797 project formulation

Tobias Kuhlmann, Rui Zhang

11/14/2018

Data

For our project we simulate univariate data

$$[x_i, y_i] \quad i\epsilon\{1, ..., n\},$$

where $y_i \sim iid$ and an unknown and known (different cases) smooth density f(x). This may not be just one dataset, but several.

Models

Univariate kernel density estimator

We use a univariate kernel density function following Wand and Jones (1995). A density function can be estimated by

$$\hat{f}(x; h) = (nh)^{-1} \sum_{i=1}^{n} K\{(x - X_i)/h\},$$

where K is a kernel function satisfying $\int K(x)dx = 1$ and h is the bandwidth.

R resources

- Wand and Jones (1995)
- https://stat.ethz.ch/R-manual/R-devel/library/stats/html/density.html

Univariate density estimation with logspline

Let B be a collection of feasible column vectors following Stone, Hansen, Kooperberg, and Truong (1997). If $\beta\epsilon B$, then

$$f(x; \beta) = exp(\beta_1 B_1(x) + \cdots + \beta_J B_J(x) - C(\beta)), L < x < U$$

where

$$C(\beta) = log(\int_L^U exp(\beta_1 B_1(x) + \cdots + \beta_J B_J(x))dy).$$

Then $f(x; \beta)$ is a positive density function on (L,U).

R resources

- https://www.rdocumentation.org/packages/logspline/versions/2.1.11
- https://www.rdocumentation.org/packages/logspline/versions/2.1.11/topics/logspline
- https://www.rdocumentation.org/packages/logspline/versions/2.1.11/topics/dlogspline
- Stone, Hansen, Kooperberg, and Truong (1997)

Goal

After estimating both models on several sets of simulated data with different sample sizes, our goal is to study and compare the rates of convergence of the MISE as $n- > \infty$.

Simulation experiment

Simulation

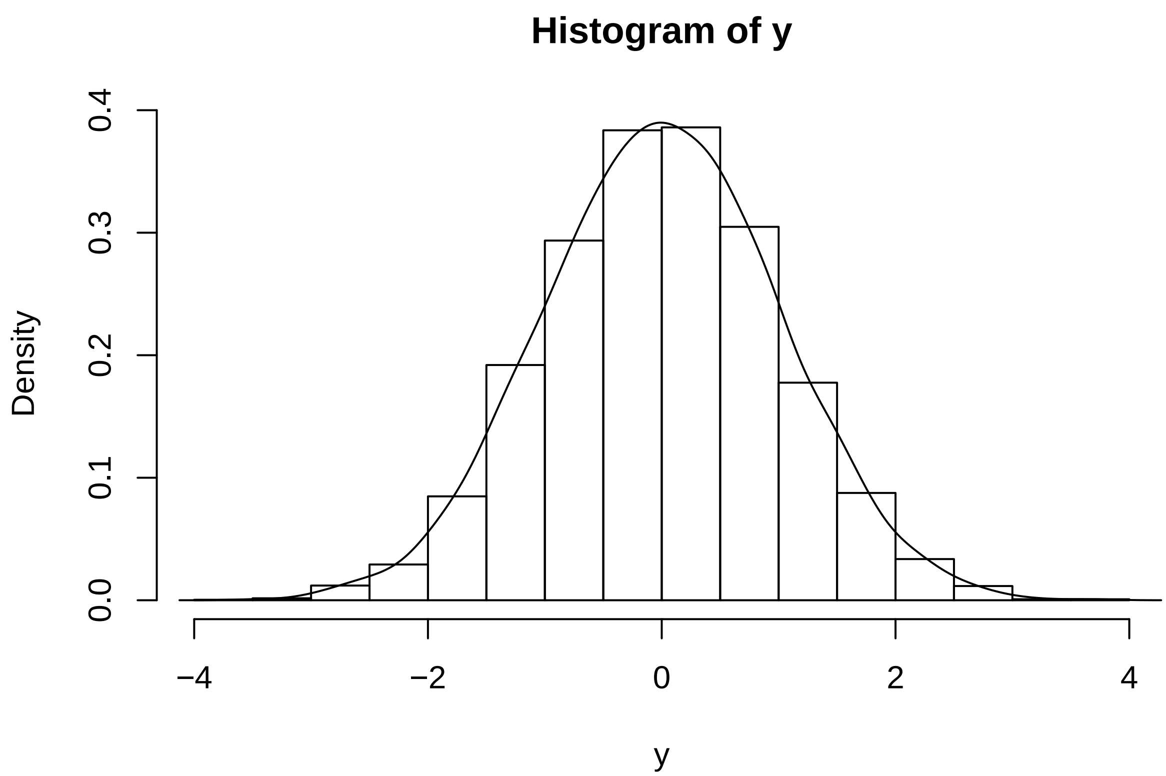
```
# TODO: Change function to known density to evaluate MISE
# rnorm

# function to estimate
f <- function(x)
    return(exp(3*x)*sin(10*x))

# aribitrary noise level
sigma <- 5
```

Stats: Univariate Kernel density estimator test

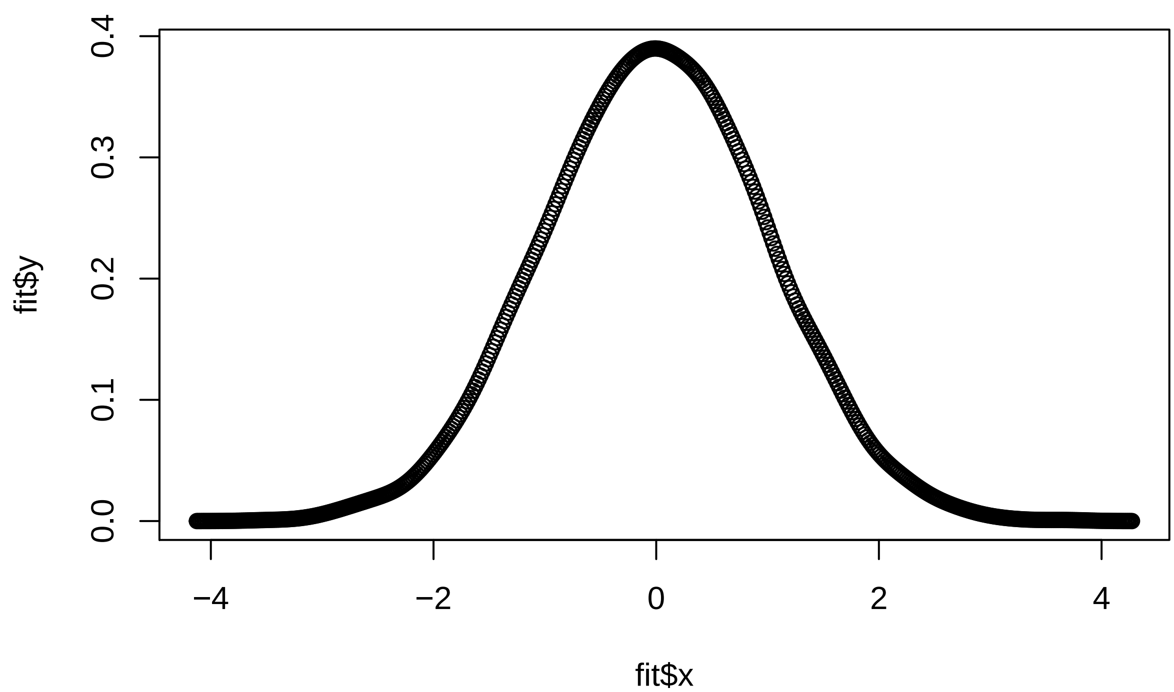
```
y <- rnorm(5000, 0, 1)

# Univariate kernel density estimator
# use bandwidth estimation as recommended in Venables and Ripley (2002)
fit <- density(y, bw = 'sj')

keep.fits <- fit$y
mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
mise
```

```
## [1] 1.566223e-05
```

```
keep.log.mise   <- log10(mise)

# histogram overlay
hist(y, freq = FALSE)
lines(fit)
```




```r
# plot fit over x
plot(x=fit$x, y=fit$y)
```

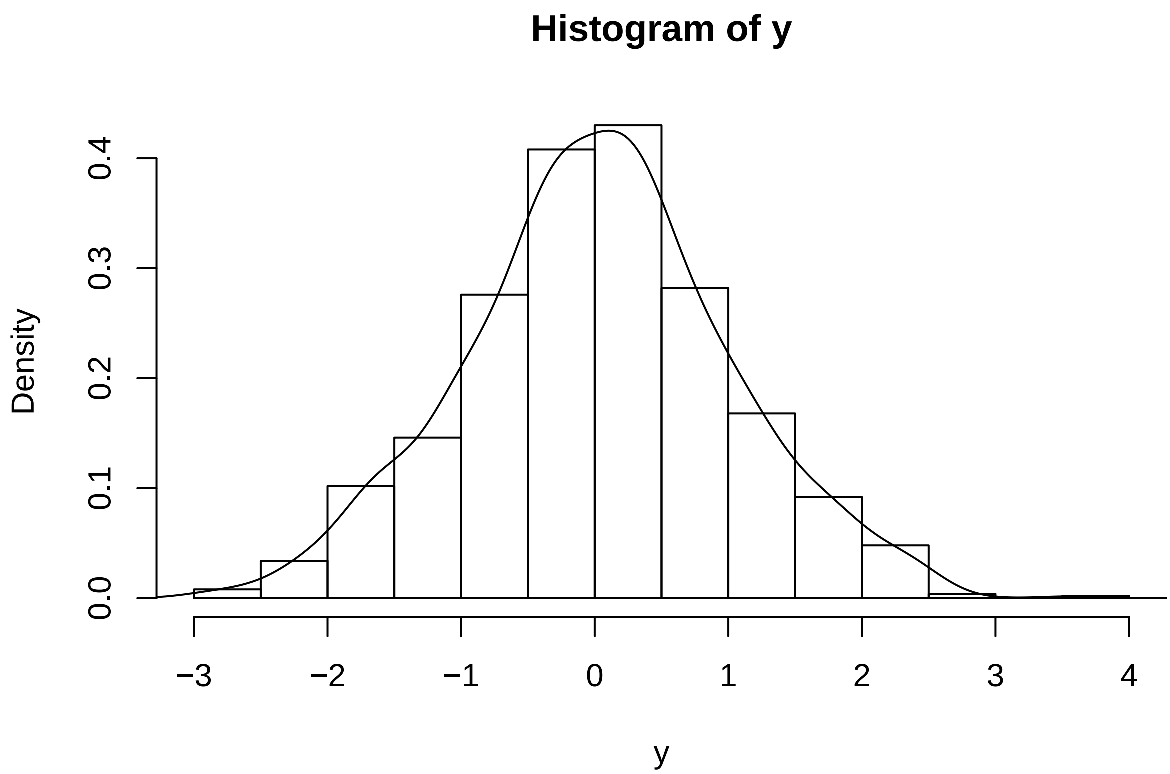


KernSmooth (Wand): Univariate Kernel density estimator test

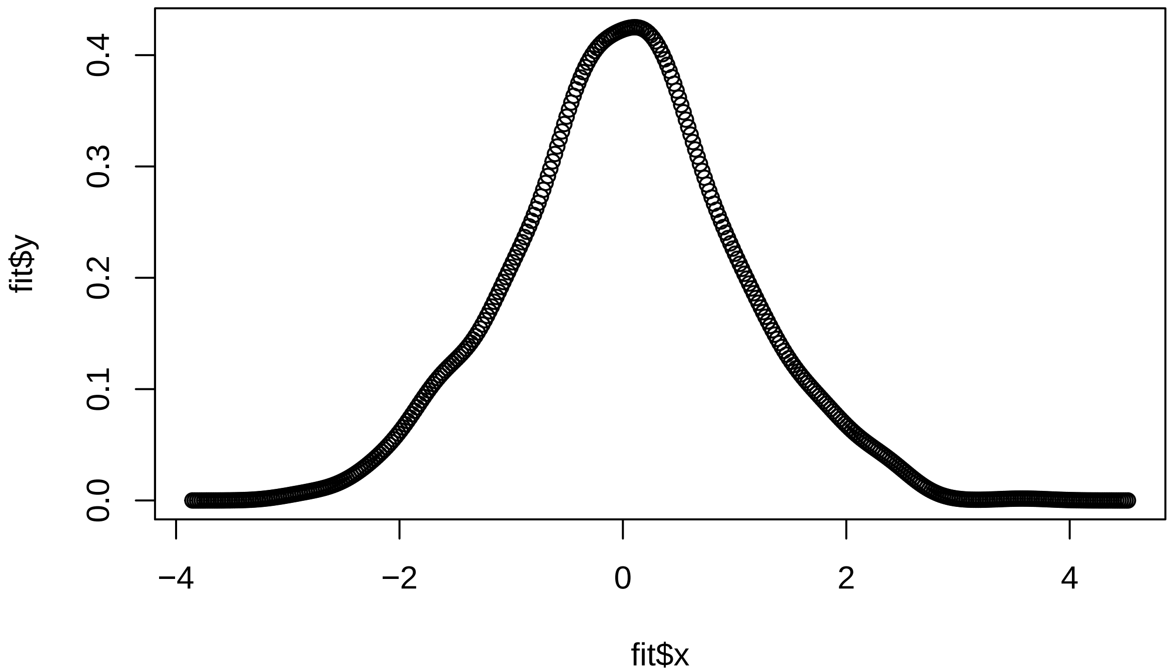
```r
# Univariate kernel density estimator following Wand (1995)
#x <- seq(from=-5,to=5,length=1000)
y <- rnorm(1000, 0, 1)

# select optimal bandwidth
h <- dpik(y)
# kde following Wand (1995)
fit <- bkde(x=y, bandwidth=h)

keep.fits <- fit$y
mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
mise
```

```
## [1] 0.0001844793
```

```r
keep.log.mise   <- log10(mise)

# histogram overlay
hist(y, freq = FALSE)
lines(fit)
```




```r
# plot fit over x
plot(x=fit$x, y=fit$y)
```

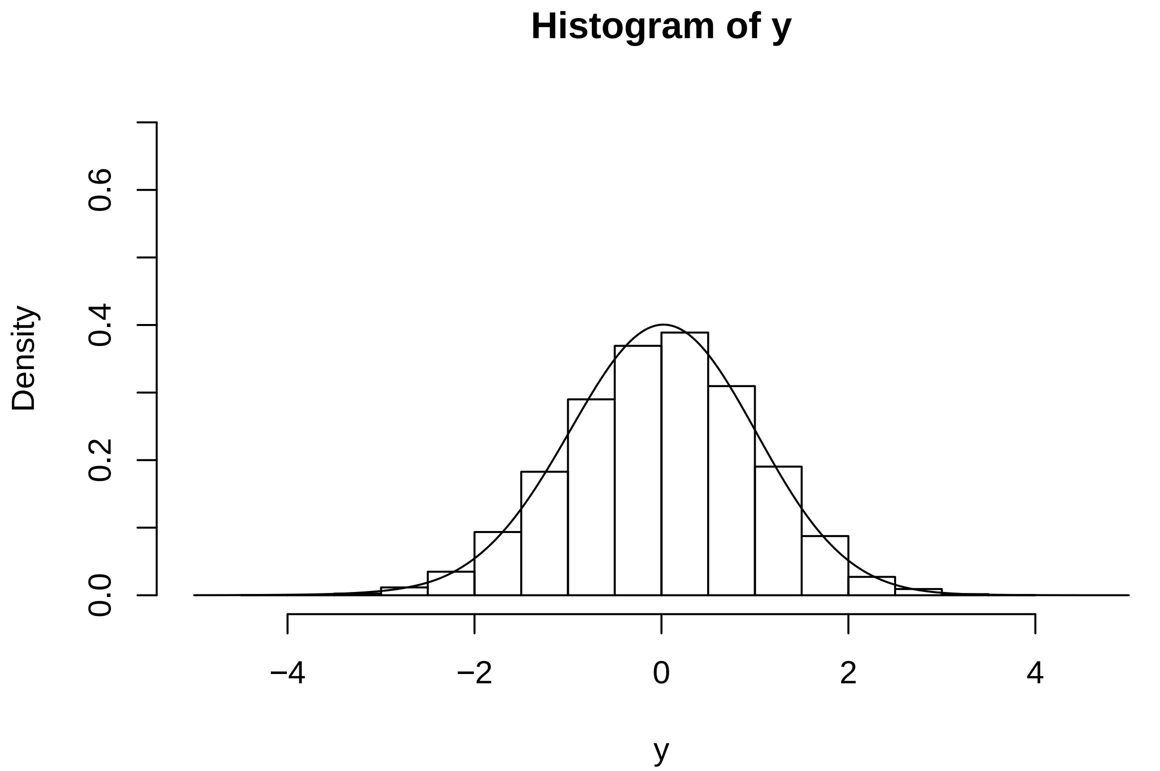


Logspline density estimator test

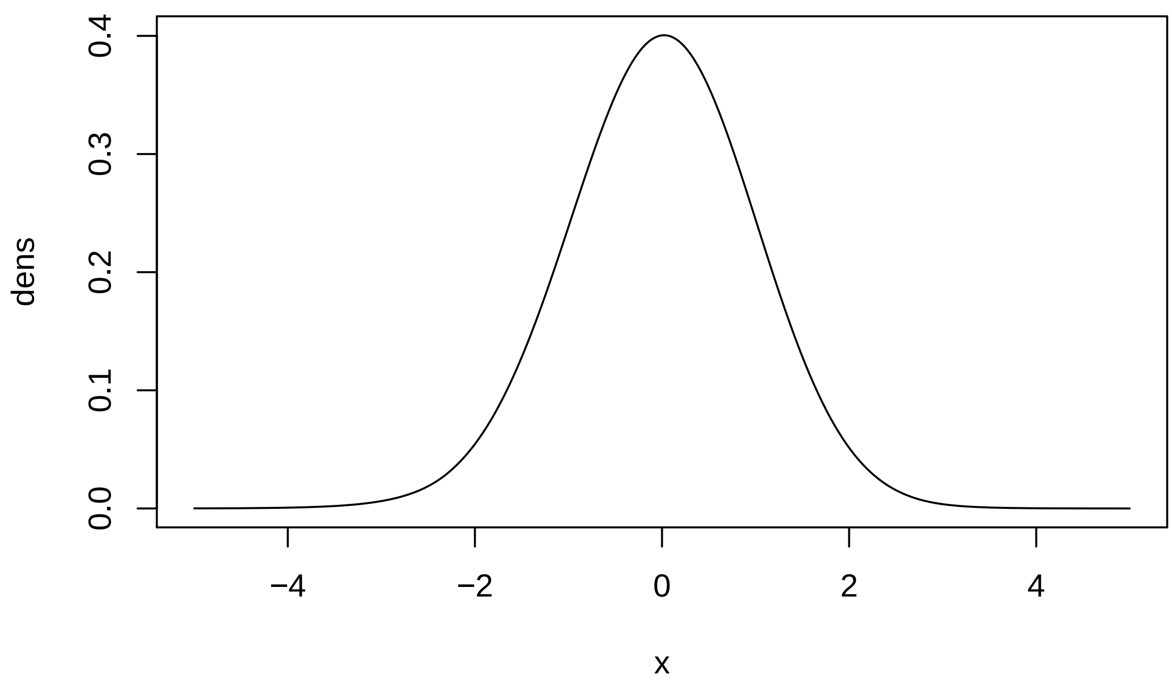
```
y <- rnorm(5000, 0, 1)

# logspline density estimator
fit <- logspline(y)
# summary(fit)
# density object
x = seq(from=-5, to=5, length.out=401)
dens <- dlogspline(q=x, fit=fit)
#summary(dens)

# MISE: mean(dlogspline quantiles - true_quantiles)^2
keep.fits <- dens
mise <- mean((dens-dnorm(x, 0, 1))^2)
mise
```

```
## [1] 3.500502e-06
```

```
keep.log.mise   <- log10(mise)

# histogram overlay
hist(y, freq = FALSE, xlim=c(-5,5), ylim=c(0,0.7))
# plot density of logsplinefit
#plot(fit, n = 101, what = "d")
# density overlay
lines(x, dens, type = "l")
```




```r
# plot dlogspline fit over x
plot(x, dens, type = "l")
```



Monte Carlo experiment

Normal distribution

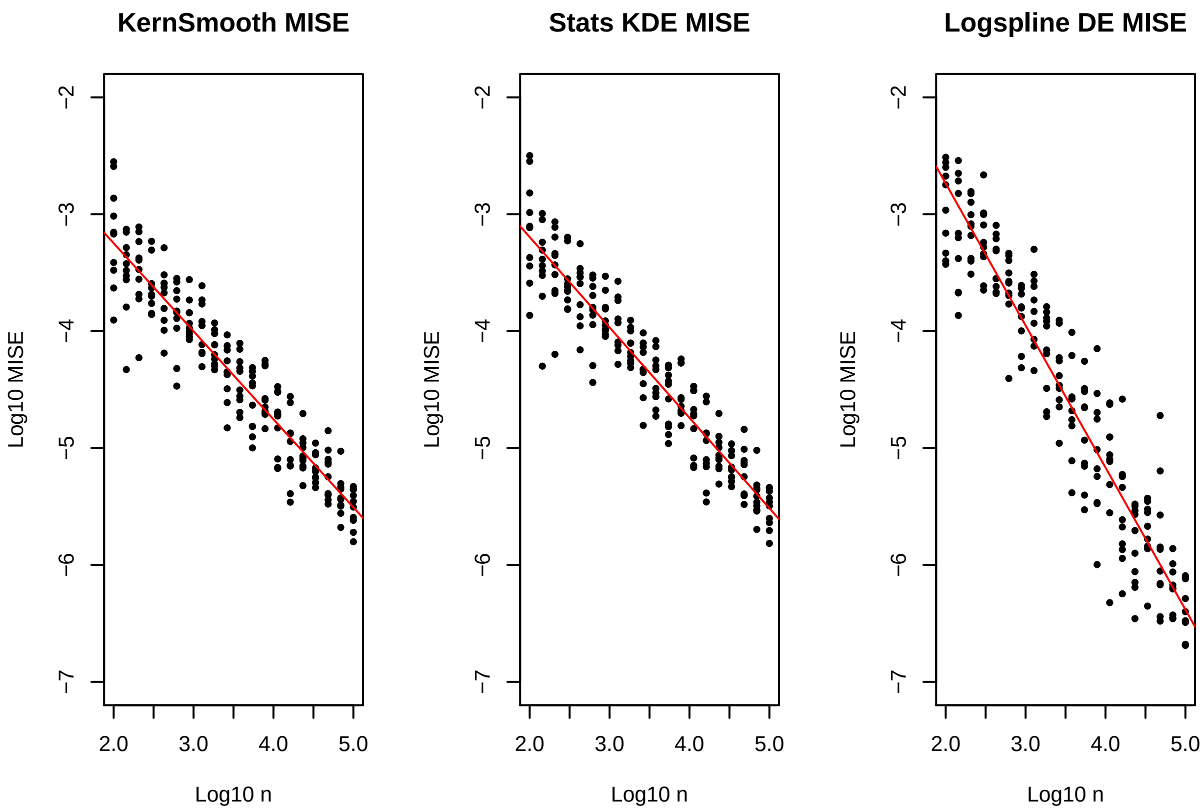
```r
reps.per.n <- 10
log10.ns <- seq(from=2,to=5,length=20) # equally space n's on log scale
ns <- round(10^log10.ns)
log10.ns <- log10(ns)

# make storage for what we want to keep
keep.kernel.wand <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)
keep.kernel <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)
keep.logspline <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)

# let's keep the fits too (it's always useful to look at estimates)
keep.kernel.wand.fits <- matrix(NA,length(keep.kernel$log.n),401)
keep.kernel.fits <- matrix(NA,length(keep.kernel$log.n),401)
keep.logspline.fits <- matrix(NA,length(keep.logspline$log.n),401)


counter <- 1
for (n.i in ns)
{
    for (mc.i in 1:reps.per.n)
    {
        # generate data
      # TODO1: add some noise to distributions?
        y <- rnorm(n.i, 0, 1)

    # Univariate kernel density estimator from KernSmooth package (Wand (1995))
        # select optimal bandwidth
    h <- dpik(y)
    # kde following Wand (1995)
    fit <- bkde(x=y, bandwidth=h, gridsize = 401L)
        # keep fits
        keep.kernel.wand.fits[counter,] <- fit$y
        # calc mse
        mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
        # log mise
        keep.kernel.wand$log.mise[counter]  <- log10(mise)

        # store kde x values for log spline evaluation
        x = fit$x


        # Univariate kernel density estimator from stats package
        # use smoothing parameter (standard deviation of kernel) 'sj', as recommended in Venables and R
        fit <- density(y, bw = 'sj', n=401)
        # keep fits
        keep.kernel.fits[counter,] <- fit$y
        # calc mse
        mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
        # log mise
        keep.kernel$log.mise[counter]   <- log10(mise)


        # Logspline density estimator
```

```
        fit <- logspline(y)
    # density values on 401 equally spaced points
    dens <- dlogspline(q=x, fit=fit)
        # keep fits
        keep.logspline.fits[counter,] <- dens
        # calc mise
        mise <- mean((dens-dnorm(x, 0, 1))^2)
        # log mise
        keep.logspline$log.mise[counter]    <- log10(mise)

        # runtime
        counter <- counter + 1
        if (counter %% 20 == 0){
          print(paste0("Iteration: ", counter))
          }
    }
}
```

```
## [1] "Iteration: 20"
## [1] "Iteration: 40"
## [1] "Iteration: 60"
## [1] "Iteration: 80"
## [1] "Iteration: 100"
## [1] "Iteration: 120"
## [1] "Iteration: 140"
## [1] "Iteration: 160"
## [1] "Iteration: 180"
## [1] "Iteration: 200"
```

Visualization

```
# plot results
par(mfrow=c(1,3))
# kde Wand (1995) plot
plot(keep.kernel.wand$log.n,keep.kernel.wand$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c
points(keep.kernel.wand$log.n,keep.kernel.wand$log.mise,pch=16,cex=.75)
lmfit_kernel.wand <- lm(log.mise~log.n,data=keep.kernel.wand)
abline(lmfit_kernel.wand,col="red")
title("KernSmooth MISE")
# kde plot
plot(keep.kernel$log.n,keep.kernel$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c(-7,-2))
points(keep.kernel$log.n,keep.kernel$log.mise,pch=16,cex=.75)
lmfit_kernel <- lm(log.mise~log.n,data=keep.kernel)
abline(lmfit_kernel,col="red")
title("Stats KDE MISE")
# logspline plot
plot(keep.logspline$log.n,keep.logspline$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c(-7,
points(keep.logspline$log.n,keep.logspline$log.mise,pch=16,cex=.75)
lmfit_logspline <- lm(log.mise~log.n,data=keep.logspline)
abline(lmfit_logspline,col="red")
title("Logspline DE MISE")
```




```r
# look at kernel coefficient of interest
print(summary(lmfit_kernel.wand)$coefficients)
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -1.7413043 0.06903777 -25.22249 9.477074e-64
## log.n       -0.7532185 0.01908953 -39.45715 9.299547e-96
```

```r
print(confint(lmfit_kernel.wand))
```

```
##                  2.5 %     97.5 %
## (Intercept) -1.8774480 -1.6051606
## log.n       -0.7908634 -0.7155737
```

```r
# look at kernel coefficient of interest
print(summary(lmfit_kernel)$coefficients)
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -1.6488327 0.06968400 -23.66157 1.282456e-59
## log.n       -0.7732373 0.01926822 -40.13018 4.730646e-97
```

```r
print(confint(lmfit_kernel))
```

```
##                  2.5 %    97.5 %
## (Intercept) -1.7862507 -1.511415
## log.n       -0.8112345 -0.735240
```

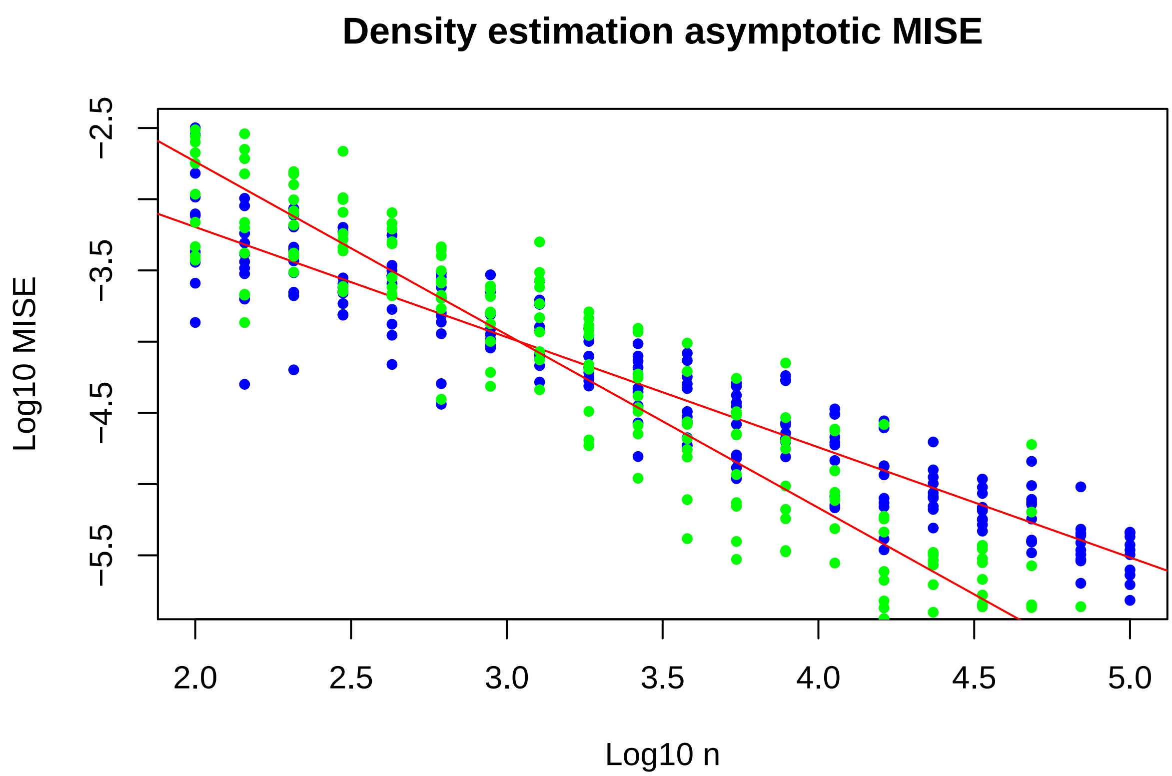
```r
# look at coefficient of interest
print(summary(lmfit_logspline)$coefficients)
```

```
##               Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -0.3063016 0.11115880  -2.755532 6.406512e-03
## log.n       -1.2151070 0.03073636 -39.533214 6.628501e-96
```

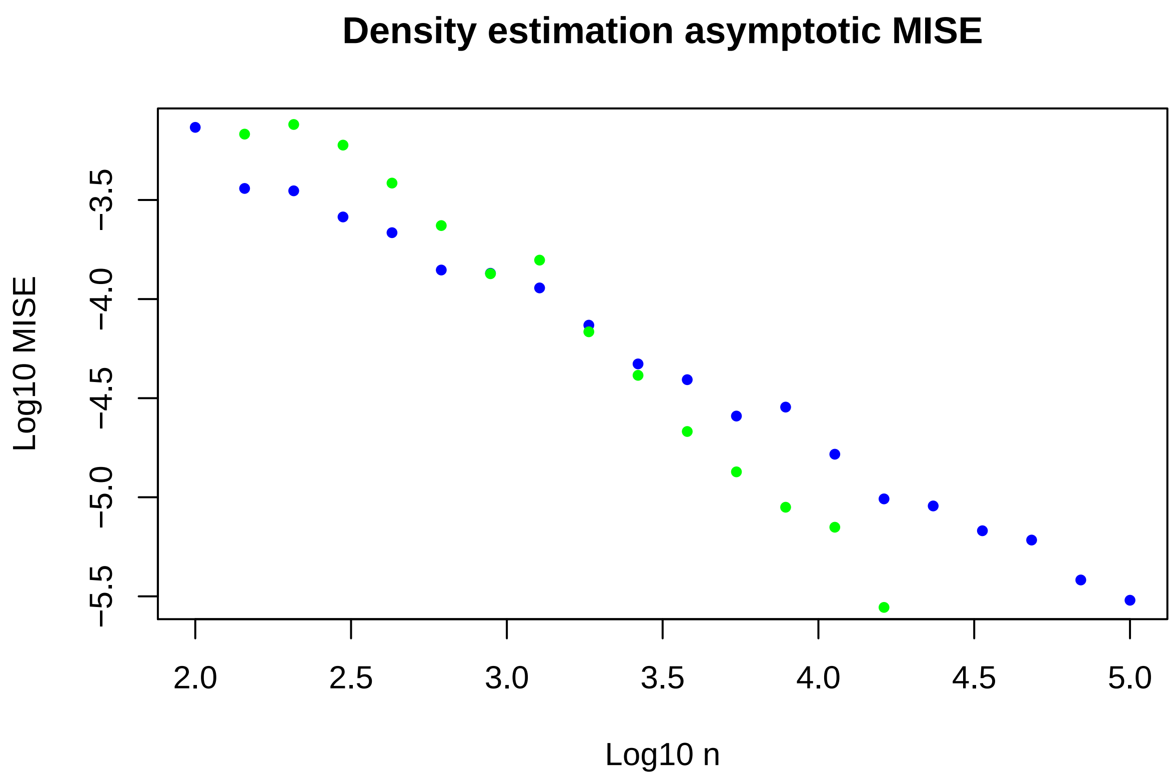
```r
print(confint(lmfit_logspline))
```

```
##                  2.5 %     97.5 %
## (Intercept) -0.5255087 -0.0870945
## log.n       -1.2757196 -1.1544944
```

```r
# plot results
par(mfrow=c(1,1))
# kernel
plot(keep.kernel$log.n,keep.kernel$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n")
points(keep.kernel$log.n,keep.kernel$log.mise,pch=16,cex=.75, col='blue')
lmfit <- lm(log.mise~log.n,data=keep.kernel)
abline(lmfit,col="red")
# logspline
points(keep.logspline$log.n,keep.logspline$log.mise,pch=16,cex=.75, col='green')
lmfit <- lm(log.mise~log.n,data=keep.logspline)
abline(lmfit,col="red")
title("Density estimation asymptotic MISE")
```

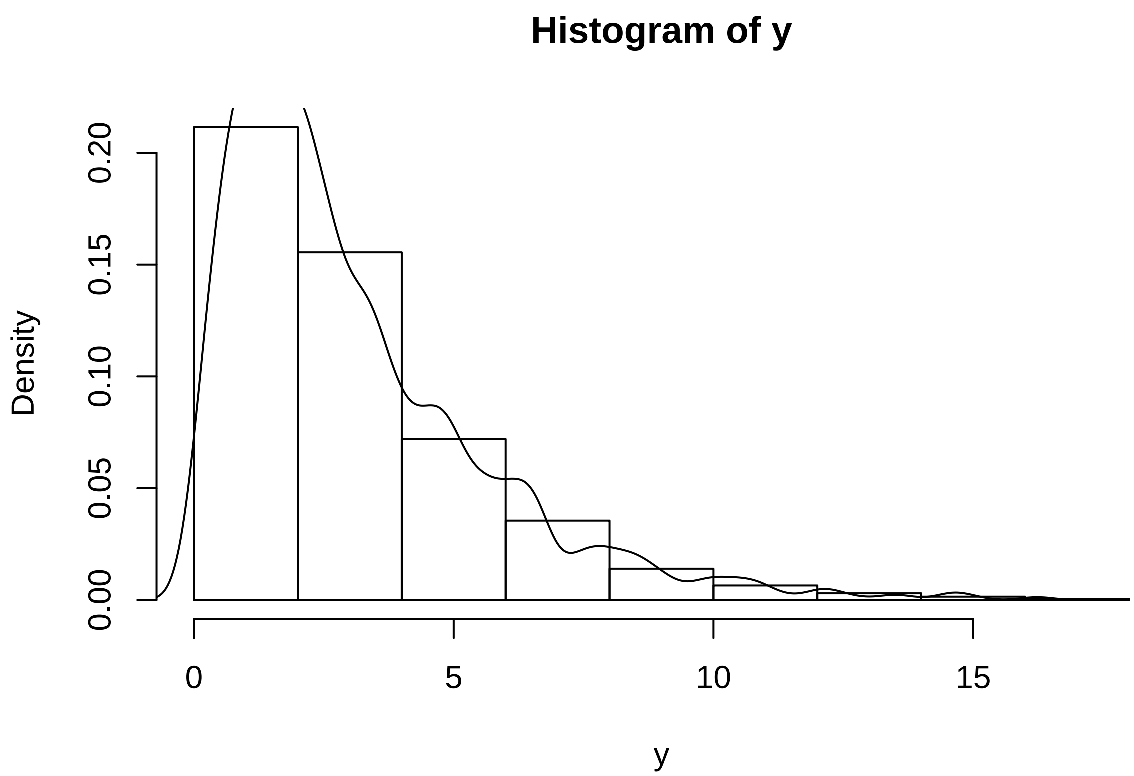



```r
# legend

# mean plots
kernel_mean_mise = aggregate(keep.kernel, by=list(keep.kernel$log.n), mean)
logspline_mean_mise = aggregate(keep.logspline, by=list(keep.logspline$log.n), mean)

plot(kernel_mean_mise$log.n,kernel_mean_mise$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n")
points(kernel_mean_mise$log.n,kernel_mean_mise$log.mise,pch=16,cex=.75, col='blue')
points(logspline_mean_mise$log.n,logspline_mean_mise$log.mise,pch=16,cex=.75, col='green')
title("Density estimation asymptotic MISE")
```

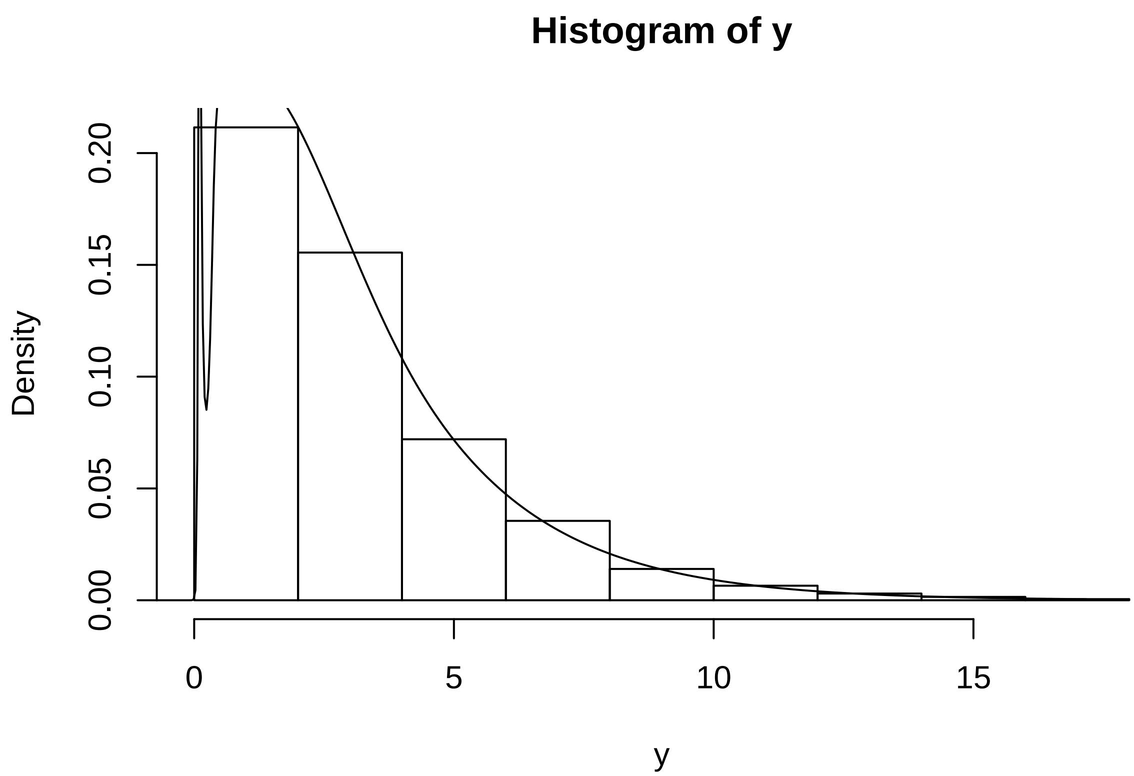


Chi squared distribution

```
y <- rchisq(seq(from=0,to=25,length=1000), df=3)

# Univariate kernel density estimator
# use bandwidth estimation as recommended in Venables and Ripley (2002)
fit <- density(y, bw = 'sj')

# histogram overlay
hist(y, freq = FALSE)
lines(fit)
```




```r
x <- fit$x

# logspline density estimator
fit <- logspline(y)
# summary(fit)
# density object
dens <- dlogspline(q=x, fit=fit)
#summary(dens)

# histogram overlay
hist(y, freq = FALSE)
# plot density of logsplinefit
#plot(fit, n = 101, what = "d")
# density overlay
lines(x, dens, type = "l")
```




```
reps.per.n <- 10
log10.ns <- seq(from=2,to=5,length=20) # equally space n's on log scale
ns <- round(10^log10.ns)
log10.ns <- log10(ns)

# make storage for what we want to keep
keep.kernel.wand <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)
keep.kernel <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)
keep.logspline <- data.frame(log.n = rep(log10.ns,each=reps.per.n),log.mise=NA)

# let's keep the fits too (it's always useful to look at estimates)
keep.kernel.wand.fits <- matrix(NA,length(keep.kernel$log.n),401)
keep.kernel.fits <- matrix(NA,length(keep.kernel$log.n),401)
keep.logspline.fits <- matrix(NA,length(keep.logspline$log.n),401)

counter <- 1
for (n.i in ns)
{
    for (mc.i in 1:reps.per.n)
    {
        # generate data
      # TODO1: add some noise to distributions!
        y <- rchisq(n.i, df=3)

        # Univariate kernel density estimator from KernSmooth package (Wand (1995))
        # select optimal bandwidth
    h <- dpik(y)
    # kde following Wand (1995)
    fit <- bkde(x=y, bandwidth=h, kernel='normal', gridsize = 401L)
```

```
        # keep fits
        keep.kernel.wand.fits[counter,] <- fit$y
        # calc mse
        mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
        # log mise
        keep.kernel.wand$log.mise[counter]  <- log10(mise)

        # store kde x values for log spline evaluation
        x = fit$x


        # Univariate kernel density estimator from stats package
        # use smoothing parameter (standard deviation of kernel) 'sj', as recommended in Venables and R
        fit <- density(y, bw = 'sj', n=401)
        # keep fits
        keep.kernel.fits[counter,] <- fit$y
        # calc mse
        mise <- mean((fit$y-dnorm(fit$x, 0, 1))^2)
        # log mise
        keep.kernel$log.mise[counter]   <- log10(mise)


        # Logspline density estimator
        fit <- logspline(y)
    # density values on 401 equally spaced points
    dens <- dlogspline(q=x, fit=fit)
        # keep fits
        keep.logspline.fits[counter,] <- dens
        # calc mise
        mise <- mean((dens-dchisq(x, df=3))^2)
        # log mise
        keep.logspline$log.mise[counter]    <- log10(mise)

        # runtime
        counter <- counter + 1
        if (counter %% 20 == 0){
          print(paste0("Iteration: ", counter))
          }
    }
}
```

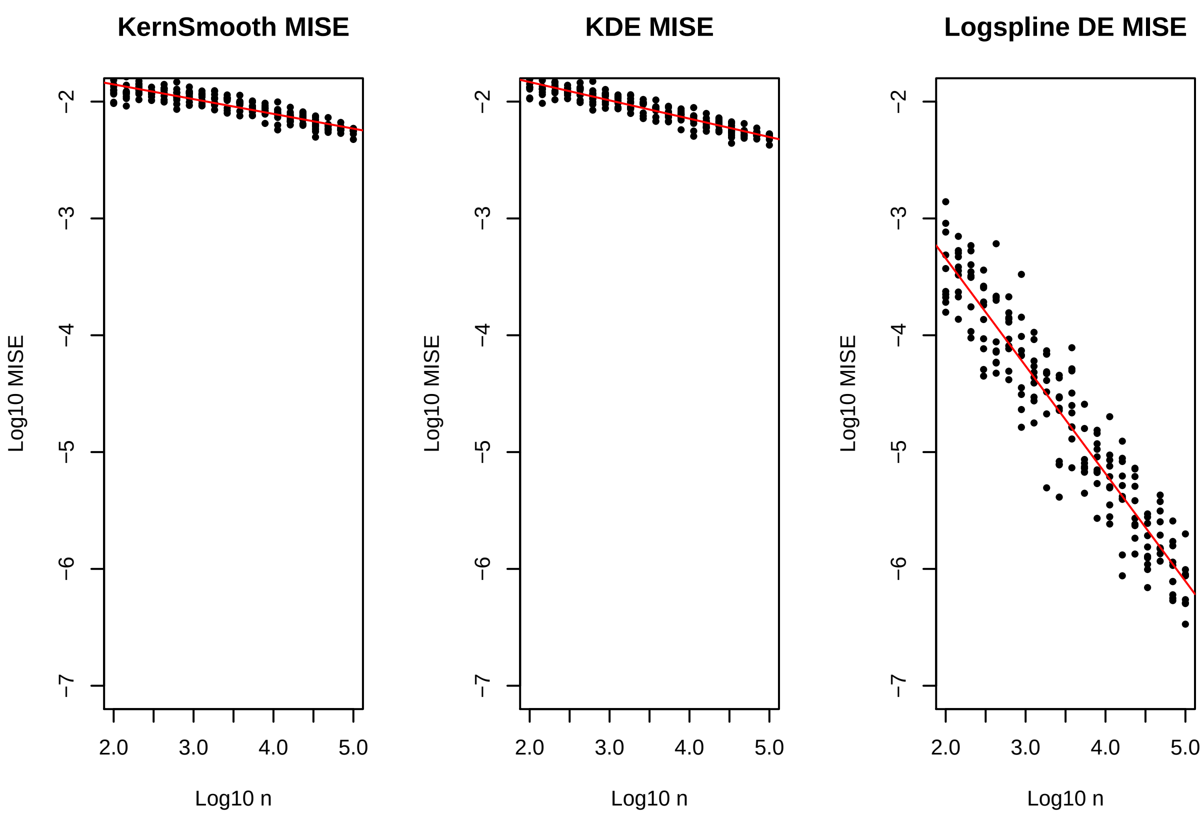
```
## [1] "Iteration: 20"
## [1] "Iteration: 40"
```

```
## Warning in logspline(y): Not all models could be fitted
```

```
## [1] "Iteration: 60"
## [1] "Iteration: 80"
## [1] "Iteration: 100"
## [1] "Iteration: 120"
## [1] "Iteration: 140"
## [1] "Iteration: 160"
## [1] "Iteration: 180"
## [1] "Iteration: 200"
```

Visualization

```
# plot results
# plot results
par(mfrow=c(1,3))
# kde Wand (1995) plot
plot(keep.kernel.wand$log.n,keep.kernel.wand$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c
points(keep.kernel.wand$log.n,keep.kernel.wand$log.mise,pch=16,cex=.75)
lmfit_kernel.wand <- lm(log.mise~log.n,data=keep.kernel.wand)
abline(lmfit_kernel.wand,col="red")
title("KernSmooth MISE")

# stat kde
plot(keep.kernel$log.n,keep.kernel$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c(-7,-2))
points(keep.kernel$log.n,keep.kernel$log.mise,pch=16,cex=.75)
lmfit_kernel <- lm(log.mise~log.n,data=keep.kernel)
abline(lmfit_kernel,col="red")
title("KDE MISE")

# logspline de
plot(keep.logspline$log.n,keep.logspline$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n", ylim=c(-7,
points(keep.logspline$log.n,keep.logspline$log.mise,pch=16,cex=.75)
lmfit_logspline <- lm(log.mise~log.n,data=keep.logspline)
abline(lmfit_logspline,col="red")
title("Logspline DE MISE")
```




```
# look at kernel coefficient of interest
print(summary(lmfit_kernel)$coefficients)
```

```
##               Estimate  Std. Error    t value      Pr(>|t|)
## (Intercept) -1.518808 0.014600015 -104.02786 8.946046e-175
## log.n       -0.156863 0.004037029  -38.85604  1.375058e-94
```

```r
print(confint(lmfit_kernel))
```

```
##                  2.5 %     97.5 %
## (Intercept) -1.5475998 -1.4900168
## log.n       -0.1648241 -0.1489019
```

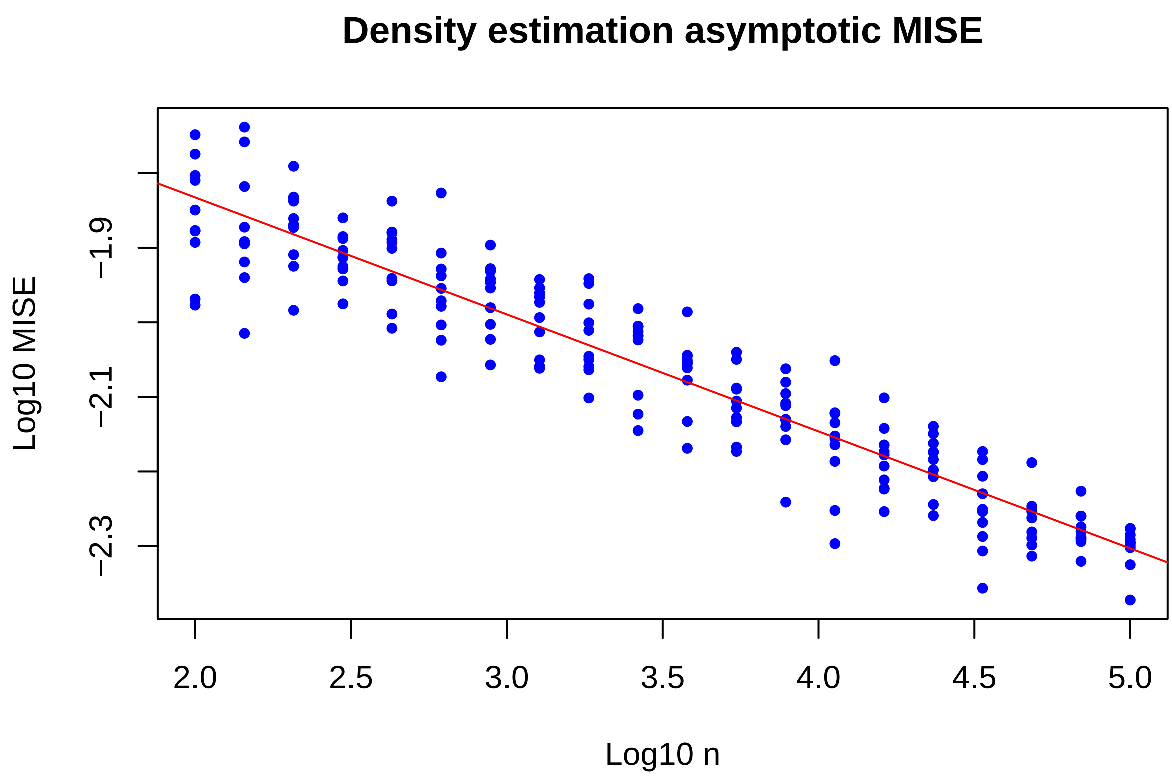
```r
# look at coefficient of interest
print(summary(lmfit_logspline)$coefficients)
```

```
##                Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) -1.4994096 0.07974696 -18.80209  6.365343e-46
## log.n       -0.9211373 0.02205072 -41.77358 3.859145e-100
```

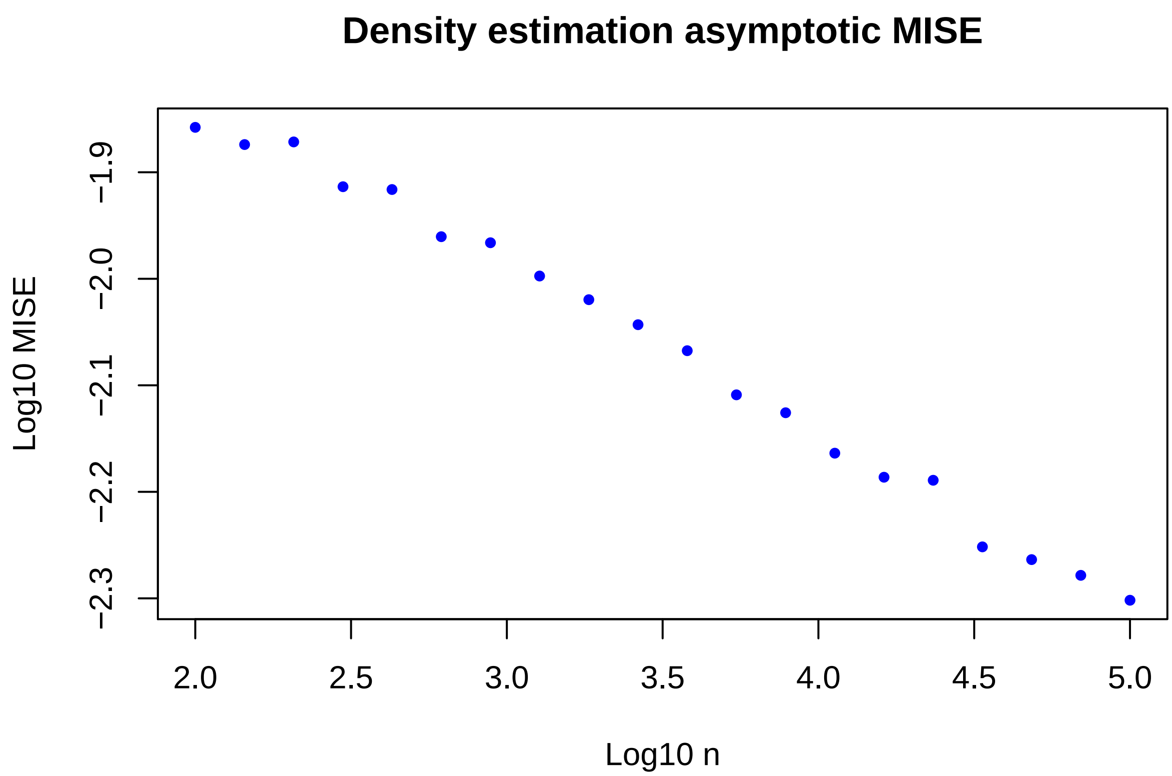
```r
print(confint(lmfit_logspline))
```

```
##                  2.5 %     97.5 %
## (Intercept) -1.6566720 -1.3421472
## log.n       -0.9646217 -0.8776529
```

```r
# plot results
par(mfrow=c(1,1)) # Assumes 20 different sample sizes
# kernel
plot(keep.kernel$log.n,keep.kernel$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n")
points(keep.kernel$log.n,keep.kernel$log.mise,pch=16,cex=.75, col='blue')
lmfit <- lm(log.mise~log.n,data=keep.kernel)
abline(lmfit,col="red")
# logspline
points(keep.logspline$log.n,keep.logspline$log.mise,pch=16,cex=.75, col='green')
lmfit <- lm(log.mise~log.n,data=keep.logspline)
abline(lmfit,col="red")
title("Density estimation asymptotic MISE")
```




```r
# legend

# mean plots
kernel_mean_mise = aggregate(keep.kernel, by=list(keep.kernel$log.n), mean)
logspline_mean_mise = aggregate(keep.logspline, by=list(keep.logspline$log.n), mean)

plot(kernel_mean_mise$log.n,kernel_mean_mise$log.mise,xlab="Log10 n",ylab="Log10 MISE",type="n")
points(kernel_mean_mise$log.n,kernel_mean_mise$log.mise,pch=16,cex=.75, col='blue')
points(logspline_mean_mise$log.n,logspline_mean_mise$log.mise,pch=16,cex=.75, col='green')
title("Density estimation asymptotic MISE")
```



Questions

- Why are convergance rates different (normal vs. chi2)? Maybe dependend on Kernel?

Conclusions

- Is logspline log mise asymptotic behaviour linear?
- Logspline log mise goes faster to zero than kernel density estimation!

```
#par(mfrow=c(4,5),mar=c(1,1,1,1)) # Assumes 20 different sample sizes
#for (log.n.i in unique(keep$log.n))
#{
#   junk <- keep.fits[keep$log.n==log.n.i,]
#   plot(fit$x,f(fit$x),type="n",ylim=range(junk),main=paste("n=",10^log.n.i),
#   xlab="n",ylab="n",axes=F)
#   for (i in 1:dim(junk)[1])
#       lines(fit$x,junk[i,])
#   lines(fit$x,f(fit$x),col="red")
#
#}
```