4 Nonparametric Regression

Let us consider the case of the case of univariate nonparametric regression, i.e., with one single explanatory variable $X \in \mathbb{R}$.

Data: (Y_i, X_i), $i = 1, \ldots, n$, where

- Y_i response variable
- $X_i \in [a, b] \subset \mathbb{R}$ explanatory variable
- n sufficiently large (e.g., $n \geq 40$)

The Nonparametric Regression Model:

$$Y_i = m(X_i) + \epsilon_i$$

- $m(X_i) = \mathbb{E}(Y_i | X = X_i)$ regression function
- $\epsilon_1, \epsilon_2, \ldots$ i.i.d., $\mathbb{E}(\epsilon_i) = 0$, $\mathbb{V}(\epsilon_i) = \sigma^2$
- ϵ_i independent of X_i.

Special cases of **parametric** regression models:

- Linear regression: $m(x)$ is a straight line

$$m(X) = \beta_0 + \beta_1 X$$

- Possible generalizations: $m(x)$ quadratic or cubic polynomial

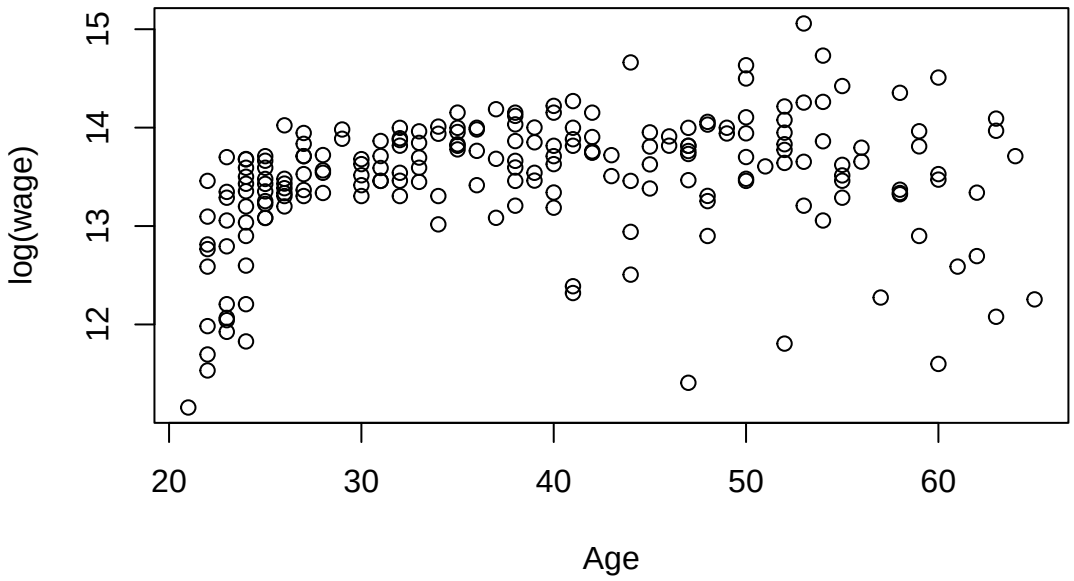
$$\begin{aligned} & m(X) = \beta_0 + \beta_1 X + \beta_2 X^2 \\ \text{or} \quad & m(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \end{aligned}$$

Many important applications lead to regression functions possessing a complicated structure. Standard models then are "too simple" and do not provide useful approximations of $m(x)$

"All models are *false*, but some are useful" (G. Box)

Example: Canadian cross-section wage data consisting of a random sample taken from the 1971 Canadian Census Public Use Tapes for male individuals having common education (grade 13).

```
library("np")
data("cps71")
plot(cps71$age, cps71$logwage, xlab="Age", ylab="log(wage)")
```



Nonparametric Regression:
There are no specific assumptions about the structure of the regression function. It is only assumed that m is *smooth*.

An important point in theoretical analysis is the way how the observations $X_1, \ldots, X_n$ have been generated. One distinguishes between "fixed" and "random design".

Fixed design: The observation points $X_1, \ldots, X_n$ are fixed (non stochastic) values. Example: Crop yield (Y) in dependence of the amount of fertilizer (X) used.

Equidistant Design: (most important special case of fixed design)

$$X_{i+1} - X_i = \frac{b-a}{n}.$$

Random design: The observation points $X_1, \ldots, X_n$ are (realizations of) i.i.d. random variables with density f. The density f is called "design density". Throughout this chapter it will be assumed that $f(x) > 0$ for all $x \in [a, b]$.

Example: Sample $(Y_1, X_1), \ldots, (Y_n, X_n)$ of log-wages (Y) and age (X) of randomly selected individuals.
In the case of random design $m(x)$ is the conditional expectation of Y given $X = x$,

$$m(x) = \mathbb{E}(Y|\ X = x)$$

and $\mathbb{V}(\epsilon_i|X_i) = \sigma^2$. Note: For random design all expectations (as well as variances) have to be interpreted as *conditional* expectations (variances) given $X_1, \ldots, X_n$.

4.1 Basis function expansions

Some frequently used approaches to nonparametric regression rely on expansions of the form

$$m(x) \approx \sum_{j=1}^{p} \beta_j b_j(x),$$

where $b_1(x), b_2(x), \ldots$ are suitable basis functions.

The basis functions $b_1, b_2, \ldots$ have to be chosen in such a way that for *any possible* smooth function m the approximation error

$$\min_{\beta} |m(x) - \sum_{j=1}^{p} \beta_j b_j(x)|$$

tends to zero as $p \to \infty$ (approximation theory).

For a fixed value p an estimator $\hat{m}_p$ is determined by

$$\hat{m}(x) = \sum_{j=1}^{p} \hat{\beta}_j b_j(x),$$

where the coefficients $\hat{\beta}_j$ are obtained by ordinary least squares

$$\sum_{i=1}^{n}\left(Y_i - \sum_{j=1}^{p}\hat{\beta}_j b_j(X_i)\right)^2 = \min_{\beta_1,\ldots,\beta_p}\sum_{i=1}^{n}\left(Y_i - \sum_{j=1}^{p}\beta_j \underbrace{b_j(X_i)}_{X_{ij}}\right)^2$$

Examples are approximations by polynomials, spline functions, wavelets or Fourier expansions (for periodic functions).

4.1.1 Polynomial Regression

Theoretical Justification: Every smooth function can be well approximated by a polynomial of sufficiently high degree (approximation theory).

Approach:

- Choose p and fit a polynomial of degree p:

$$\min_{\beta_1,\ldots,\beta_p}\sum_{i=1}^{n}\left(Y_i - \sum_{j=1}^{p}\beta_j X^{j-1}\right)^2$$

$$\Rightarrow \quad \hat{m}_p(X) = \hat{\beta}_1 + \sum_{j=2}^{p-1}\hat{\beta}_j X_i^{j-1}$$

- This corresponds to an approximation with basis functions

$$b_1(x) = 1, b_2(x) = x, b_3(x) = x^2, \ldots, b_p(x) = x^{p-1}.$$

- Note: It is only assumed that m is well approximated by a polynomial of degree p. That is, there will usually still exist an **approximation error** (i.e., bias $\neq 0$).

R-Code to compute polynomial regressions:
Generate some data:

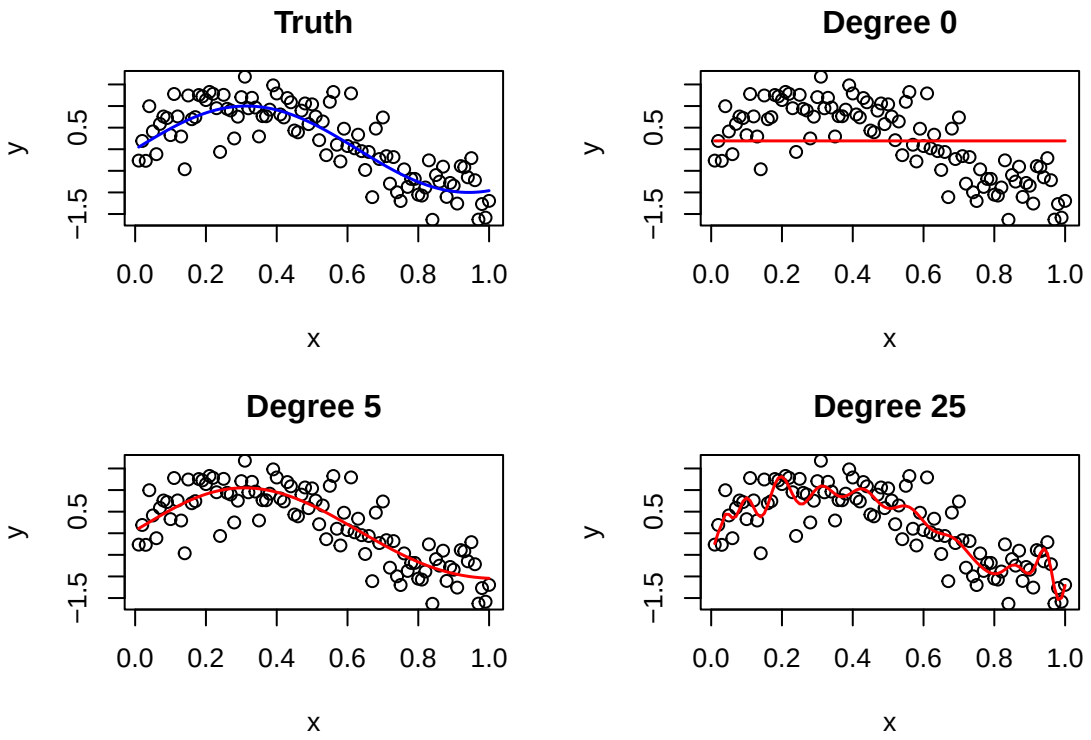
```
set.seed(1)
# Generate some data: ###################################
n      <- 100     # Sample Size
x_vec  <- (1:n)/n # Equidistant X
# Gaussian iid error term
e_vec  <- rnorm(n = n, mean = 0, sd = .5)
# Dependent variable Y
y_vec  <-  sin(x_vec * 5) + e_vec
# Save all in a dataframe
db     <-  data.frame(x=x_vec,y=y_vec)
########################################################
```

Compute the ordinary least squares regressions of different polynomial regression models:

```
# Fitting of polynomials to the data (parametric models):
# Constant line fit: (Basis function x^0)
reg_1 <- lm(y ~ 1, data=db)
# Basis functions: x^0 + ... + x^5
reg_2 <- lm(y ~ poly(x, degree = 5), data=db)
# Basis functions: x^0 + ... + x^25
reg_3 = lm(y ~ poly(x, degree = 25), data=db)
```

Take a look at the fits:

```
par(mfrow=c(2,2), mar=c(4.1,4.1,3.1,2.1))
plot(db, main="Truth")
lines(y=sin(x_vec * 5), x=x_vec, col="blue", lwd=1.5)
##
plot(db, main="Degree 0")
lines(y = predict(reg_1, newdata = db),
      x = x_vec, col="red", lwd=1.5)
plot(db, main="Degree 5")
lines(y = predict(reg_2, newdata = db),
      x = x_vec, col="red", lwd=1.5)
plot(db, main="Degree 25")
lines(y = predict(reg_3, newdata = db),
      x = x_vec, col="red", lwd=1.5)
```



The quality of the approximation obviously depends on the choice of p which serves as a "smoothing parameter"

- p small: variability of the estimator is small, but there may exist a high systematic error (bias).
- p large: bias is small, but variability of the estimator is high.

Remark:
Polynomial regression is not very popular in practice. Reasons are numerical problems in fitting high dimensional polynomials. Furthermore, high order polynomials often posses an erratic, difficult to interpret behavior at the boundaries.

4.1.2 Regression Splines

The practical disadvantages of global basis functions (like polynomials), explain the success of local basis functions. A frequently used system of basis functions are **local polynomials**, i.e., so-called "spline functions".

A spline function is a *piece wise* polynomial function. They are defined with respect to a pre-specified sequence of q "knots"

$$a = \tau_1 < \tau_2 < \cdots < \tau_q = b.$$

Different specifications of the knot sequence lead to different splines.

More precisely, for a given knot sequence a spline function $s(x)$ of degree k is defined by the following properties:

- $s(x)$ is a polynomial of degree k in every interval $[\tau_j, \tau_{j+1}]$, i.e. $s(x) = s_0 + s_1 x + s_2 x^2 + \cdots + s_k x^k$, $s_0, \ldots, s_k \in \mathbb{R}$, for all $x \in [\tau_j, \tau_{j+1}]$.
- $s(x)$ is $k-1$ times continuously differentiable at each knot point $x = \tau_j$, $j = 1, \ldots, q$.

$s(x)$ is called a *linear spline* if $k = 1$, $s(x)$ is a *quadratic spline* if $k = 2$, and $s(x)$ is a *cubic spline* if $k = 3$.

In practice, the most frequently used splines are *cubic* spline functions based on an equidistant sequence of q knots, i.e., $\tau_{j+1} - \tau_j = \tau_j - \tau_{j-1}$ for all j.

The space of all spline functions of degree k defined with respect to a given knot sequence $a = \tau_1, \ldots, \tau_q \leq b$ is a $p := q + k - 1$ dimensional linear function space $\mathcal{S}_{k,\tau_1,\ldots,\tau_q} = \text{span}(b_{1,k}, \ldots, b_{p,k})$.

B-Spline Basis Functions: The so-called B-spline basis functions are almost always used in practice, since they possess a number of advantages from a numerical point of view.

The B-Spline basis functions $b_{j,k}$, $j = 1, \ldots, k + q - 2$, for splines of order k based on a knot sequence $a \leq \tau_1, \ldots, \tau_q \leq b$ are calculated by a recursive procedure:

$$b_{j,0}(x) = \begin{cases} 1 & \text{if } x \in [\tau_j^*, \tau_{j+1}^*] \\ 0 & \text{else} \end{cases}, \quad j = 1, \ldots, q + 2k - 1$$

and

$$b_{j,l}(x) = \frac{x - \tau_j^*}{\tau_{l+j}^* - \tau_j^*} b_{j,l-1}(x) + \frac{\tau_{l+j+1}^* - x}{\tau_{l+j+1}^* - \tau_{j+1}^*} b_{j+1,l-1}(x),$$

for $l = 1, \ldots, k$, $j = 1, \ldots, q + k - 1$, and $x \in [a, b]$. Here, $\tau_1^* = \cdots = \tau_{k+1}^* = \tau_1$, $\tau_{k+2}^* = \tau_2, \ldots, \tau_{k+q}^* = \tau_q$ and $\tau_{k+q+1}^* = \cdots = \tau_{2k+q}^* = \tau_q$.

R-Code to generate B-Spline basis functions:
Generate cubic ($k = 3$) B-spline functions for an equidistant knot sequence with $\tau_1 = 0$, $\tau_2 = 0.25$, $\tau_3 = 0.5$, $\tau_4 = 0.75$, $\tau_5 = 1$.

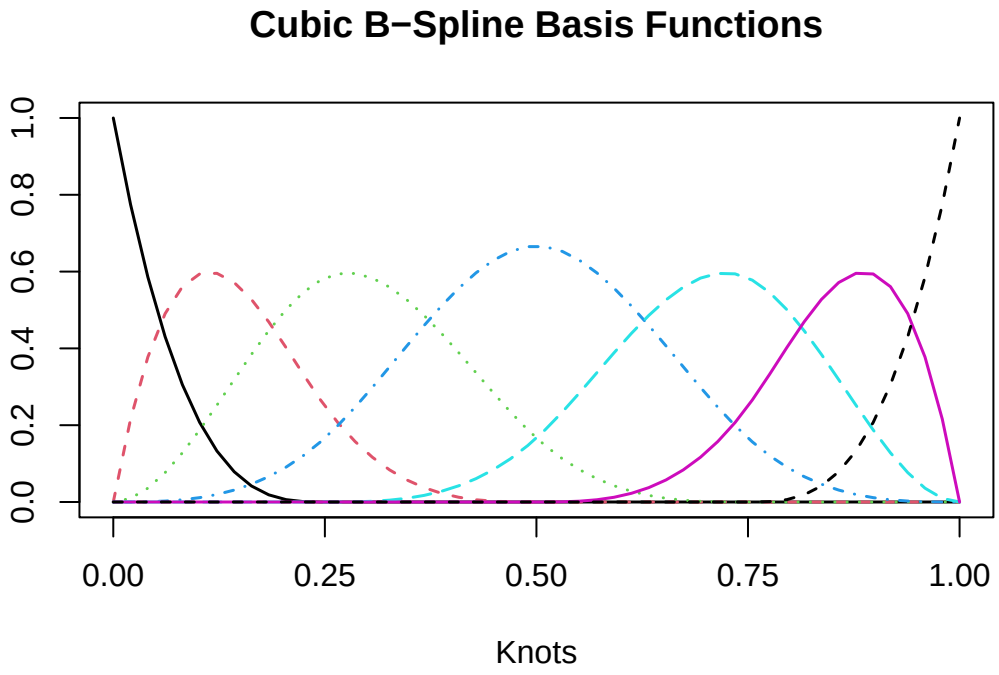
```
suppressMessages(library("fda"))
degree   <- 3 # piecewise cubic splines
knot.seq <- seq(from=0,to=1,len=5)# knots
knot.seq

## [1] 0.00 0.25 0.50 0.75 1.00

cubic.spl <- create.bspline.basis(
             norder = degree + 1, # order=degree+1
             breaks = knot.seq)
```

This leads to

$$p = \underbrace{\texttt{Numbr.of Knots}}_{q=5} + \underbrace{\texttt{degree}}_{k=3} - 1 = 7$$

cubic B-spline basis functions. Let's take a look at them:

```
# evaluation grid
eval_grid     <- seq(from=0,to=1,len=50)
# evaluate the 7 basis functions at eval_grid:
X.basis.mat <- eval.basis(basisobj = cubic.spl,
                          evalarg  = eval_grid)
dim(X.basis.mat)

## [1] 50  7

# plot:
matplot(y=X.basis.mat, x=eval_grid, type="l", lwd=1.5,
        axes=F, ylab="", xlab="Knots",
        main="Cubic B-Spline Basis Functions")
axis(1, at=knot.seq); axis(2); box()
```



Regression Splines:

The so-called "regression spline" (or "B-spline") approach to estimating a regression function $m(x)$ is based on fitting a set of spline basis functions to the data. Frequently, cubic splines ($k = 3$) with equidistant knots are applied. Then $\tau_1 = a, \tau_q = b$ and $\tau_{j+1} - \tau_j = (b-a)/(q-1)$.

In this case the number of knots q (or more precisely the total number of basis functions $p = q + k - 1$ with $k = 3$ in the case of cubic B-splines) serves as the "**smoothing parameter**" which has to be selected by the statistician.

An estimator $\hat{m}_p(x)$ is then given by

$$\hat{m}_p(x) = \sum_{j=1}^{p} \hat{\beta}_j b_{j,k}(x),$$

and the coefficients $\hat{\beta}_j$ are determined by ordinary least squares.

Let $Y = (Y_1, \ldots, Y_n)^\top$ denote the vector of response variables and let $\mathbf{X}$ denote the $n \times p$ matrix with elements $X_{ij} = b_{j,k}(X_i)$.

Then the OLS estimate $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^\top$ can be written as

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top Y.$$

The fitted values are given by

$$\begin{pmatrix} \hat{m}_p(X_1) \\ \vdots \\ \hat{m}_p(X_n) \end{pmatrix} = \mathbf{X}\hat{\beta} = \underbrace{\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top}_{=:S_p} Y$$

The matrix S_p is referred to as the **smoothing matrix** and the number of B-spline basis function p is referred to as the **smoothing parameter**.

Remark: Quite generally, the most important nonparametric regression procedures are "linear smoothing methods". This means that in dependence of some smoothing parameter (here p), estimates of the vector $(m(X_1), \ldots, m(X_n))^\top$ are obtained by multiplying a "smoother matrix" S_p with Y. That is,

$$\begin{pmatrix} m(X_1) \\ \vdots \\ m(X_n) \end{pmatrix} \approx \begin{pmatrix} \hat{m}_p(X_1) \\ \vdots \\ \hat{m}_p(X_n) \end{pmatrix} = S_p Y$$

R-Code to compute regression splines:
Generate some data:

```
set.seed(1)
# Generate some data: ####################################
n      <- 100     # Sample Size
x_vec  <- (1:n)/n # Equidistant X
# Gaussian iid error term
e_vec  <- rnorm(n = n, mean = 0, sd = .5)
# Dependent variable Y
y_vec  <-  sin(x_vec * 5) + e_vec
##########################################################
```

Generate cubic B-spline basis functions with equidistant knot sequence `seq(from=0,to=1,len=15)` and evaluate them at `x_vec`:

```
degree      <- 3 # piecewise cubic splines
knot.seq.5  <- seq(from=0,to=1,len=5)# knots
cubic.spl.7 <- create.bspline.basis(
                norder = degree + 1, # order=degree+1
                breaks = knot.seq.5)
knot.seq.15  <- seq(from=0,to=1,len=15)# knots
cubic.spl.17 <- create.bspline.basis(
                norder = degree + 1, # order=degree+1
                breaks = knot.seq.15)

# evaluate the B-spline basis functions at x_vec:
X.p7  <- eval.basis(basisobj=cubic.spl.7,  evalarg=x_vec)
X.p17 <- eval.basis(basisobj=cubic.spl.17, evalarg=x_vec)
```

Compute the smoothing matrices S_p for $p = 7$ and $p = 17$:

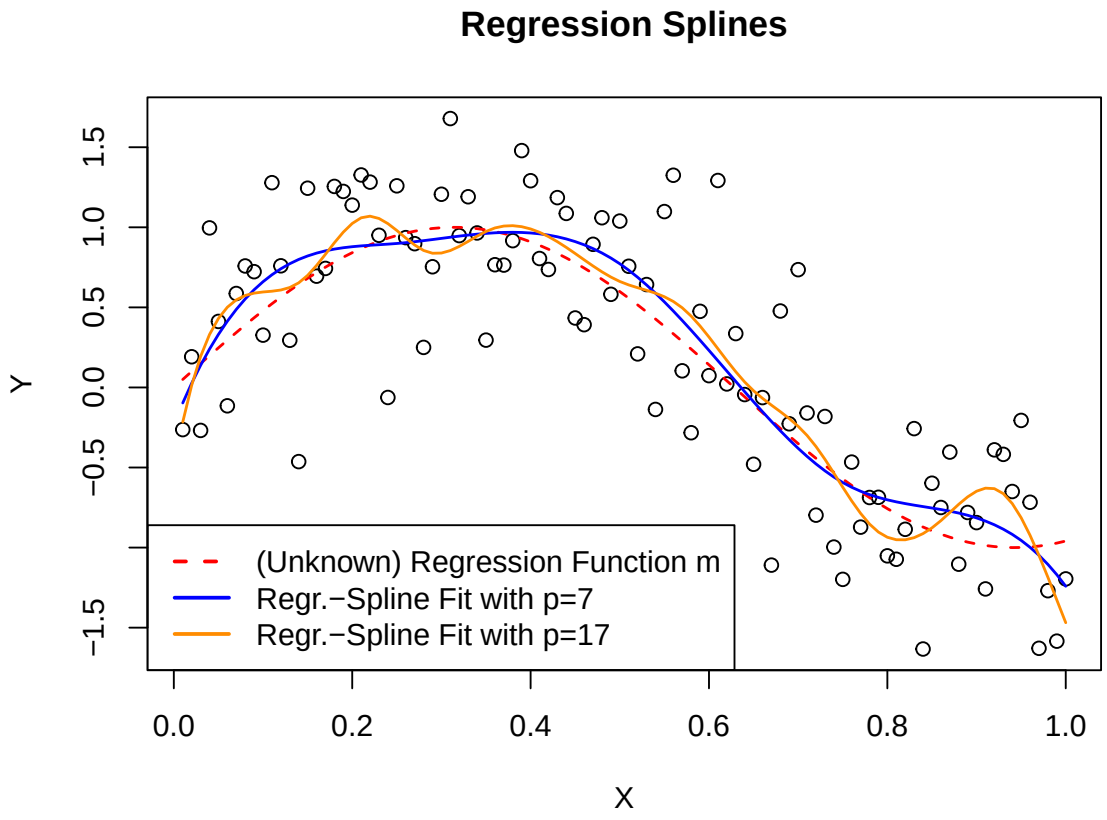
```
S.p7  <- X.p7  %*% solve(t(X.p7)  %*% X.p7)  %*% t(X.p7)
S.p17 <- X.p17 %*% solve(t(X.p17) %*% X.p17) %*% t(X.p17)
```

Compute the estimates $\hat{m}_p(X_1), \ldots, \hat{m}_p(X_n)$ for $p = 7$ and $p = 17$:

```
m.hat.p7  <- S.p7  %*% y_vec
m.hat.p17 <- S.p17 %*% y_vec
```

Let's plot the results:

```
plot(y=y_vec, x=x_vec, xlab="X", ylab="Y",
     main="Regression Splines")
lines(y=sin(x_vec * 5), x=x_vec, col="red", lty=2, lwd=1.5)
lines(y=m.hat.p7, x=x_vec, col="blue", lwd=1.5)
lines(y=m.hat.p17, x=x_vec, col="darkorange", lwd=1.5)
legend("bottomleft",
       c("(Unknown) Regression Function m",
         "Regr.-Spline Fit with p=7",
         "Regr.-Spline Fit with p=17"),
       col=c("red","blue", "darkorange"),
       lty=c(2,1,1), lwd=c(2,2,2))
```



4.1.3 Mean Average Squared Error of Regression Splines

In a nonparametric regression context we do **not** assume that the unknown true regression function $m(x)$ exactly corresponds to a spline function. Thus, $\hat{m}_p = (\hat{m}_p(X_1), \ldots, \hat{m}_p(X_n))^\top$ possesses a systematic estimation error (bias). That is,

$$\mathbb{E}_\epsilon(\hat{m}_p(X_i)) \neq m(X_i).$$

To simplify notation, we will in the following write "$\mathbb{E}_\epsilon$" as well as "$\mathbb{V}_\epsilon$" to denote expectation and variance "with respect to the random variable ϵ, only".

In the case of random design, "$\mathbb{E}_\epsilon$" and "$\mathbb{V}_\epsilon$" thus denote the conditional expectation $\mathbb{E}(\cdot|X_1, \ldots, X_n)$ and variance $\mathbb{V}(\cdot|X_1, \ldots, X_n)$ given the observed X-values. For random design, these conditional expectations depend on the observed sample, and thus are random. For fixed design, such expectations are of course fixed values.

It will always be assumed that the matrix $\mathbf{X}^\top\mathbf{X}$, with $\mathbf{X} = (b_{j,k}(X_i))_{i,j}$, is invertible (under our conditions on the design density this holds with probability 1 for the random design).

The behavior of nonparametric function estimates is usually evaluated with respect to quadratic risk. A commonly used measure of accuracy of a spline estimator $\hat{m}_p$ is the Mean Average Squared Error (MASE):

$$\begin{aligned}
\text{MASE}(\hat{m}_p) &:= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_\epsilon \left(m(X_i) - \hat{m}_p(X_i)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left(m(X_i) - \mathbb{E}_\epsilon(\hat{m}_p(X_i))\right)^2}_{(\text{Bias}_\epsilon(\hat{m}_p(X_i)))^2} + \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{E}_\epsilon\left(\left(\hat{m}_p(X_i) - \mathbb{E}_\epsilon(\hat{m}_p(X_i))\right)^2\right.}_{\mathbb{V}_\epsilon(\hat{m}_p(X_i))}
\end{aligned}$$

Another frequently used measure is the Mean Integrated Squared Error (MISE)

$$\text{MISE}(\hat{m}_p) := \int_a^b \mathbb{E}_\epsilon \left(m(x) - \hat{m}_p(x)\right)^2 dx$$

MASE($\hat{m}_p$) vs. MISE($\hat{m}_p$):

- Equidistant design: MISE($\hat{m}_p$) = MASE($\hat{m}_p$) + $O(n^{-1})$
- MISE and MASE are generally not asymptotically equivalent in the case of random design

$$\text{MASE}(\hat{m}_p) = \int_a^b \mathbb{E}_\epsilon \left(m(x) - \hat{m}_p(x)\right)^2 f(x)dx + O_P(n^{-1}).$$

In the following we focus on the MASE which has the advantage that we can use matrix algebra. For this we have to analyze bias and variance of the estimator $\hat{m}_p(X_i)$.

Let's start with deriving the $\text{Bias}_\epsilon(\hat{m}_p(X_i)) = m_p(X_i) - \mathbb{E}_\epsilon(\hat{m}_p(X_i))$:

$$\begin{aligned}\mathbb{E}_\epsilon(\hat{m}_p(X_i)) &= \mathbb{E}_\epsilon\left(\sum_{j=1}^p \hat{\beta}_j b_{j,k}(X_i)\right) \\ &= \sum_{j=1}^p \mathbb{E}_\epsilon(\hat{\beta}_j) b_{j,k}(X_i),\end{aligned}$$

where $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^\top = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top Y$.

Then

$$\begin{aligned}\mathbb{E}_\epsilon(\hat{\beta}) &= \mathbb{E}_\epsilon\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top(\underbrace{m+\epsilon}_{=Y})\right) \\ &= \underbrace{(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top m}_{=:(\beta_1,\ldots,\beta_p)^\top=\beta} + 0,\end{aligned}$$

where $m = (m(X_1), \ldots, m(X_n))^\top$ and $\epsilon = (\epsilon_1, \ldots, \epsilon_n)^\top$.

Remember that $\beta = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top m$ is a solution of

$$\begin{aligned}\sum_i (m(X_i) - \sum_{j=1}^p \beta_j b_{j,k}(X_i))^2 &= \min_{\vartheta_1,\ldots,\vartheta_p} \sum_i (m(X_i) - \sum_{j=1}^p \vartheta_j b_{j,k}(X_i))^2 \\ &= \min_{s\in\mathcal{S}_{k,\tau_1,\ldots,\tau_q}} \sum_i (m(X_i) - s(X_i))^2.\end{aligned}$$

That is, the mean of our spline regression estimator, i.e.,

$$\mathbb{E}_\epsilon(\hat{m}_p(x)) = \sum_{j=1}^{p} \beta_j b_j(x) =: \tilde{m}_p(x)$$

is the best (L_2) approximation of the true, but unknown, regression function $m(x)$ by means of spline functions in $\mathcal{S}_{k,\tau_1,\ldots,\tau_q}$.

By the general approximation properties of cubic splines ($k = 3$) with $q = p - 2$ equidistant knots, we will thus expect that[1]:

- if m is twice continuously differentiable, then

$$(\text{Bias}(\hat{m}_p))^2 = \frac{1}{n}\sum_{i=1}^{n} (m(X_i) - \tilde{m}_p(X_i)))^2 = O_p(p^{-4})$$

- if m is four times continuously differentiable, then

$$(\text{Bias}(\hat{m}_p))^2 = \frac{1}{n}\sum_{i=1}^{n} (m(X_i) - \tilde{m}_p(X_i)))^2 = O_p(p^{-8})$$

The next step is to compute the (average) **variance** of the estimator, which can be obtained by the usual type of arguments applied in parametric regression:

$$\begin{aligned}
\frac{1}{n}\sum_{i=1} \mathbb{V}(\hat{m}_p(X_i)) &= \frac{1}{n}\,\mathbb{E}_\epsilon\left(\|\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top Y - \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top m\|_2^2\right) \\
&= \frac{1}{n}\,\mathbb{E}_\epsilon\left(\|\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top \epsilon\|_2^2\right) \\
&= \frac{1}{n}\,\mathbb{E}_\epsilon\left(\epsilon^\top(\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top)^\top \;\; \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\epsilon\right) = \\
&= \frac{1}{n}\,\mathbb{E}_\epsilon\left(\epsilon^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\epsilon\right) \\
&= \frac{1}{n}\,\text{trace}\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\,\mathbb{E}_\epsilon(\epsilon\epsilon^\top)\mathbf{X}\right) \quad (\text{with } \mathbb{E}_\epsilon(\epsilon\epsilon^\top) = I_n\sigma_\epsilon) \\
&= \frac{1}{n}\sigma^2\text{trace}\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{X}\right) \\
&= \sigma^2\frac{p}{n} =: \mathbb{V}(\hat{m}_p)
\end{aligned}$$

[1] C. de Boor, "A practical guide to splines" or
R. Eubank, "Spline smoothing and nonparametric regression"

Remark: For any $j \times l$ matrix A and any $l \times j$ matrix B we have the identity

$$\text{trace}(AB) = \text{trace}(BA)$$

Summary: For cubic splines with equidistant knots and a twice differentiable function m we will expect that:

- $(\text{Bias}(\hat{m}_p))^2 = O_p(p^{-4})$
- $\mathbb{V}(\hat{m}_p) = \sigma^2 \frac{p}{n}$

This leads to the classic trade-off between (average) squared bias and (average) variance that is typical for nonparametric statistics:

- $(\text{Bias}(\hat{m}_p))^2$ *decreases* as p increases.
- $\mathbb{V}(\hat{m}_p)$ *increases* as p increases.

An **optimal smoothing parameter** p, balancing bias and variance, will be of order $p_{opt} \sim n^{1/5}$. Then

$$\text{MASE}(\hat{m}_{p_{opt}}) = O_p(n^{-4/5}).$$

Note: For an estimator $\hat{m}$ based on a **valid** (!) parametric model we have

$$\text{MASE}(\hat{m}_{p_{opt}}) = O_p(n^{-1}).$$

Similar results can be obtained for the mean integrated squared error (MISE): If m is twice continuously differentiable, and $p_{opt} \sim n^{1/5}$, then

$$\text{MISE}(\hat{m}_{p_{opt}}) = \mathbb{E}_\epsilon \left(\int_a^b (m(x) - \hat{m}_{p_{opt}}(x))^2 dx \right) = O_p(n^{-4/5}).$$

Problem: Since m is unknown, we cannot directly compute MASE and p_{opt}. However, we need to choose the smoothing parameter p in an (somehow) optimal and objective manner.

Approach: Determine an estimate $\hat{p}_{opt}$ of the "optimal" number p_{opt} of basis functions by minimizing a suitable error criterion with the following properties:

- For every possible p the corresponding criterion function can be calculated from the *data*.
- For any p the error criterion provides "information" about the respective MASE

Recall: With $\hat{m}_p = (\hat{m}_p(X_1), \ldots, \hat{m}_p(X_n))^\top$ we have

$$\hat{m}_p = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top Y = S_p Y$$

and

$$\frac{p}{n} = \frac{\text{trace}(S_p)}{n}.$$

That is, for given p, the number of parameters to estimate by the spline method (one also speaks of the "degrees of freedom" of the smoothing procedure) is equal to p. This corresponds to the trace of the "smoother matrix" $S_p = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$.

Most frequently used error criteria are Cross-Validation (CV) and Generalized Cross-Validation (GCV):

- **Cross-Validation (CV)**: For a given value p, cross-validation tries to approximate the corresponding prediction error.

  $$\text{CV}(p) = \frac{1}{n}\sum_{i=1}^{n}\Big(Y_i - \hat{m}_{p,-i}(X_i)\Big)^2.$$

  Here, for any $i = 1, \ldots, n$, $\hat{m}_{p,-i}$ is the "leave-one-out" estimator of m to be obtained when a spline function is fitted to the $n-1$ observations:

  $$(Y_1, X_1), \ldots, (Y_{i-1}, X_{i-1}), (Y_{i+1}, X_{i+1}), \ldots, (Y_n, X_n).$$

Motivation:
We have

$$\begin{aligned}\mathbb{E}_\epsilon(\mathrm{CV}(p)) =& \frac{1}{n}\,\mathbb{E}_\epsilon\left(\sum_{i=1}^{n}\left(\overbrace{m(X_i)+\epsilon_i}^{=Y_i} - \hat{m}_{p,-i}(X_i)\right)^2\right)\\ =& \underbrace{\frac{1}{n}\,\mathbb{E}_\epsilon\left(\sum_{i=1}^{n}\left(m(X_i)-\hat{m}_{p,-i}(X_i)\right)^2\right)}_{\mathrm{MASE}(\hat{m}_p)}\\ &+ 2\frac{1}{n}\underbrace{\mathbb{E}_\epsilon\left(\sum_{i=1}^{n}(m(X_i)-\hat{m}_{p,-i}(X_i))\epsilon_i\right)}_{=0} + \sigma^2\end{aligned}$$

- **Generalized Cross-Validation (GCV)**:

$$\mathrm{GCV}(p) = \frac{1}{n(1-\frac{p}{n})^2}\sum_{i=1}^{n}\left(Y_i - \hat{m}_p(X_i)\right)^2$$

Motivation:
It is easily verified that with

$$\mathrm{ARSS}(p) := \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{m}_p(X_i)\right)^2$$

we have

$$\mathbb{E}_\epsilon(\mathrm{ARSS}(p)) = \mathrm{MASE}(\hat{m}_p) - 2\sigma^2\frac{p}{n} + \sigma^2$$

If $p \to \infty$ such that $p/n \to 0$, a Taylor expansion yields

$$\mathrm{GCV}(p) = \mathrm{ARSS}(p) + 2\frac{p}{n}\underbrace{\mathrm{ARSS}(p)}_{=\sigma^2+o_p(1)} + O_p\left(\left(\frac{p}{n}\right)^2\right)$$

As motivated above, for large n $\mathrm{CV}(p)$ as well as $\mathrm{GCV}(p)$ can be seen as estimates of $\mathrm{MASE}(\hat{m}_p) + \sigma^2$.

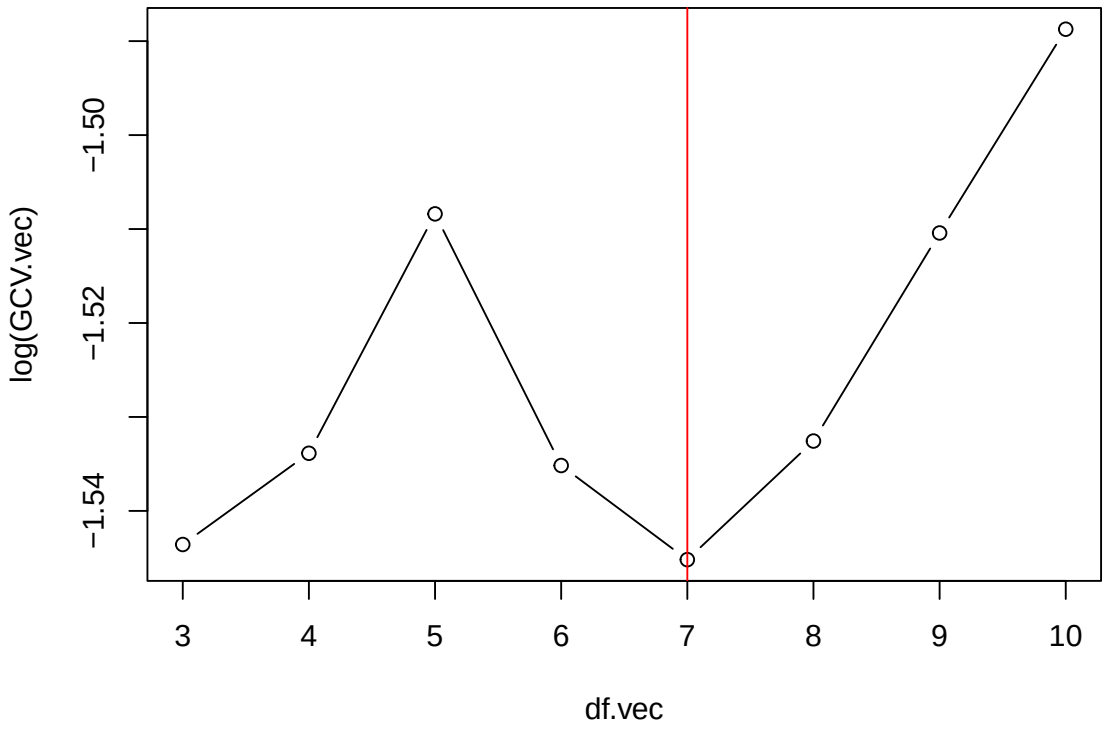
More precisely, as $n \to \infty$, $\frac{p}{n} \to 0$,

$$\mathbb{E}_\epsilon(\mathrm{CV}(p)) = \mathbb{E}_\epsilon(\mathrm{GCV}(p) = \mathrm{MASE}(\hat{m}_p)\cdot(1+o_p(1)) + \sigma^2$$

There are theoretical results which show that if $\hat{p}_{opt}$ is determined by minimizing CV(p) or GCV(p), then for large n MASE($\hat{m}_{\hat{p}_{opt}}$) will be "close" to MASE($\hat{m}_{p_{opt}}$).

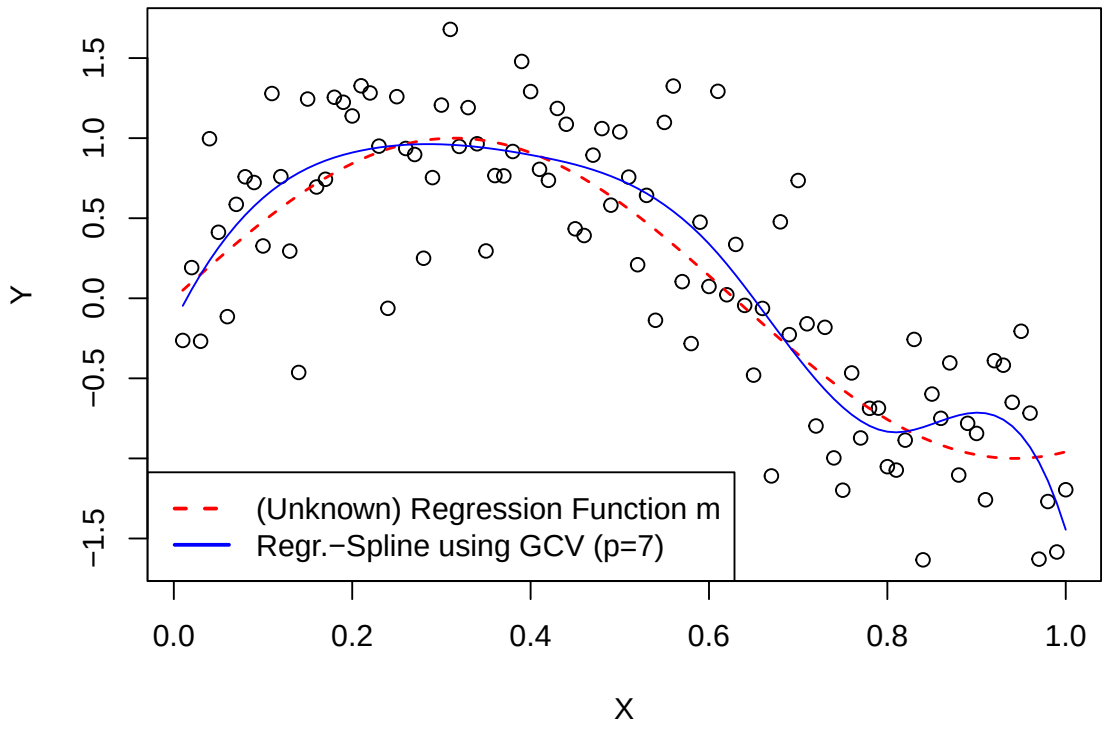
R-Code for computing GCV(p) for regression splines:

```
library("splines")  # B-splines (alternativ to fda-package)
n       <- length(x_vec)
df.vec  <- 3:10
GCV.vec <- rep(NA, length(df.vec))

## GCV:
for(i in 1:length(df.vec)){
  fit <- lm(y_vec ~ bs(x_vec, df = df.vec[i]))

  GCV.vec[i] <- sum((y_vec - predict(fit))^2)/
                    (n*(1 - df.vec[i]/n)^2)
}
plot(y=log(GCV.vec), x=df.vec, type="b")
abline(v=df.vec[which.min(GCV.vec)], col="red")
```



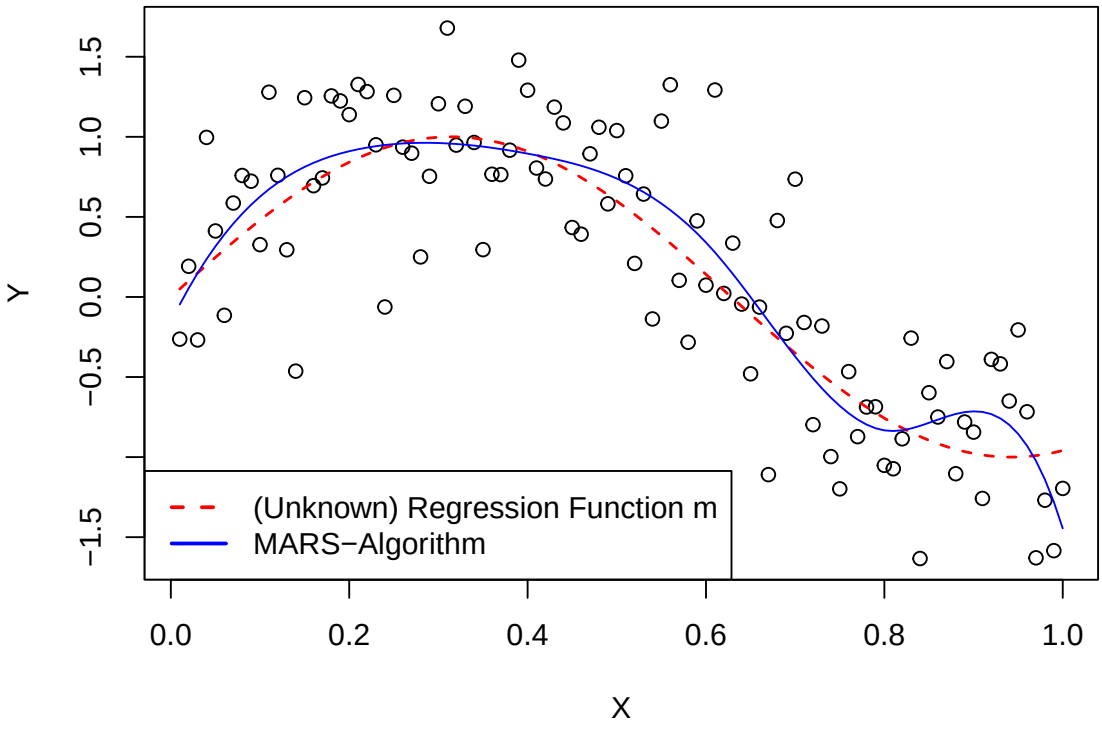
R-Code for computing the GCV-optimal fit:

```
fit <- lm(y_vec ~ bs(x_vec,df=df.vec[which.min(GCV.vec)]))
## plot
plot(y=y_vec, x=x_vec, xlab = "X", ylab = "Y")
lines(y=sin(x_vec * 5), x=x_vec, col="red", lty=2, lwd=1.5)
lines(x=x_vec, y=predict(fit), col="blue")
legend("bottomleft",
       c("(Unknown) Regression Function m",
         "Regr.-Spline using GCV (p=7)"),
       col=c("red","blue"),
       lty=c(2,1), lwd=c(2,2))
```



MARS-Algorithm: There are more advanced procedures which estimate p *as well as a best placement* of the knots $\tau_1, \ldots, \tau_q$ simultaneously from the data (MARS algorithm).

```
## Multivariate Adaptive Regression Splines (MARS)
suppressMessages(library("earth"))

## Error in library("earth"):  there is no package called 'earth'

fit <- earth(formula=y_vec ~ x_vec)

## Error in earth(formula = y_vec ~ x_vec):  could not find function
"earth"

## plot
plot(y=y_vec, x=x_vec, xlab = "X", ylab = "Y")
lines(y=sin(x_vec * 5), x=x_vec, col="red", lty=2, lwd=1.5)
lines(x=x_vec, y=predict(fit), col="blue")
legend("bottomleft",
c("(Unknown) Regression Function m", "MARS-Algorithm"),
col=c("red","blue"), lty=c(2,1), lwd=c(2,2))
```


Y
1.5
1.0
0.5
0.0
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
X
(Unknown) Regression Function m
MARS−Algorithm

4.2 Approaches Based on Roughness Penalties

A different approach to spline fitting, which is widely used in practice, is based on the use of a *roughness penalty*. The basic idea can be described as follows:

- In order to guarantee a small systematic error, spline functions are defined with respect to a large number of knots ($\frac{p}{n}$ close to 1).
- Variability of the estimator is controlled by fitting the coefficients subject to a penalty which penalizes roughness (non-smoothness) of the resulting function. A convenient measure of smoothness is $\int_a^b m''(x)^2 dx$.

Smoothing Splines

The so-called "(cubic) smoothing spline approach" relies on cubic splines with knots at each observation point. More precisely:

- $q = n$ and
- $\tau_1 = X_1, \tau_2 = X_2, \ldots, \tau_n = X_n$.
- The side conditions $s''(a) = 0$ and $s''(b) = 0$ are additionally imposed in order to ensure that the number of coefficients to be estimated is equal to the sample size, i.e., that $p = n$.

In the following we will consider cubic splines ($k = 3$). For a **smoothing parameter** $\lambda > 0$ (to be selected by the statistician), an estimate $\hat{m}_\lambda(x) = \sum_j \hat{\beta}_j b_j(x)$ is determined by

$$\begin{aligned}
&\frac{1}{n}\sum_i (Y_i - \hat{m}_\lambda(X_i))^2 + \lambda \int_a^b \hat{m}''_\lambda(x)^2 dx \\
&= \min_{s \in \mathcal{S}_{3,\tau_1,\ldots,\tau_q}} \left\{ \frac{1}{n}\sum_i (Y_i - s(X_i))^2 + \lambda \int_a^b s''(x)^2 dx \right\},
\end{aligned}$$

or equivalently,

$$\begin{aligned}
&\frac{1}{n}\sum_i (Y_i - \sum_j \hat{\beta}_j b_j(X_i))^2 + \lambda \int_a^b (\sum_j \hat{\beta}_j b''_j(x))^2 dx \\
&= \min_{\vartheta_1,\ldots,\vartheta_p} \left\{ \frac{1}{n}\sum_i (Y_i - \sum_j \vartheta_j b_j(X_i))^2 + \lambda \int_a^b (\sum_j \vartheta_j b''_j(x))^2 dx \right\}.
\end{aligned}$$

Let $\mathbf{X}$ denote the $n \times p$ matrix with elements $b_j(X_i)$, and let $\mathbf{B}$ denote the $p \times p$ matrix with elements $n \int_a^b b_j''(x) b_l''(x) dx$, $j, l = 1, \ldots, p$. Then the solutions are given by

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{B})^{-1} \mathbf{X}^\top Y, \quad \hat{m}_\lambda = \underbrace{\mathbf{X}(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{B})^{-1} \mathbf{X}^\top}_{S_\lambda} Y,$$

where $\hat{m}_\lambda = (\hat{m}_\lambda(X_1), \ldots, \hat{m}_\lambda(X_n))^\top$.

Here only the choice of the **smoothing parameter** λ is crucial for the quality of the estimator. Let's take a look at the effect of λ. First, we need some data:

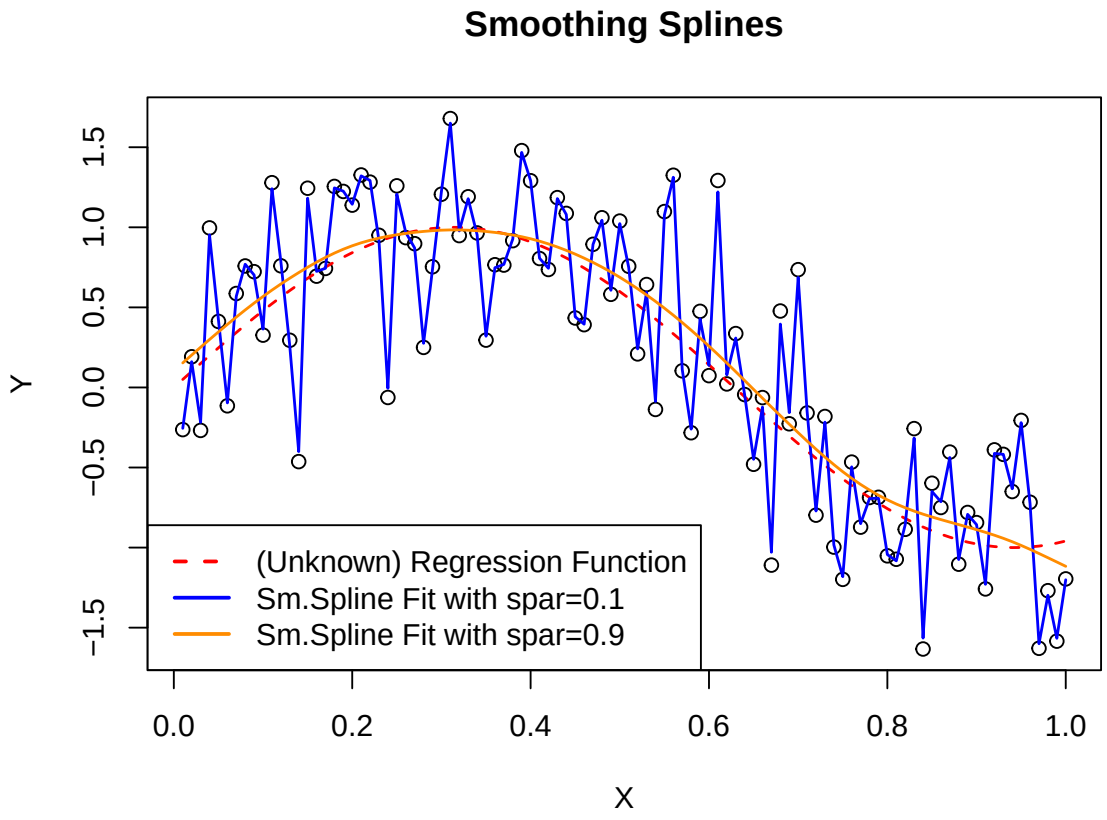
```
set.seed(1)
# Generate some data: ###################################
n      <- 100     # Sample Size
x_vec  <- (1:n)/n # Equidistant X
# Gaussian iid error term
e_vec  <- rnorm(n = n, mean = 0, sd = .5)
# Dependent variable Y
y_vec  <-  sin(x_vec * 5) + e_vec
########################################################
```

Below we use the pre-installed smoothing spline function of R, bu there are many R-packages that contain routines for computing smoothing-splines (see, e.g., `library(pspline)`).

```
lambda.1 <- 0.1
sm.spl.1 <- smooth.spline(x=x_vec, y=y_vec, all.knots=TRUE,
                          spar = lambda.1)
lambda.2 <- 0.9
sm.spl.2 <- smooth.spline(x=x_vec, y=y_vec, all.knots=TRUE,
                          spar = lambda.2)
```

Let's plot the results:

```
plot(y=y_vec, x=x_vec, xlab="X", ylab="Y",
    main="Smoothing Splines")
lines(y=sin(x_vec * 5), x=x_vec, col="red", lty=2, lwd=1.5)
lines(sm.spl.1, col="blue",       lwd=1.5)
lines(sm.spl.2, col="darkorange", lwd=1.5)
legend("bottomleft",
      c("(Unknown) Regression Function",
        "Sm.Spline Fit with spar=0.1",
        "Sm.Spline Fit with spar=0.9"),
      col=c("red","blue", "darkorange"),
      lty=c(2,1,1), lwd=c(2,2,2))
```



Bias, Variance and MASE

The typical bias variance trade-off arises:

- $(\text{Bias}(\hat{m}_\lambda))^2$ *increases* as λ increases.
  Extreme case: $\lambda = \infty \Rightarrow$ straight line fit.
- $\mathbb{V}(\hat{m}_\lambda)$ *decreases* as λ increases. As a consequence, the estimated functions $\hat{m}_\lambda$ become the smoother the larger λ is.

In the following, we will additionally assume that the true regression function m is twice continuously differentiable.

An optimal smoothing parameter λ_{opt} will again balance bias and variance.

- It can be verified that

$$\frac{1}{n}\sum_i \mathbb{V}_\epsilon(\hat{m}_\lambda(X_i)) = \frac{1}{n}\,\mathbb{E}_\epsilon\left(\epsilon^\top S_\lambda^2 \epsilon\right) = \frac{\sigma^2}{n}\,\text{trace}(S_\lambda^2)$$

- And that, as $n \to \infty$, $n\lambda \to \infty$,

$$\text{trace}(S_\lambda^2) = O_p\left(\frac{1}{\lambda^{1/4}}\right)$$

- For a twice continuously differential function m it can be shown that $(\text{Bias}(\hat{m}_\lambda))^2$ is proportional to λ.

Hence, as $n \to \infty$, $\lambda \to 0$, $n\lambda \to \infty$,

$$\text{MASE}(\hat{m}_\lambda) = O_p\left(\lambda + \frac{1}{n\lambda^{1/4}}\right),$$

which implies that an optimal smoothing parameter balancing bias and variance will be of order $\lambda_{opt} \sim n^{-4/5}$. Then

$$\text{MASE}(\hat{m}_{\lambda_{opt}}) = O_p(n^{-4/5}).$$

Again, estimates of λ_{opt} may be determined by minimizing $\text{CV}(\lambda)$ or $\text{GCV}(\lambda)$:

- **Cross-validation (CV)**:

$$\mathrm{CV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\Big(Y_i - \hat{m}_{\lambda,-i}(X_i)\Big)^2,$$

  Here, for any $i = 1, \ldots, n$, $\hat{m}_{p,-i}$ is the "leave-one-out" estimator of m to be obtained when only the $n-1$ observations $(Y_1, X_1), \ldots, (Y_{i-1}, X_{i-1}), (Y_{i+1}, X_{i+1}), \ldots, (Y_n, X_n)$ are used.

- **Generalized cross-validation (GCV)**:

$$\mathrm{GCV}(\lambda) = \frac{1}{n\Big(1 - \frac{df_\lambda}{n}\Big)^2}\sum_{i=1}^{n}\Big(Y_i - \hat{m}_\lambda(X_i)\Big)^2,$$

  where $\mathrm{df}_\lambda := \mathrm{trace}(S_\lambda)$ (= degrees of freedom)

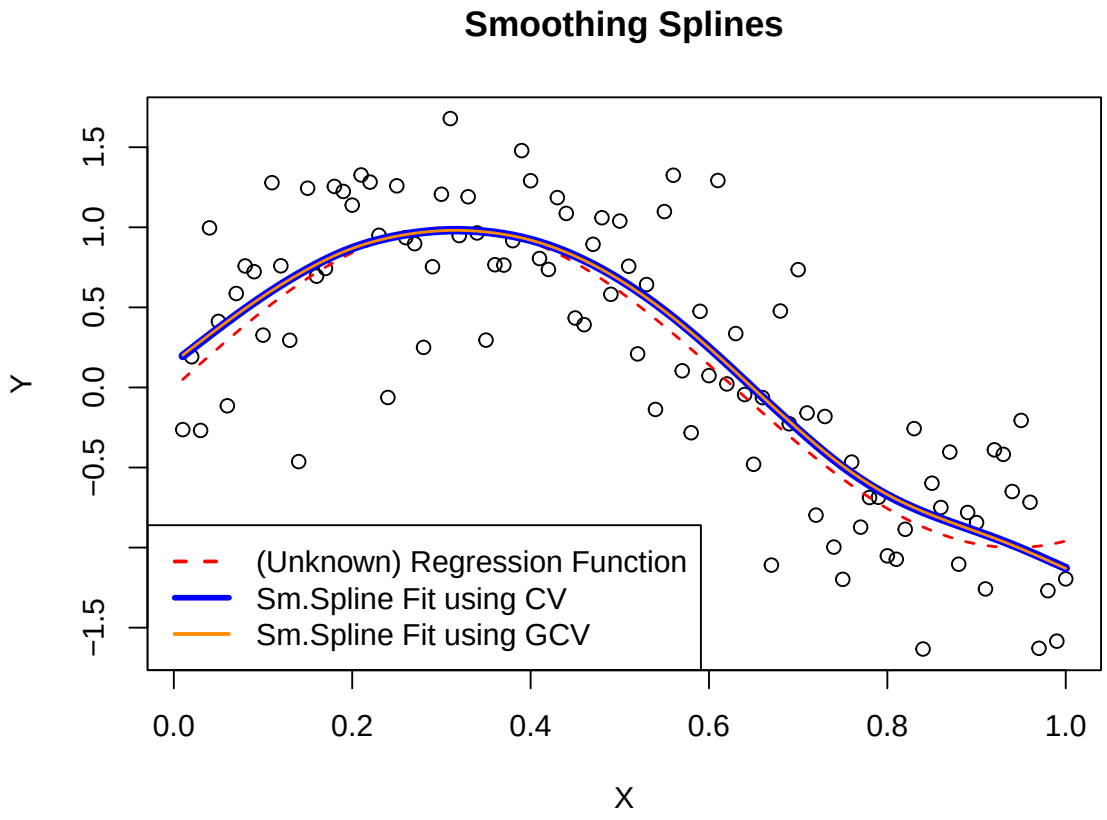
The "degrees of freedoms" of the estimation procedure are defined as $\mathrm{df}_\lambda = \mathrm{trace}(S_\lambda)$ (sometimes also $\mathrm{df}^*_\lambda = \mathrm{trace}(S^2_\lambda)$ is considered). These degrees of freedom can be seen as a nonparametric equivalent of the "number of parameters to estimate" in parametric regression.

The R-function `smooth.spline` has build-in routines for CV and GCV:

```
sm.spl.cv  <- smooth.spline(x=x_vec,y=y_vec,all.knots=TRUE,
                            cv=TRUE)
sm.spl.gcv <- smooth.spline(x=x_vec,y=y_vec,all.knots=TRUE,
                            cv=FALSE)
```

Let's plot the results:

```
plot(y=y_vec, x=x_vec, xlab="X", ylab="Y",
    main="Smoothing Splines")
lines(y=sin(x_vec * 5), x=x_vec, col="red", lty=2, lwd=1.5)
lines(sm.spl.cv,  col="blue",       lwd=4.5)
lines(sm.spl.gcv, col="darkorange", lwd=1.5)
legend("bottomleft",
      c("(Unknown) Regression Function",
        "Sm.Spline Fit using CV",
        "Sm.Spline Fit using GCV"),
      col=c("red","blue", "darkorange"),
      lty=c(2,1,1), lwd=c(2,3,2))
```



4.3 The Nadaraya-Watson Kernel Estimator

General Idea: Approximation of $m(x)$ by a local average of the observations Y_i:

$$\hat{m}_h(x) = \sum_{i=1}^{n} w(x, X_i, h) Y_i$$

- The weight function w is constructed in such a way that the weight of an observation Y_i is the smaller the larger the distance $|x - X_i|$. A smoothing parameter ("bandwidth") h determines the rate of decrease of the weights $w(x, X_i, h)$ as $|x - X_i|$ increases.

Kernel estimators calculate weights on the basis of a pre-specified **kernel function** K. Usually K is chosen as a symmetric density function.

Nadaraya-Watson (NW) kernel estimator:

$$\begin{aligned}\hat{m}_h(x) &= \sum_{i=1}^{n} \frac{K(\frac{x-X_i}{h})}{\sum_{j=1}^{n} K(\frac{x-X_j}{h})} Y_i \\ &= \frac{\frac{1}{nh}\sum_{i=1}^{n} K(\frac{x-X_i}{h}) Y_i}{\frac{1}{nh}\sum_{j=1}^{n} K(\frac{x-X_j}{h})}\end{aligned}$$

Some properties:

- For every possible bandwidth $h > 0$ the sum of all weights

$$w(x, X_i, h) = K\Big(\frac{x - X_i}{h}\Big) / \sum_{j=1}^{n} K\Big(\frac{x - X_j}{h}\Big)$$

  is always equal to 1, i.e., $\sum_i w(x, X_i, h) = 1$.

- Kernel estimators are **linear** smoothing methods:

$$\hat{m}_h = (\hat{m}_h(X_1), \ldots, \hat{m}_h(X_n))^T = S_h Y,$$

  where the elements of the $n \times n$ matrix S_h are given by

$$(S_h)_{ij} = \frac{K\Big(\frac{X_i - X_j}{h}\Big)}{\sum_{l=1}^{n} K\Big(\frac{X_i - X_l}{h}\Big)}, \quad \text{and where} \quad \text{trace}(S_h) = O\left(\frac{1}{h}\right).$$

Similar to density estimation, usually second-order kernel functions are used in practice. The most important examples are:

- Epanechnikov kernel
- Gaussian kernel
- Biweight (quartic) kernel

R-Code for computing the NW-Estimator:
First generate some data.

```
# Generate some data: ################################
# Equidistant X
n      <- 100
x_vec  <- (1:n)/n
# Gaussian iid error term
e_vec  <- rnorm(n = n, mean = 0, sd = .5)
# Dependent variable Y
y_vec  <-  sin(x_vec * 5) + e_vec
# Save all in a dataframe
db     <-  data.frame(x=x_vec,y=y_vec)
```

Choose a kernel function (e.g., the Epanechnikov kernel) and write a function for the weights $w(x, X_i, h)$:

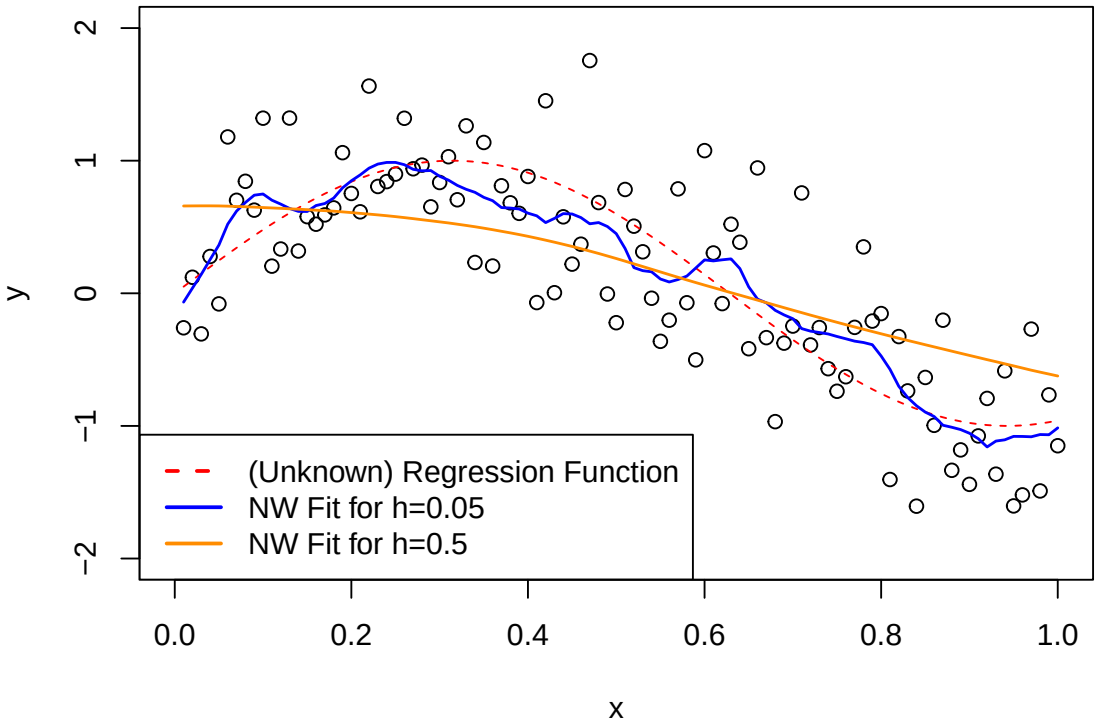
```
# Epanechnikov kernel function
epanech_kern_f   <- function(u){
   ifelse(abs(u)<=1,(3/4)*(1-u^2), 0)}

# Function to compute the kernel-weights:
w_fun <- function(x0, x_vec, h){
   u     <- (x_vec - x0)/h
   epa_w <- epanech_kern_f(u=u)
   return(epa_w)
}
```

Select a bandwidth h, compute the smoothing matrix S_h, and the fitted values $S_h Y$:

```
# Choose a bandwidth:
h.1   <- 0.05
h.2   <- 0.5
# Compute the Smoothing-Matrix S_h:
S_h_mat.1 <- matrix(NA, nrow = n, ncol = n)
S_h_mat.2 <- matrix(NA, nrow = n, ncol = n)
for(i in 1:n){
S_h_mat.1[i,] <- w_fun(x0=x_vec[i], x_vec=x_vec, h=h.1)/
                 sum(w_fun(x0=x_vec[i], x_vec=x_vec, h=h.1))
S_h_mat.2[i,] <- w_fun(x0=x_vec[i], x_vec=x_vec, h=h.2)/
                 sum(w_fun(x0=x_vec[i], x_vec=x_vec, h=h.2))
}
# Compute the fitted values
m_hat.1 <- S_h_mat.1 %*% y_vec
m_hat.2 <- S_h_mat.2 %*% y_vec
```

Let's take a look at the result:

```
# Data points:
plot(db, ylim=c(-2,2), xlim=c(0,1))
# True regression function m(x):
lines(x=x_vec, y=sin(x_vec * 5), col="red", lty=2, lwd=1)
# Plotting the fitted values
lines(y= m_hat.1, x = x_vec, col="blue",       lwd=1.5)
lines(y= m_hat.2, x = x_vec, col="darkorange", lwd=1.5)
legend("bottomleft",
       c("(Unknown) Regression Function",
         "NW Fit for h=0.05",
         "NW Fit for h=0.5"),
       col=c("red","blue", "darkorange"),
       lty=c(2,1,1), lwd=c(2,2,2))
```


2
1
0
−1
−2
y
0.0
0.2
0.4
0.6
0.8
1.0
x
(Unknown) Regression Function
NW Fit for h=0.05
NW Fit for h=0.5

MSE and Optimal Bandwidth for Equidistant Design

Assumptions:

- Equidistant Design on $[a, b] = [0, 1]$. I.e., with $X_{i+1} - X_i = 1/n$
- m twice continuously differentiable.
- Second order kernel with compact support $[-1, 1]$.
- $n \to \infty$, $h \to 0$, $nh \to \infty$

This is the simplest possible situation. Then for any x in the interior of $[0, 1]$

$$\frac{1}{nh} \sum_{j=1}^{n} K\Big(\frac{x - X_j}{h}\Big) \to 1,$$

and

$$\frac{1}{nh} \sum_{j=1}^{n} K\Big(\frac{x - X_j}{h}\Big) m(X_i) = \int_{-1}^{1} K\Big(\frac{x - u}{h}\Big) m(u) du + o\left(\frac{1}{n}\right)$$

Straightforward Taylor expansions then yield

- Local bias (x in the interior of $[0, 1]$):

  $$\tilde{m}_h(x) = \mathbb{E}_\epsilon\left(\hat{m}_h(x)\right) = m(x) + \frac{1}{2} h^2 \, m''(x) \nu_2(K) + o(h^2)$$

  $$\Rightarrow (\text{Bias}(\hat{m}_h(x)))^2 = (m(x) - \tilde{m}_h(x))^2 = \frac{1}{4} h^4 m''(x)^2 \nu_2(K)^2 + o(h^4),$$

  where $\nu_2(K) = \int_{-\infty}^{\infty} K(z) z^2 dz$.

- Local variance (x in the interior of $[0, 1]$):

  $$\mathbb{V}_\epsilon(\hat{m}_h(x)) = \frac{\sigma^2}{nh} R(K) + o(\frac{1}{nh}),$$

  where $R(K) = \int_{-\infty}^{\infty} K(z)^2 dz$.

The local mean squared error is then given by:

$$\begin{aligned} &\text{MSE}(\hat{m}_h(x)) = (\text{Bias}_\epsilon(\hat{m}_h(x)))^2 + \mathbb{V}_\epsilon(\hat{m}_h(x)) \\ &= \tfrac{1}{4} h^4 m''(x)^2 \nu_2(K)^2 + \tfrac{\sigma^2}{nh} R(K) + o(h^4 + \tfrac{1}{nh}) \end{aligned}$$

The optimal (local) bandwidth balancing bias and variance:

$$h_{opt,x} = n^{-1/5}\left(\frac{R(K)}{m''(x)^2 2\nu_2(K)^2}\right)^{1/5}$$

This implies $\text{MSE}(\hat{m}_{h_{opt}}(x)) = O(n^{-4/5})$.

NW-Estimator as a Local Polynomial (degree=0) Regression

The NW-Estimator can be seen as a **locally** weighted least squares estimate with only an intercept as regressor. That is, we can simply use

$$\texttt{lm(y} \sim \texttt{1, weights=w)}$$

with local weights $\texttt{w} = w(x, X_i, h)$ in order to compute the NW-estimator. But this is then nothing else then performing polynomial regression (with degree zero) that is using only a weighted subset of the total data. I.e., a local polynomal regression (with degree zero).

The following R-Code visualizes this:

```
# Define the point at which m(x) shall be estimated:
x0       <- 0.2
# Choose a bandwidth:
h        <- 0.1
# Compute the local weights:
w        <- w_fun(x0 = x0, x_vec = x_vec, h = h)
# Center the X-data around x:
db_x0    <- data.frame("y"=y_vec, "x"=x_vec-x0)
# Computing the Nadaraya-Watson estimate
# (= Fitting a local constant):
NW_estim <- lm(y ~ 1, data=db_x0, weights=w)
# Save the NW estimate
# (=estimate of intercept-parameter)
y0_NW    <- coef(NW_estim)
```

R-Code to visualize the NW-Estimator as a *locally* weighted least squares estimator:

```
# Plot: ##################################
plot(db, cex=abs(w)*4, type="n")
# light-gray background
rect(par("usr")[1], par("usr")[3], par("usr")[2],
     par("usr")[4], col = gray(.80), border=gray(.80))
# plotting All data pairs (Y_i, X_i)
points(db, pch=21, cex=.3, bg=gray(.5),
       col=gray(.5), lwd=1)
# dark-gray background for the interval [x0-h, x0+h]
```

```r
rect(x0-h, par("usr")[3], x0+h, par("usr")[4],
     col = gray(.70), border=gray(.70))
# re-draw the outer box of the plot
box()
# plotting the data pairs (Y_i, X_i),
# plus visualization of the weigths
points(db, pch=21, cex=abs(w)*1.8, bg=gray(0),
       col=gray(0), lwd=1)
# Estimated local constant regression line:
lines(x=c(x0-h, x0+h), y=c(y0_NW,y0_NW), col="red")
# True regression function m(x):
lines(x=x_vec, y=sin(x_vec * 5), col="red", lty=2)
# Estimated point: \hat{m}_h(x0)
points(y = y0_NW, x = x0, pch=21, bg="red", col ="red")
abline(v=x0, lty=2, lwd=.8)
```