive        128   2.02ms   2.59ms      350.
##  7 r density    256 270.91us 334.82us     2684.
##  8 Naive        256   3.28ms   4.15ms      221.
##  9 r density    512 269.84us 347.92us     2532.
## 10 Naive        512   6.05ms   7.17ms      130.
## 11 r density   1024 281.79us 366.52us     2219.
## 12 Naive       1024  10.87ms  11.85ms       77.6
```

```
autoplot(dens_bench)
```



Our implementation is much slower.

Binned density estimator

Implementation

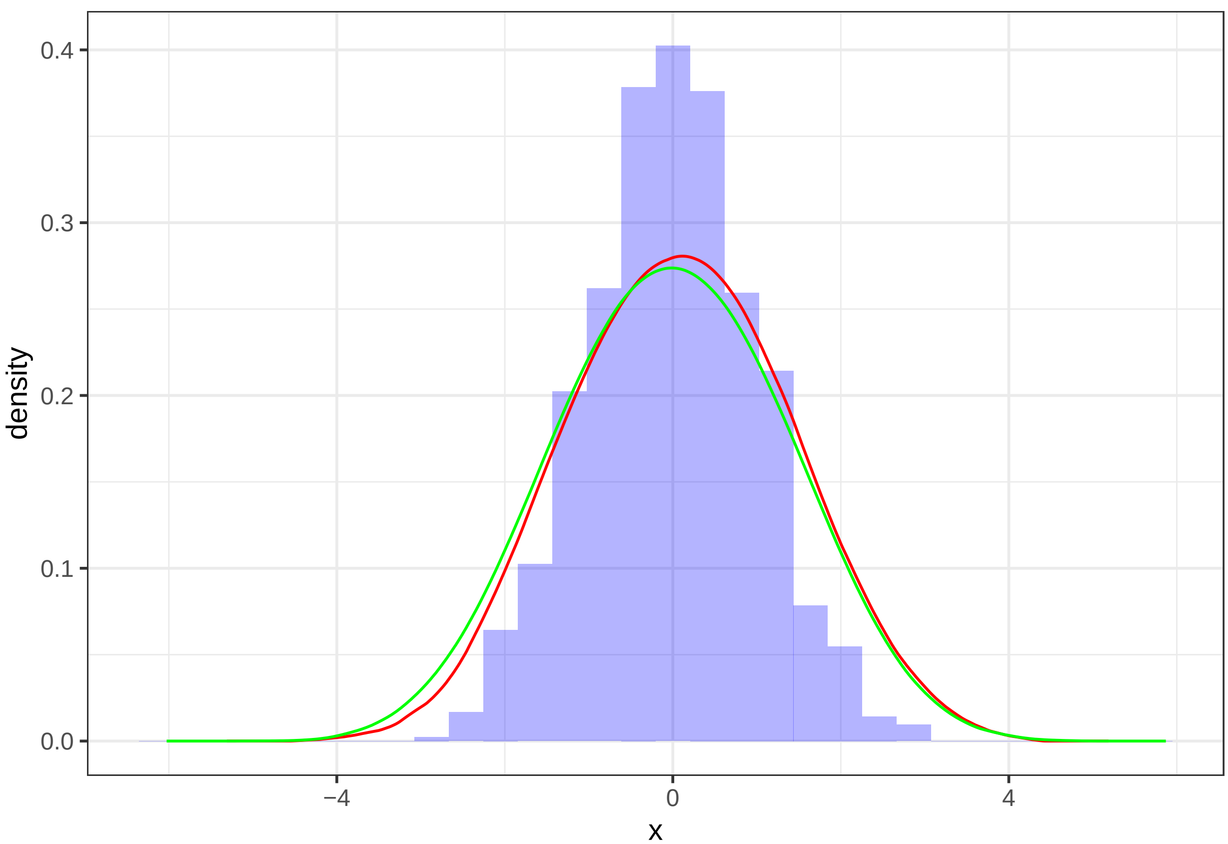
The binned density estimator is an alteration of the previous density estimator. It has the advantage of computational efficiency. The binned density estimator is computed as

$$\hat{f}(x) = \frac{1}{hN}\sum_{i=1}^{B} n_j K\left(\frac{x - c_j}{h}\right)$$

The function bins the observations into B equal length bins spanning the length of x observations. The number of observations in each bin is n_j. c_j is the center of each bin. The computational efficiency is achieved by, instead of computing K in each data point, we only compute K in each centerpoint, and then multiply by the number of observations in that bin. The concern could be that we lose accuracy, but as we'll see, this is not a problem.

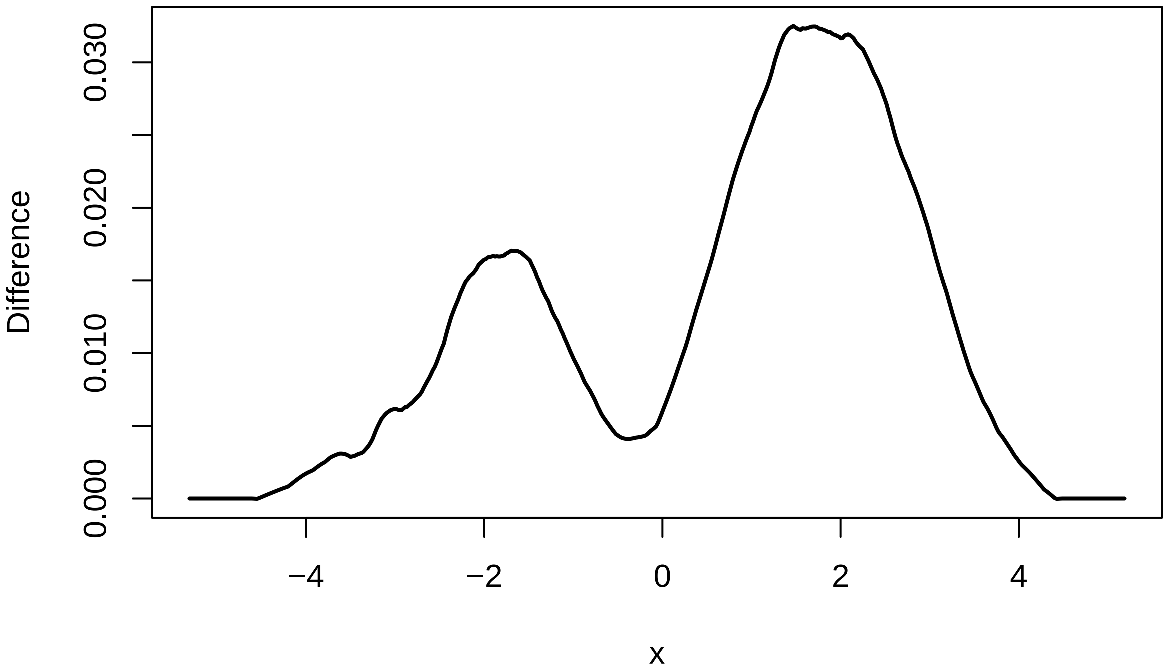
We check how the binned kernel density estimator approximates data with an arbitrary bandwidth parameter.

```r
# Compute kernel density estimates along grid
dens_est2 <- as.data.frame(kern_dens_bin(x, h = 1)[1:2])

# Plot along with kernel
ggplot(tibble(x = x), aes(x = x, y = ..density..)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.3) +
  geom_line(data = dens_est, aes(x = x, y = y), color = "red")+
  geom_line(data = dens_est2, aes(x = x, y = y), color = "green")
```



Differences in the two functions

```r
plot(
  dens_est$x,
  dens_est$y - dens_est2$y,
  type = "l",
  lwd = 2,
  xlab = "x",
  ylab = "Difference"
)
```




```r
test_that("Our binned density implementation corresponds to the naive", {
  expect_equal(
    kern_dens_naive(x,1)$y,
    kern_dens_bin(x, 1)$y,
    tolerance = 1e-2
  )
})
```

```
## Test passed
```

We benchmark the two functions.

```r
# Generate random numbers
set.seed(19)
x <- rnorm(5^10)

dens_bench <- bench::press(
  k = 2^(5:10),
  {
    bench::mark(
      "r density" = kern_dens_bin(x[1:k], 0.2),
      "Naive" = kern_dens_naive(x[1:k], 0.2),
      check = FALSE
    )
  }
```

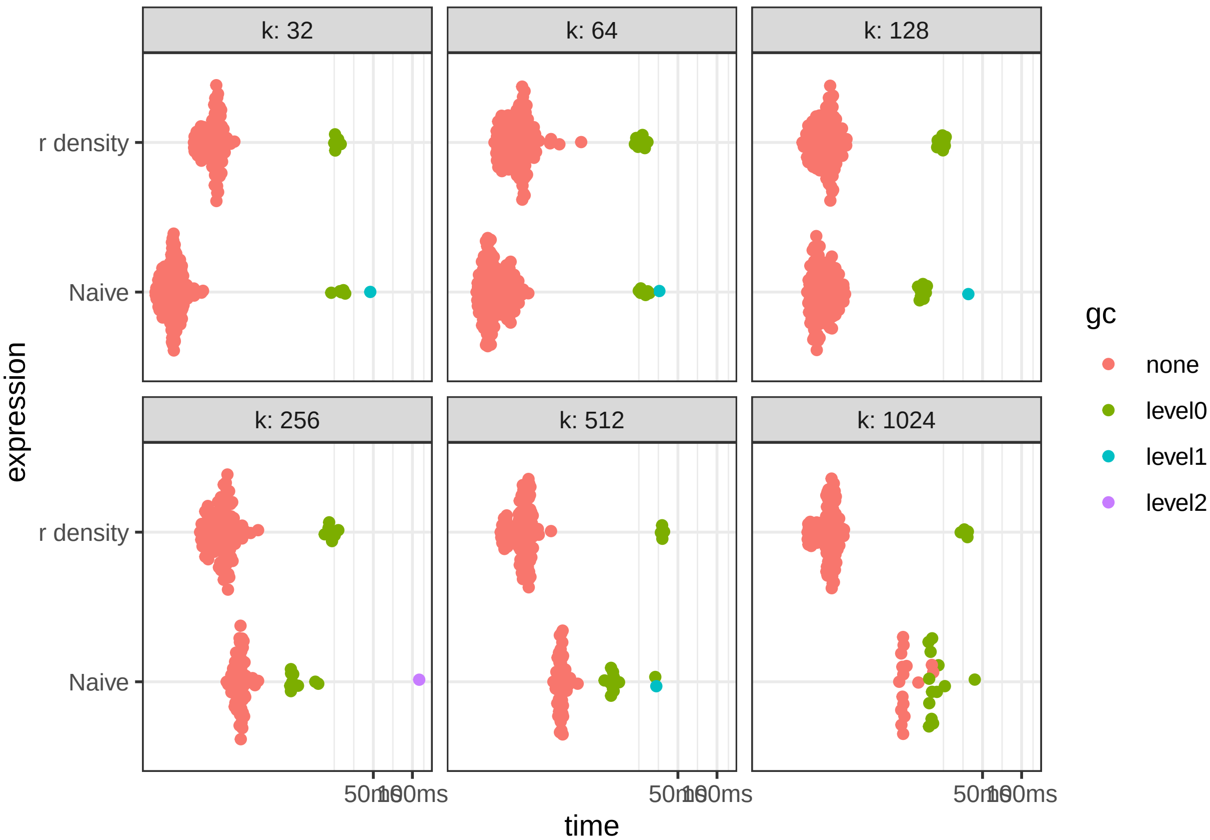
```
)
```

```
## Running with:
##       k
## 1    32
## 2    64
## 3   128
## 4   256
## 5   512
## 6  1024
```

```
dens_bench[c("expression", "k", "min", "median", "itr/sec", "n_gc")]
```

```
## # A tibble: 12 x 5
##    expression     k      min   median `itr/sec`
##    <bch:expr> <dbl> <bch:tm> <bch:tm>     <dbl>
##  1 r density     32   2.08ms   3.03ms     340.
##  2 Naive         32   1.05ms   1.44ms     690.
##  3 r density     64   1.93ms   2.96ms     332.
##  4 Naive         64   1.41ms   1.87ms     492.
##  5 r density    128   2.05ms   3.16ms     321.
##  6 Naive        128   2.23ms   2.83ms     332.
##  7 r density    256   2.31ms   3.52ms     287.
##  8 Naive        256   3.71ms    4.7ms     212.
##  9 r density    512   2.16ms   3.22ms     313.
## 10 Naive        512   5.49ms   6.32ms     157.
## 11 r density   1024   2.24ms   3.36ms     306.
## 12 Naive       1024   11.4ms  12.25ms      74.4
```

```
autoplot(dens_bench)
```



Bandwidth selection

AMISE plug-in

The Epanechnikov kernel is a square-integrable probability density with mean 0. It can be shown that

$$\sigma_K^2 = \frac{1}{5}||K||_2^2 = \frac{3}{5}$$

For the Epanechnikov kernel. The AMISE is defined as

$$AMISE(h) = \frac{||K||_2^2}{nh} + \frac{h^4\sigma_K^4||f_0''||_2^2}{4}$$

$||f_0''||_2^2$ is the squared L_2-norm of the true unknown density. Using various estimate $AMISE(h)$ can be used to estimate the asymptotically optimal bandwidth in a mean integrated squared error sense. By minimizing $AMISE(h)$ the asymptotically optimal oracle bandwidth is

$$h_N = \left(\frac{||K||_2^2}{||f_0''||_2^2\sigma_K^4}\right)^{1/5} n^{-1/5}$$

Inserting the values we have for the Epanechnikov kernel

$$h_N = \left(\frac{15}{||f_0''||_2^2}\right)^{1/5} n^{-1/5}$$

We have now arrived at a circular problem. In order to select bandwidth we need to know f, but to estimate f we need the bandwidth. A solution to this problem is to estimate h according to Silverman's rule of thumb, obtain a pilot density estimate $\tilde{f}$ and compute the squared L_2-norm, use this estimate to find h_N, in order to arrive at our final kernel density estimate.

Silverman's rule of thumb for the Epanechnikov kernel is

$$\hat{h}_n = (40\sqrt{\pi})^{1/5}\tilde{\sigma}n^{-1/5}$$

Where $\tilde{\sigma} = \min\{\hat{\sigma}, \frac{\text{IQR}}{1.34}\}$.

For the Epanechnikov kernel (H) with bandwidth h we can compute the squared L_2-norm as

$$||\tilde{f}||_2^2 = \frac{1}{n^2h^6}\sum_{i=1}^{N}\sum_{j=1}^{N}\int H''\left(\frac{x-x_i}{h}\right)H''\left(\frac{x-x_j}{h}\right)dx$$

We have

$$H''(x) = -\frac{3}{2}1_{[-1,1]}(x)$$

Note

$$\left(\frac{x-x_i}{h}\right) \in [-1,1] \Leftrightarrow x \in [x_i - r, x_i + r]$$

So

$$||\tilde{f}||_2^2 = \frac{9}{4n^2h^6}\sum_{i=1}^{N}\sum_{j=1}^{N}\int_{\max\{x_i-h,x_j-h\}}^{\min\{x_i+h,x_j+h\}} 1dx = \frac{9}{4n^2h^6}\sum_{i=1}^{N}\sum_{j=1}^{N}(\max\{0, 2h - |x_i - x_j|\})$$

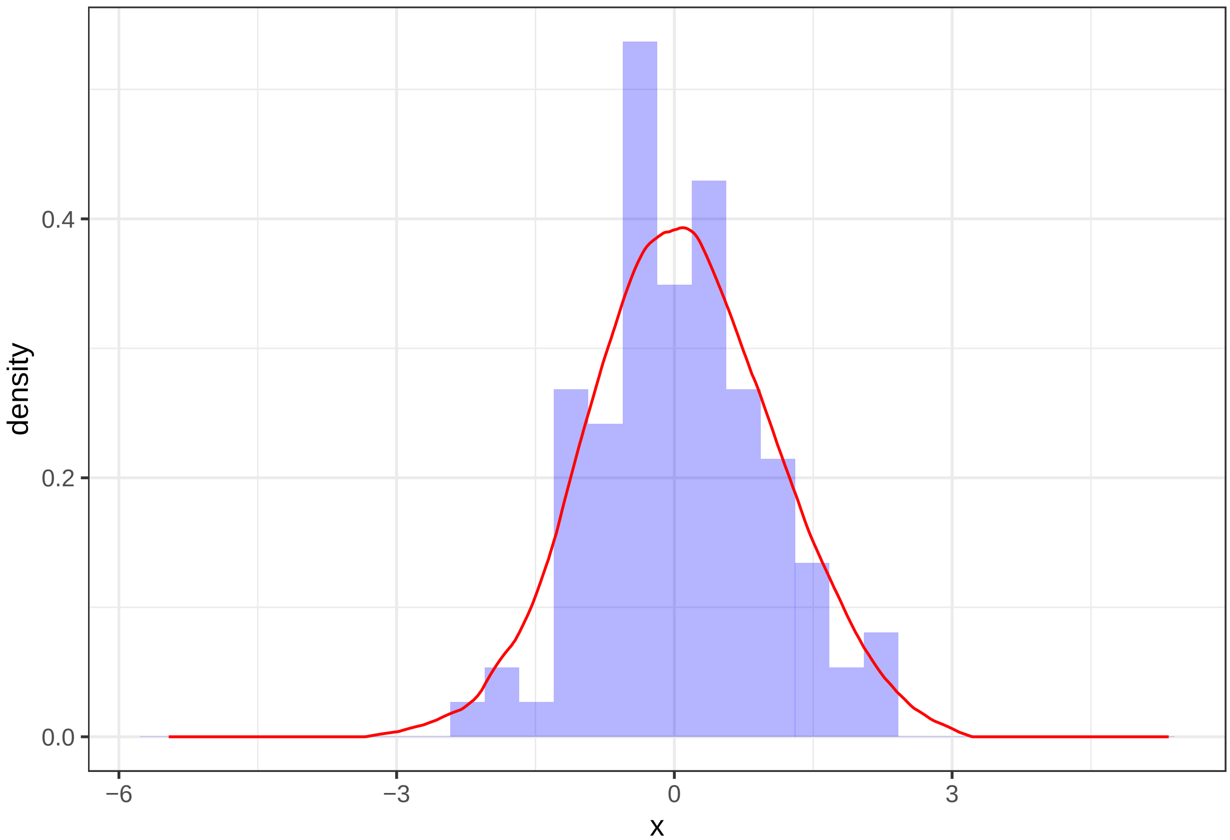
We implement the method described above in the AMISE_bw function and calculate the optimal bandwidth.

```r
# Simulate data
n <- 100
set.seed(123)
x <- rnorm(n)

op_bw <- AMISE_bw(x)
op_bw
```

```
## [1] 1.050904
```

We find a binned kernel density estimate using the optimal bandwidth found by the AMISE.

```r
# Compute kernel density estimates along grid
dens_est3 <- as.data.frame(kern_dens_bin(x, h = op_bw, norm = FALSE)[1:2])

# Plot along with kernel
ggplot(tibble(x = x), aes(x = x, y = ..density..)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.3) +
  geom_line(data = dens_est3, aes(x = x, y = y), color = "red")
```



Cross-validation methods

Test of implementation using real data