Spline regression and additive models (with C++)

BBVA D&A

Roberto Maestre

3/21/2017

Introduction

In order to create the C++ code to provide several simple functions to study regression splines and additive models, we have used the code and ideas provided by the book Generalized Additive Models: An Introduction with R.

```
library(data.table)
library(ggplot2)
library(gridExtra)
library(microbenchmark)
library(mgcv) # To compare with full model
```

```
## Loading required package: nlme

## This is mgcv 1.8-14. For overview type 'help("mgcv-package")'.
```

```
library(additiveRegressionSpline) # Include C++ library
```

```
##
## Attaching package: 'additiveRegressionSpline'

## The following object is masked from 'package:base':
##
##     trunc
```

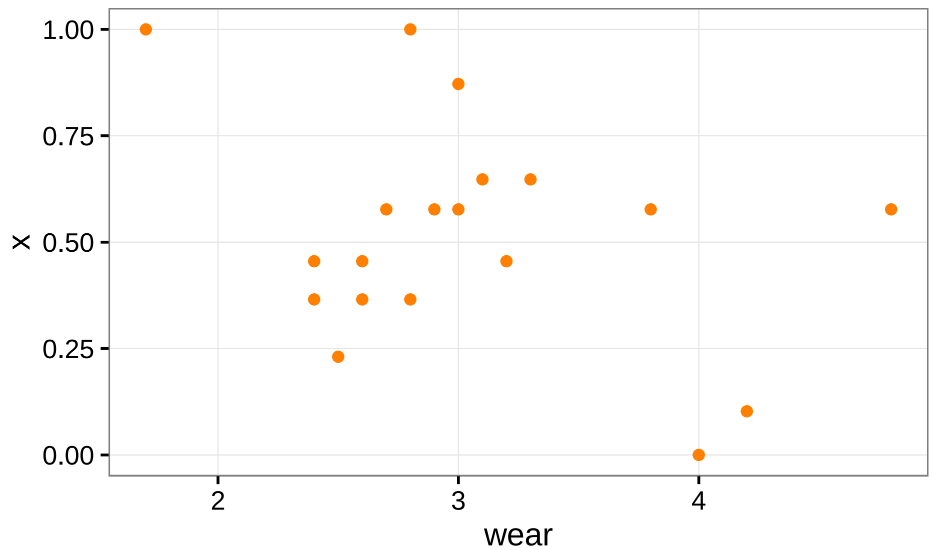
Data points

additive-splines-regression-

Spline regression and additive models (Simple introduction with C++). We provide an example of C++ implementation of additive regression splines through Rcpp and RcppArmadillo

Simple data points

```
size = c(1.42,1.58,1.78,1.99,1.99,1.99,2.13,2.13,2.13,
         2.32,2.32,2.32,2.32,2.32,2.43,2.43,2.78,2.98,2.98)
wear = c(4.0,4.2,2.5,2.6,2.8,2.4,3.2,2.4,2.6,4.8,2.9,
         3.8,3.0,2.7,3.1,3.3,3.0,2.8,1.7)
x = size - min(size); x = x/max(x) #Standarize
d = data.frame(wear, x)
ggplot(aes(x=wear, y=x), data=d) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



Univariate spline regression

We use the next model matrix to adjust a univariate cubic spline regresion

$$\mathbf{y} \sim \begin{pmatrix} 1 & x_1 & b_{11} & \cdots & b_{k1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_n & b_{1n} & \cdots & b_{k1n} \end{pmatrix}$$

```
# Adjust the model
knots = seq(from=0.0,to=1,by=0.3)
knots
```

```
## [1] 0.0 0.3 0.6 0.9
```

```
fit <- splineModel(x, knots, as.matrix(wear))
```

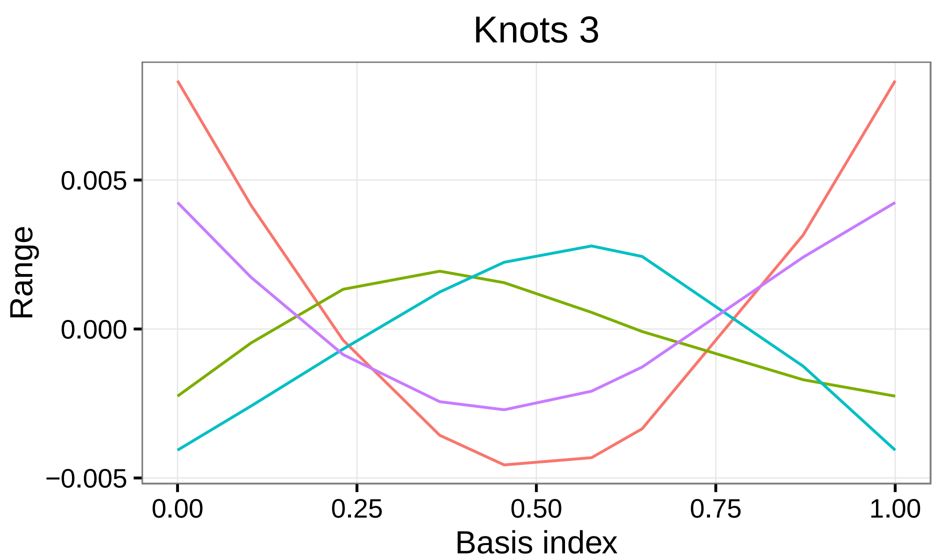
Plot basis

```
# Plot all basis
X <- fit$modelMatrix
dtBases <- data.table("index"=X[,2], X[,3:dim(X)[2]])
dtBases
```

```
##          index            V1            V2            V3            V4
##   1: 0.0000000  0.0083333333 -2.254167e-03 -0.0040666667  0.0042458333
##   2: 0.1025641  0.0041451277 -4.656813e-04 -0.0025832101  0.0017319535
##   3: 0.2307692 -0.0003760956  1.333400e-03 -0.0006723177 -0.0008607281
##   4: 0.3653846 -0.0035686012  1.939754e-03  0.0012408700 -0.0024402719
##   5: 0.3653846 -0.0035686012  1.939754e-03  0.0012408700 -0.0024402719
##   6: 0.3653846 -0.0035686012  1.939754e-03  0.0012408700 -0.0024402719
##   7: 0.4551282 -0.0045618265  1.554120e-03  0.0022402848 -0.0027109782
##   8: 0.4551282 -0.0045618265  1.554120e-03  0.0022402848 -0.0027109782
##   9: 0.4551282 -0.0045618265  1.554120e-03  0.0022402848 -0.0027109782
```

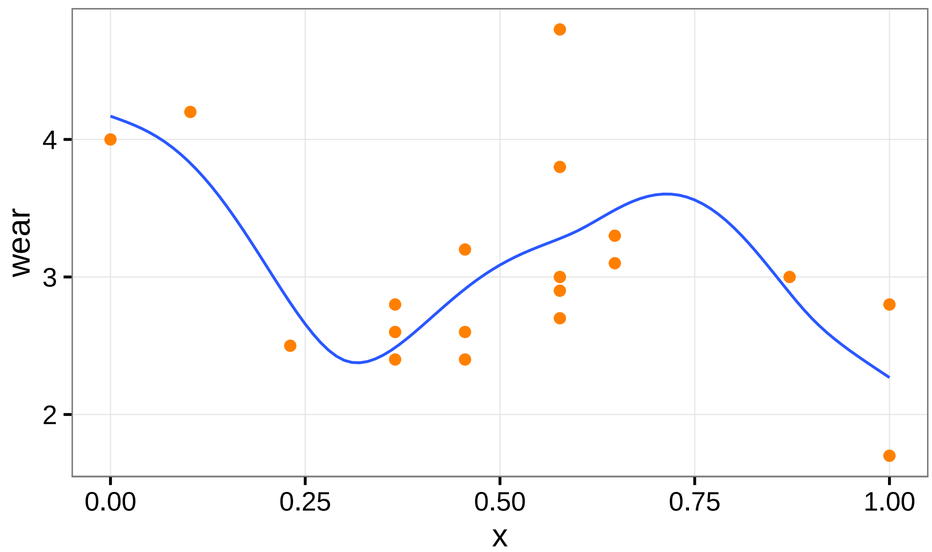
```
## 10: 0.5769231 -0.0043191364  5.569503e-04  0.0027870084 -0.0020877925
## 11: 0.5769231 -0.0043191364  5.569503e-04  0.0027870084 -0.0020877925
## 12: 0.5769231 -0.0043191364  5.569503e-04  0.0027870084 -0.0020877925
## 13: 0.5769231 -0.0043191364  5.569503e-04  0.0027870084 -0.0020877925
## 14: 0.5769231 -0.0043191364  5.569503e-04  0.0027870084 -0.0020877925
## 15: 0.6474359 -0.0033486040 -8.564874e-05  0.0024330758 -0.0012765418
## 16: 0.6474359 -0.0033486040 -8.564874e-05  0.0024330758 -0.0012765418
## 17: 0.8717949  0.0031558005 -1.703729e-03 -0.0012497963  0.0024097988
## 18: 1.0000000  0.0083333333 -2.254167e-03 -0.0040666667  0.0042458333
## 19: 1.0000000  0.0083333333 -2.254167e-03 -0.0040666667  0.0042458333
```

```
dtBasesM <- melt(dtBases, id.vars = c("index"))
ggplot(aes(x=index, y=value, color=variable), data=dtBasesM) +
  geom_line() +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank()) +
  xlab("Basis index") +
  ylab("Range") +
  ggtitle(paste("Knots",length(knots)-1))
```



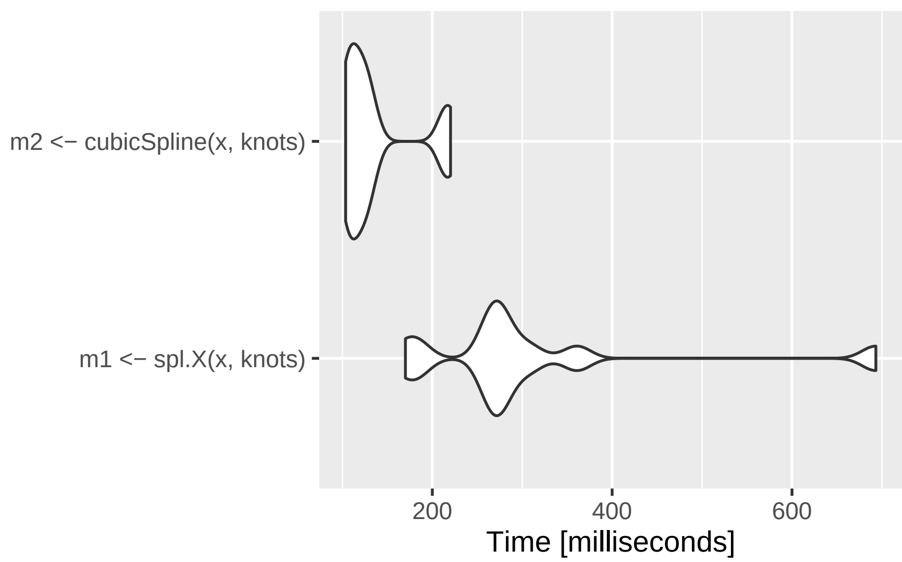
Plot univariable model

```
# Values for prediction
xp <- seq(from=0.0,to=1,by=0.01)
Xpc <- cubicSpline(xp, knots)
# PLot it
ggplot(aes(x=x, y=wear), data=data.frame(x,wear)) +
  geom_point(color="#FF8000") +
  geom_line(aes(x=xp, y=Xpc%*%fit$betas), data=data.frame(xp,Xpc), color="#2957FF") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



1. Benchmark on R Cubic Spline implementation

```
knots = seq(from=0.0,to=1,by=0.00001) # A lot of
m <- microbenchmark(m1<-spl.X(x, knots), m2<-cubicSpline(x, knots), times=10)
autoplot(m, log=F)
```



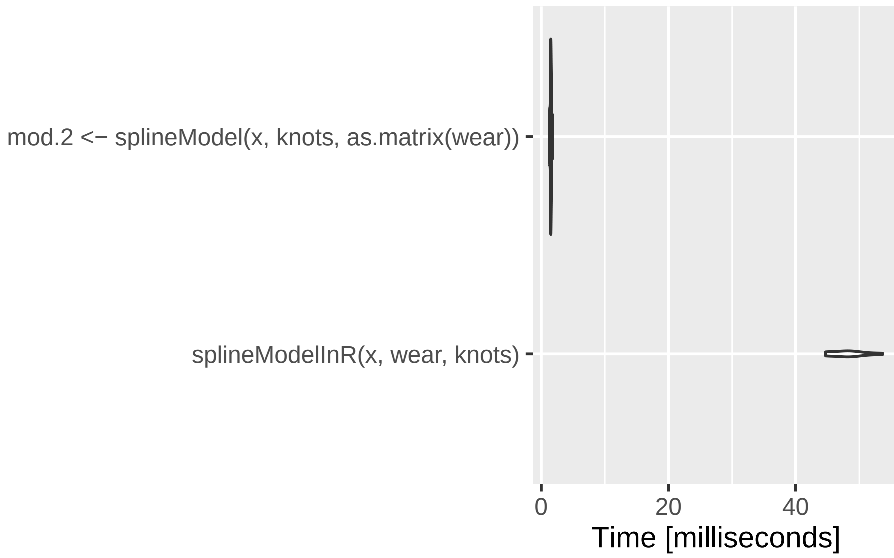

```
all(m1==m2)
```

```
## [1] TRUE
```

2. Benchmark on complete model

```
knots = seq(from=0.0,to=1,by=0.001) # A lot of
# R function has two steps
splineModelInR <- function(x,wear,knots){
```

```
  X <- spl.X(x, knots)
  mod.1 <- lm(wear ~ X - 1)
}
m <- microbenchmark(splineModelInR(x, wear,knots),
                    mod.2 <- splineModel(x, knots, as.matrix(wear)),
                    times=10)
autoplot(m, log=F)
```



Univariate spline regression with penalization

We fit the model minimizing:

$$||\mathbf{y} - \mathbf{X}\beta^2|| + \lambda \int_0^1 [f''(x)]^2 dx$$

Because f is linear in the parameters, β_i, the penalty can always be wirtten as a quadratic form in β:

$$\int_0^1 [f''(x)]^2 dx = \beta^T S \beta$$

therefore, the penalized regression spline fitting problem is to minimize:

$$||\mathbf{y} - \mathbf{X}\beta^2|| + \lambda \beta^T \mathbf{S} \beta$$

For practical computation therefore, note that:

$$\left|\left| \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{B} \end{bmatrix} \beta \right|\right|^2 = ||\mathbf{y} - \mathbf{X}\beta^2|| + \lambda \beta^T \mathbf{S} \beta$$

where $\mathbf{B}$ is any squared root of the matrix $\mathbf{S}$ such $\mathbf{B}^T\mathbf{B} = \mathbf{S}$.

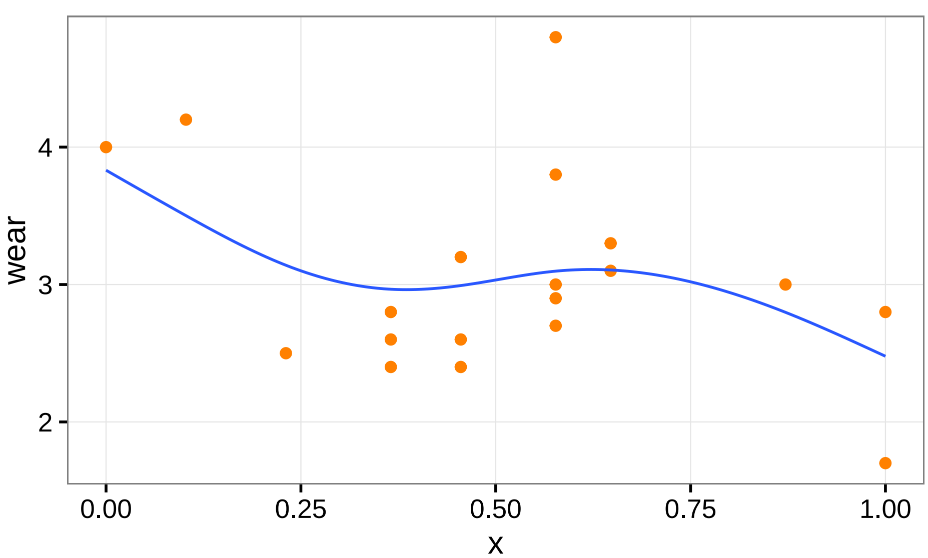
```r
# Penalization fit
knots = seq(from=0.0,to=1,by=0.01)
knots
```

```
##   [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13
##  [15] 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27
##  [29] 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41
##  [43] 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55
##  [57] 0.56 0.57 0.58 0.59 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69
##  [71] 0.70 0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83
##  [85] 0.84 0.85 0.86 0.87 0.88 0.89 0.90 0.91 0.92 0.93 0.94 0.95 0.96 0.97
##  [99] 0.98 0.99 1.00
```

```r
fitP <- splineModelPenalized(wear, x, knots,lambda = 0.01)
# Values for prediction
xp <- seq(from=0.0,to=1,by=0.001)
Xpc <- cubicSpline(xp, knots)
```

Plot regression with $\lambda = 0.01$

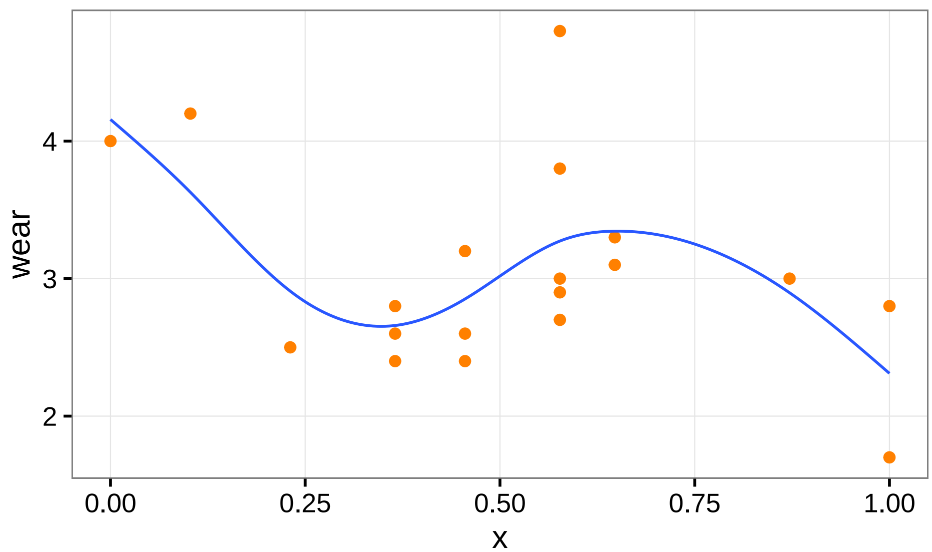
```r
# Plot it
ggplot(aes(x=x, y=wear), data=data.frame(x,wear)) +
  geom_point(color="#FF8000") +
  geom_line(aes(x=xp, y=Xpc %*% fitP$betas), data=data.frame(xp,Xpc), color="#2957FF") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



Plot regression with $\lambda = 0.001$

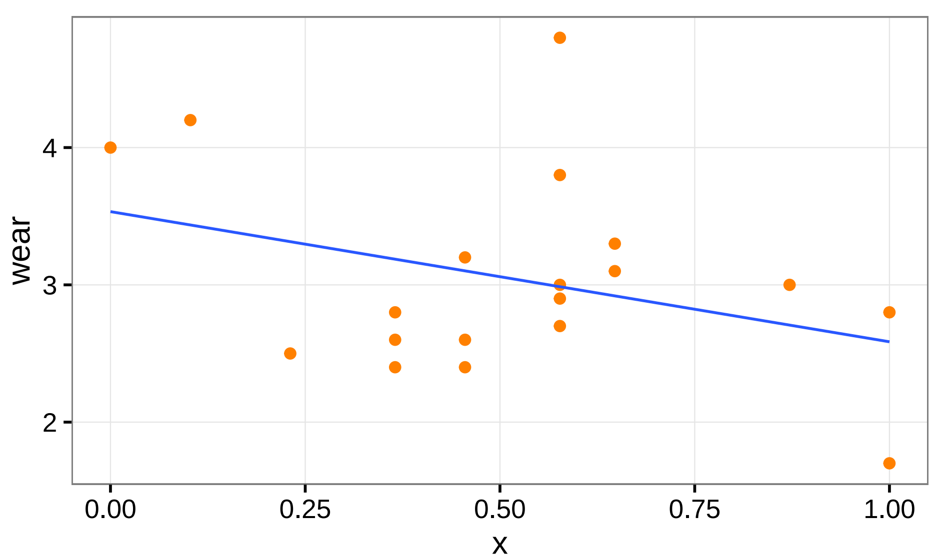
```r
fitP2 <- splineModelPenalized(wear, x, knots,lambda = 0.001)
# PLot it
ggplot(aes(x=x, y=wear), data=data.frame(x,wear)) +
  geom_point(color="#FF8000") +
  geom_line(aes(x=xp, y=Xpc %*% fitP2$betas), data=data.frame(xp,Xpc), color="#2957FF") +
```

```
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



Plot regression with $\lambda = 1000$

```
fitP3 <- splineModelPenalized(wear, x, knots,lambda = 1000)
# PLot it
ggplot(aes(x=x, y=wear), data=data.frame(x,wear)) +
  geom_point(color="#FF8000") +
  geom_line(aes(x=xp, y=Xpc %*% fitP3$betas), data=data.frame(xp,Xpc), color="#2957FF") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



GCV - General Cross Validation

Since GCV has computational adventages over OCV, the next formula can be proposed to compute GVC:

$$\mathcal{V}_g = \frac{n \sum_{i=1}^{n} (y_i - \hat{f}_i)^2}{[tr(\mathbf{I} - \mathbf{A})]^2}$$

```r
# Fit simple models again
knots = seq(from=0.0,to=1,by=0.3)
system.time(mod.1 <- prs.fit(wear, x, knots, lambda = 0.0001))
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```r
system.time(mod.2 <- splineModelPenalized(wear, x, knots, lambda = 0.0001))
```

```
##    user  system elapsed
##       0       0       0
```

```r
# Check if C function is working well
all(trunc(hatMatrix(mod.2$modelatrix, dim(mod.2$modelatrix)[1]), prec = 5)
    == trunc(hatvalues(mod.1), prec = 5))
```

```
## [1] TRUE
```

```r
# Check GCV
modC <- splineModelPenalized(wear, x, knots, lambda = 0.1)
modC$rss
```

```
## [1] 8.11175
```

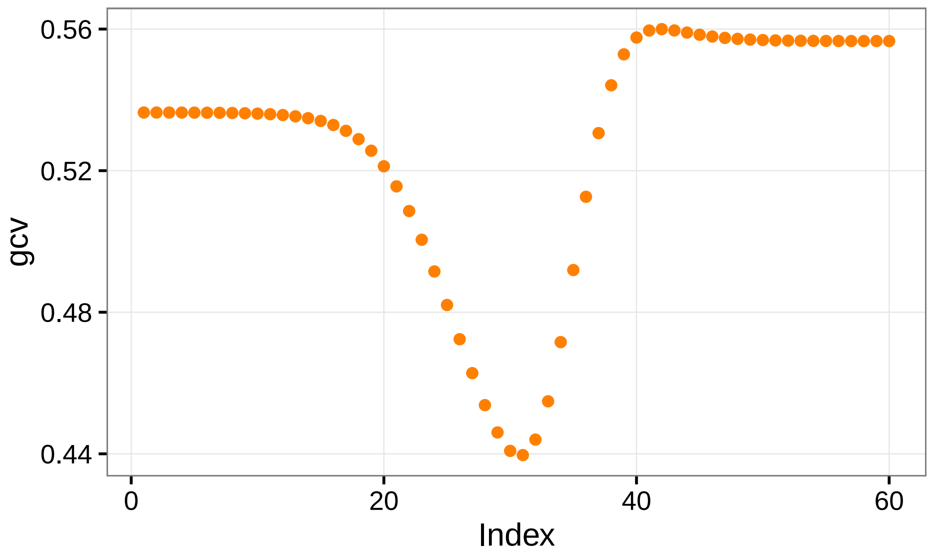
```r
modC$gcv
```

```
## [1] 0.5592888
```

Perform a GCV

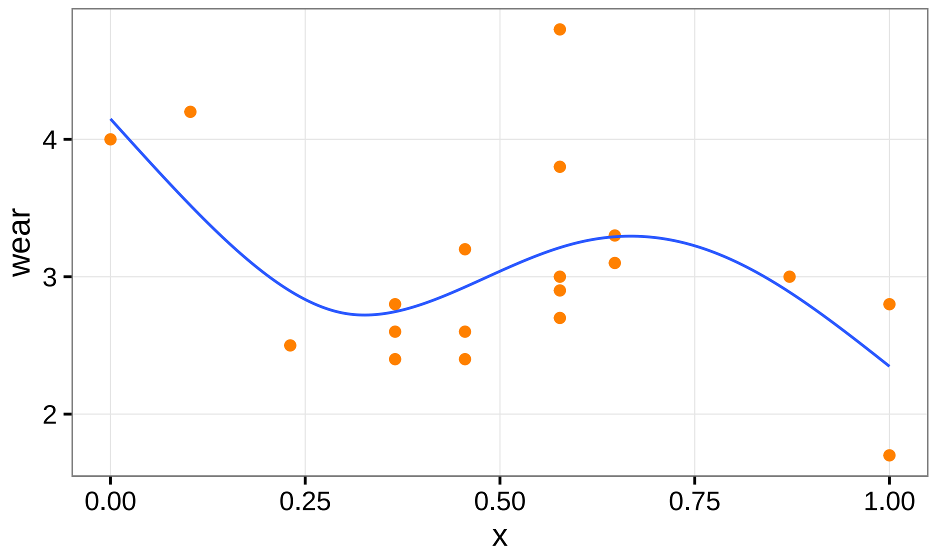
```r
# Select less knots
knots = seq(from=0.0,to=1,by=0.3)
# Perfom GCV
bestGCV <- .Machine$double.xmax
bestLambda <- 0
# Perform search
lambda <- 1e-8
V <- rep(0,60)
for(i in seq(1,60)) {
  modC <- splineModelPenalized(wear, x, knots, lambda = lambda)
  V[i] <- modC$gcv
  if (V[i] < bestGCV){
    bestGCV <- V[i]
    bestLambda <- lambda
  }
  lambda <- lambda * 1.5
```

```
}
# Plot search
ggplot(aes(seq_along(V), V), data=data.table(V)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank()) +
  xlab("Index") +
  ylab("gcv")
```



Plot best *lambda*

```
modBest <- splineModelPenalized(wear, x, knots, lambda = bestLambda)
# Values for prediction
xp <- seq(from=0.0,to=1,by=0.001)
Xpc <- cubicSpline(xp, knots)
# Plot it
ggplot(aes(x=x, y=wear), data=data.frame(x,wear)) +
  geom_point(color="#FF8000") +
  geom_line(aes(x=xp, y=Xpc %*% modBest$betas), data=data.frame(xp,Xpc), color="#2957FF") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
```



Bivariate spline regression

A simple additive model structure:

$$y = f_1(x) + f_2(z) + \epsilon_i$$

The fact that the model contains more that one function of two variables introduces an identifitability problem: f_1 and f_2 are each only estimable to within an additive constant.

An additive regression model provides a clear and unequivocal example of the frequently called $cofounding$. In general terms, cofounding accurs when a variable or variables influence the relationship between another variable and the outcome being studied.

$$\text{univariate} : y_i = A + B_1 x_{i1} + \epsilon_i, \text{bivariate} : y_i = a + b_1 x_{i1} + b_2 x_{i2} + \epsilon_i$$

As we can see, the relationship of the variable x_1 to the dependent variable y depends of the presence of the variable x_2 to the extent that the estimated regression coefficient $\hat{B}_1$ differs from $\hat{b}_1$. Thus, when $\hat{B}_1 \neq \hat{b}_1$, the variable x_2 is said to have a confounding influence on the relationship between x_1 and y. Conversely, when $\hat{B}_1 = \hat{b}_1$, then the variable x_2 does not influence the relationship between x_1 and y.

Each smooth function can be represented using a penalizaed regression spline basis:

$$f_1(x) = \delta_1 + x\delta_2 + \sum_{j=1}^{q_1-2} R(x, x_j^*)\delta_{j+2}$$

and

$$f_2(x) = \gamma_1 + x\gamma_2 + \sum_{j=1}^{q_2-2} R(z, z_j^*)\gamma_{j+2}$$

where:

- δ, γ are the unknow paramters for f_1, f_2 respectively.
- q_1, q_2 are the number of unkonw parameters for f_1, f_2.
- x_j^*, z_j^* are the knot locations for the two functions.

and the cofounding problem between δ_1, γ_1 can be solved (in a simple way) constraining one og them to zero, say $\gamma_1 = 0$. Having done this, it is easy to see that the additive model can be writen in the linear model form $\mathbf{y} - \mathbf{X}\beta^2 + \epsilon$, where the i^{th} row of the model matrix is now:

$$\mathbf{X}_i = \left[1, x_i, R(x_i, x_1^*), R(x_i, x_{q_1-2}^*), z_i, R(z_i, z_1^*), ..., R(z_i, z_{q_2-2}^*)\right]$$

and the parameter vector is $\beta = [\lambda_1, \lambda_2, ..., \lambda_{q1}, \gamma_1, \gamma_2, ..., \gamma_{q1}]$ and is obtained minimizin the penalized least squares objective:

$$||\mathbf{y} - \mathbf{X}\beta^2|| + \lambda_1 \beta^T \mathbf{S}_1 \beta + \lambda_2 \beta^T \mathbf{S}_2 \beta$$

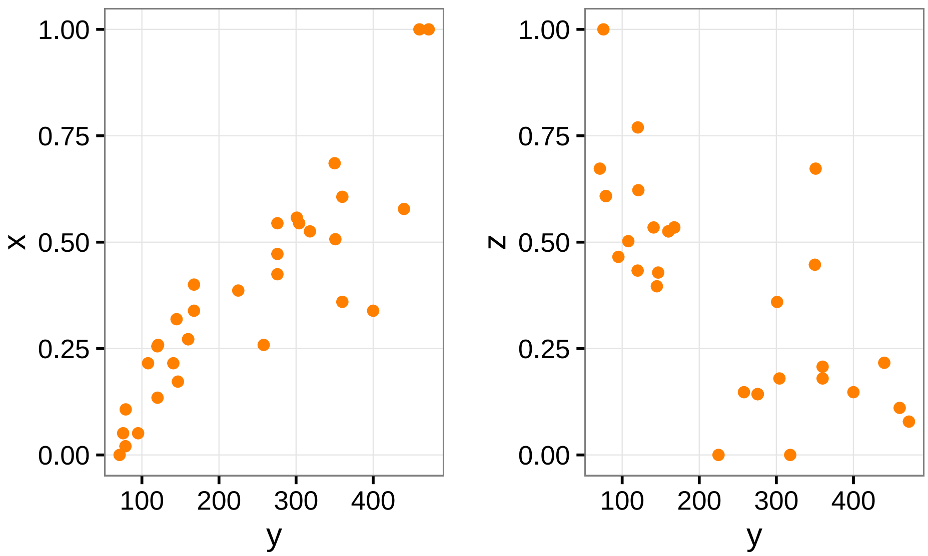
Defining, $\mathbf{S} \equiv \lambda_1 \mathbf{S}_1 + \lambda_2 \mathbf{S}_2$, the objective can be re-written:

$$\left|\left| \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{X} \\ \mathbf{B} \end{bmatrix} \beta \right|\right|^2$$

where $\mathbf{B}$ is any matrix square root such that $\mathbf{B}^T\mathbf{B} = \mathbf{S}$.

Two variables

```
# data(trees)
# rg <- range(trees$Girth)
# trees$Girth <- (trees$Girth - rg[1])/(rg[2]-rg[1])
# rh <- range(trees$Height)
# trees$Height <- (trees$Height - rh[1])/(rh[2]-rh[1])
# x <- trees$Girth
# z <- trees$Height
# y <-trees$Volume
# plot(trees)

data(mtcars)
# Standareized and form curve on data
mtcars$mpg <- mtcars$mpg^-1
rg <- range(mtcars$mpg)
mtcars$mpg <- (mtcars$mpg - rg[1])/(rg[2]-rg[1])
rh <- range(mtcars$drat)
mtcars$drat <- (mtcars$drat - rh[1])/(rh[2]-rh[1])
# Variables
x <- mtcars$mpg
z <- mtcars$drat
y <- mtcars$disp

# Plot variables
p1<-ggplot(aes(y,x), data=data.table(x,y,z)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
p2<-ggplot(aes(y,z), data=data.table(x,y,z)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())

grid.arrange(p1,p2,ncol=2)
```



Fit bivariate model

```
# Number of knots for both variables
q <- 10
# Calculate knots
xk <- as.numeric(quantile(unique(x),1:(q-2)/(q-1)))
zk <- as.numeric(quantile(unique(z),1:(q-2)/(q-1)))
# We asume the same well distributed knots for both x and z variables

# Get matrix (only one calculation is required)
matrices <- splineModelPMatrix(x,z,xk,zk)

# Model cross validation
sp<-c(0,0)
V <- rep(0,(30*30)-1)
cont <- 0
for (i in 1:30) for (j in 1:30) {
  sp[1]<-1e-5*2^(i-1)
  sp[2]<-1e-5*2^(j-1)
  modBi <- splineModelPMatrixFit(y,x,matrices$modelMatrix,matrices$sX,
                                  matrices$sZ,lambda_x=sp[1],lambda_z=sp[2])
  V[cont] <- modBi$gcv
  cont <- cont + 1
  if (i+j==2) best<-modBi else
  if (modBi$gcv<best$gcv) best<-modBi
}
best$gcv
```

```
## [1] 2882.278
```

```
bLX <- best$lambda_x
bLX
```

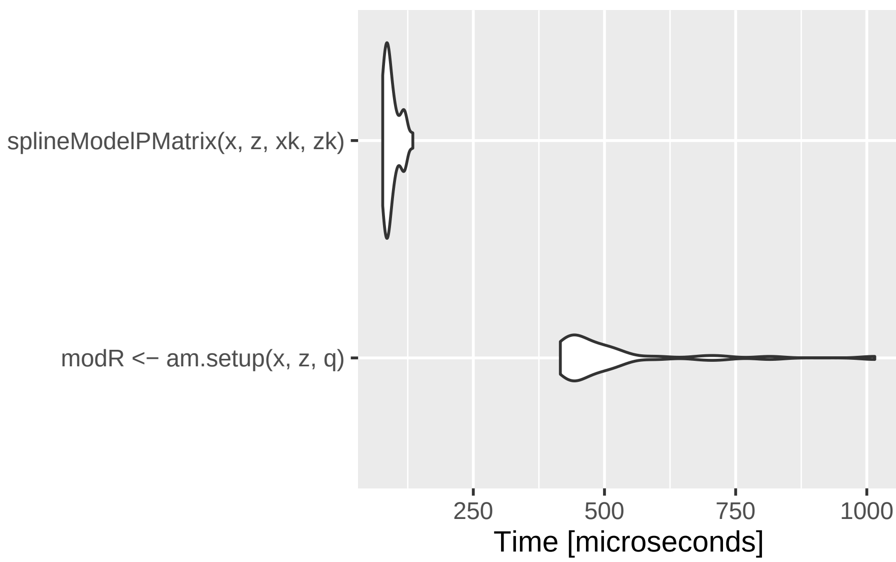
```
## [1] 5368.709
```

```r
bLZ <- best$lambda_z
bLZ
```

```
## [1] 0.00128
```

Benchmark

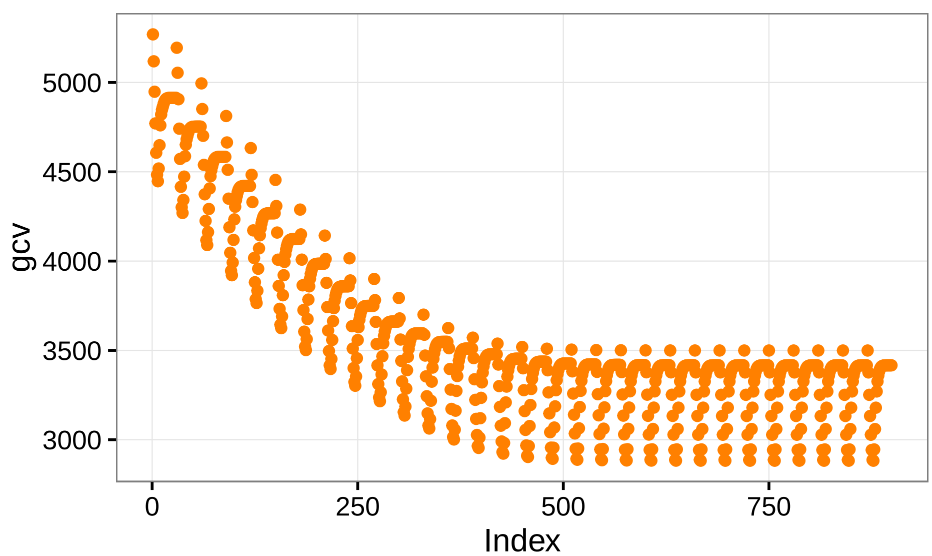
```r
# Number of knots for both variables
q <- 10
# Calculate knots
xk <- as.numeric(quantile(unique(x),1:(q-2)/(q-1)))
zk <- as.numeric(quantile(unique(z),1:(q-2)/(q-1)))

# Get matrix (only one calculation is required)
m1 <- microbenchmark(modR<-am.setup(x,z,q), splineModelPMatrix(x,z,xk,zk), times=30)
autoplot(m1, log=F)
```




```r
# Fit models
m2 <- microbenchmark(fit.am(y,modR$X,modR$S,sp),
                     splineModelPMatrixFit(y,x,matrices$modelMatrix,matrices$sX,
                                           matrices$sZ,lambda_x=sp[1],lambda_z=sp[2])
                     , times=30)
boxplot(m2)
```

Fit bivariate model GCV

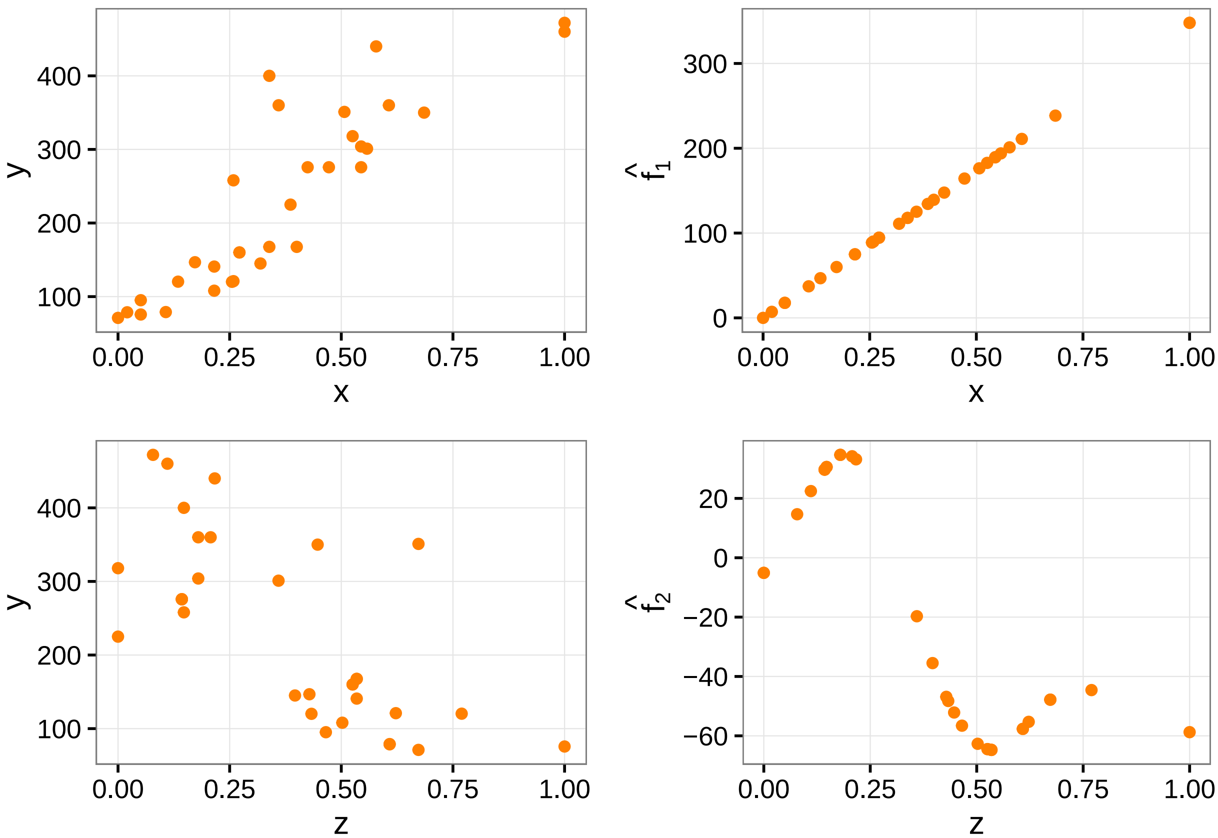
```
# Plot search
ggplot(aes(seq_along(V), V), data=data.table(V)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank()) +
  xlab("Index") +
  ylab("gcv")
```



Predict f_1 and f_2

```
# Drop intercept and Z smooth
modBi <- splineModelPMatrixFit(y,x,matrices$modelMatrix,
                                           matrices$sX,matrices$sZ,
                                           lambda_x=bLX,lambda_z=bLZ)
modBi$betas[1]<-0
modBi$betas[11:19]<-0
f0 <- modBi$modelMatrix %*% modBi$betas
# PLots
p1 <- ggplot(aes(x,y), data=data.table(x,y)) +
  geom_point(color="#FF8000") +
```

```r
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
p2 <- ggplot(aes(x,f0), data=data.table(x,f0)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank()) +
  ylab(expression(hat(f[1])))


# Drop intercept and X smooth
modBi <- splineModelPMatrixFit(y,x,matrices$modelMatrix,
                                          matrices$sX,matrices$sZ,
                                          lambda_x=bLX,lambda_z=bLZ)
modBi$betas[1]<-0
modBi$betas[2:10]<-0
f1 <- modBi$modelMatrix %*% modBi$betas
# PLots
p3 <- ggplot(aes(z,y), data=data.table(z,y)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank())
p4 <- ggplot(aes(z,f1), data=data.table(z,f1)) +
  geom_point(color="#FF8000") +
  theme_bw() +
  theme(legend.position="none", panel.grid.minor = element_blank()) +
  ylab(expression(hat(f[2])))

# Plot grid arrange
grid.arrange(p1,p2,p3,p4, ncol=2, nrow=2)
```



Gam comparision

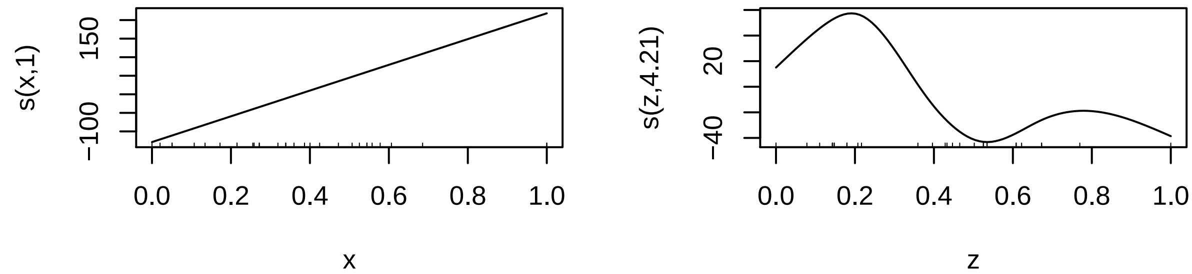
```
fitGam <- gam(y~s(x)+s(z))
# Compare GCV
fitGam$gcv.ubre
```

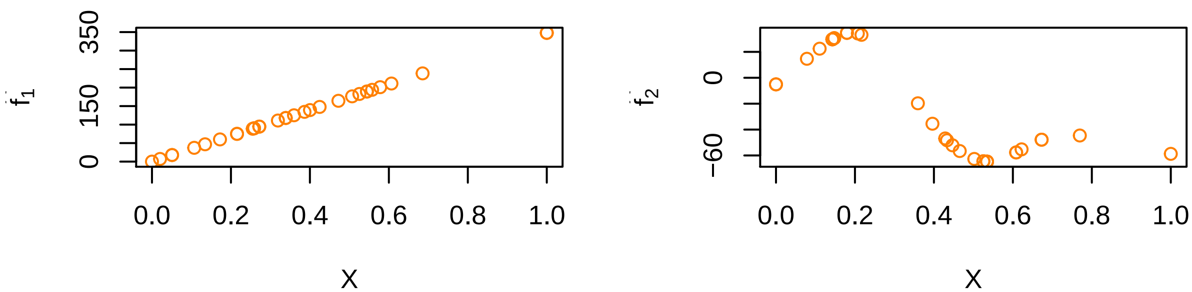
```
##   GCV.Cp
## 2925.844
```

```
modBi$gcv
```

```
## [1] 2882.278
```

```
# Components comparision
par(mfrow=c(2,2))
plot(fitGam, select=1, scale=0, se=F)
plot(fitGam, select=2, scale=0, se=F)
plot(x,f0,xlab="X", ylab=expression(hat(f[1])), col="#FF8000")
plot(z,f1,xlab="X", ylab=expression(hat(f[2])), col="#FF8000")
```





Predict with both funcions

```
# Predict one value
predict(fitGam, newdata=data.table("x"=0.8,"z"=0.2))
```

```
##        1
## 436.6982
```

```
# Predict one value
matrices2 <- splineModelPMatrix(c(0.8),c(0.2),xk,zk)
# Fit the model again (two have the whole coeeficiets)
modBi <- splineModelPMatrixFit(y,x,matrices$modelMatrix,
                               matrices$sX,matrices$sZ,
                               lambda_x=bLX,lambda_z=bLZ)
matrices2$modelMatrix %*% modBi$betas
```

```
##           [,1]
## [1,] 436.5655
```

Annex A. C++ code

```
#include <RcppArmadillo.h>
using namespace Rcpp;
//[[Rcpp::depends(RcppArmadillo)]]

//[[Rcpp::export]]
arma::vec hatMatrix(arma::mat m, int length) {
  // Calculate the influence vector
  arma::vec diag = arma::diagvec(m * arma::inv(m.t()*m) * m.t());
  // Return first length values from the diagonal
```

```
  return(diag.subvec(0,length-1));
}

//[[Rcpp::export]]
arma::mat cubicSpline(NumericVector x, NumericVector knots) {
  // Create an empty matrix plus two columns:
  // one for intercept and another for x values
  arma::mat modelMatrix = arma::ones(x.length(), knots.length()+2);
  // Update second column with x values
  modelMatrix.col(1) = Rcpp::as<arma::colvec>(x);
  // Iteration over columns
  for(int i=0; i<knots.length(); i++) {
    // Apply on first knot
    double z = knots[i];
    // Vector to save one basis i.e.: one column of design matrix
    NumericVector basis(x.length());
    // Define iterators
    NumericVector::iterator it, out_it;
    for(it = x.begin(), out_it=basis.begin();it<x.end();++it,++out_it) {
      // Cubic spline
      *out_it = ((std::pow(z-0.5,2) - (1.0/12.0)) * (std::pow(*it-0.5,2) - (1.0/12.0))) /4 -
        (std::pow(std::abs(*it-z)-0.5,4) - (std::pow(std::abs(*it-z)-0.5,2)/2) + (7.0/240.0)) / 24.0;
    }
    // Update column from model matrix
    modelMatrix.col(i+2) = Rcpp::as<arma::colvec>(basis);
  }
  return(modelMatrix);
}

//[[Rcpp::export]]
List splineModel(NumericVector x, NumericVector knots, arma::mat y) {
  // Model matrix with cubic basis
  arma::mat modelMatrix = cubicSpline(x, knots);
  // Solve system of equations (fast mode)
  arma::mat betas = arma::solve(modelMatrix, y);
  // Return results
  List results;
  results["betas"] = betas;
  results["modelMatrix"] = modelMatrix;
  return(results);
}

//[[Rcpp::export]]
List splineModelPenalized(arma::vec y, NumericVector x, NumericVector knots, double lambda) {
  // Model matrix
  arma::mat modelMatrixKnots = arma::zeros(knots.length()+2, knots.length()+2);
  // Cubic splines on knots modelMatrixKnots
  arma::mat cubicS = cubicSpline(knots,knots);
  modelMatrixKnots.submat(2,2,modelMatrixKnots.n_rows-1,modelMatrixKnots.n_cols-1) =
    cubicS.submat(0,2,cubicS.n_rows-1,cubicS.n_cols-1);

  // Model matrix
  arma::mat  modelMatrix = cubicSpline(x, knots);
  // Eign decomposition
  arma::vec eigval;
  arma::mat eigvec;
  arma::eig_sym(eigval, eigvec,modelMatrixKnots,"std");
  eigval = arma::pow(eigval, 0.5);
  // Replace NAN for 0
  eigval.replace(arma::datum::nan, 0);
  // Augmented model matrix
  arma::mat mS = eigvec * arma::diagmat(eigval) * eigvec.t();
  mS = mS * std::pow(lambda, 0.5);
  // Join two matrix
  arma::mat mSA = arma::join_cols(modelMatrix, mS);
  // Augmented y vector
  arma::mat yAug = arma::join_cols(y, arma::zeros(knots.length()+2));
```

```
  // Solve system of equations (fast mode)
  arma::mat betas;
  try {
    betas = arma::solve(mSA, yAug);
  } catch(...) {
    ::Rf_error("Solve can not find a solution for the system.");
  }
  // General Crodd Validation (GCV)) calculation
  arma::vec influence = hatMatrix(mSA, x.length());
  // Fitted values
  arma::vec fitted = mSA * betas;
  int n = x.length();
  fitted = fitted.subvec(0, n-1);
  double rss = sum(arma::pow(y - fitted, 2));
  double gcv = (n*rss) / std::pow((n-sum(influence)), 2);

  // Return results
  List results;
  results["betas"] = betas;
  results["modelatrix"] = mSA;
  results["rss"] = rss;
  results["gcv"] = gcv;
  return(results);
}



//[[Rcpp::export]]
List splineModelPMatrix(NumericVector x, NumericVector z,
                             NumericVector knots_x, NumericVector knots_z) {
  int q = (2*(knots_x.length()+2))-1;
  // Penalty matrix
  // Get penalties matrix 1
  arma::mat sCX = arma::zeros(knots_x.length()+2, knots_x.length()+2);
  arma::mat cubicSX = cubicSpline(knots_x,knots_x);
  sCX.submat(2,2,sCX.n_rows-1,sCX.n_cols-1) = cubicSX.submat(0,2,cubicSX.n_rows-1,cubicSX.n_cols-1);
  // Update matrix S with penalty matrix 1
  arma::mat sX = arma::zeros(q,q);
  sX.submat(1,1,knots_x.length()+1,knots_x.length()+1) = sCX.submat(1,1,sCX.n_rows-1,sCX.n_cols-1);

  // Get penalties matrix 2
  arma::mat sCZ = arma::zeros(knots_z.length()+2, knots_z.length()+2);
  arma::mat cubicSZ = cubicSpline(knots_z,knots_z);
  sCZ.submat(2,2,sCZ.n_rows-1,sCZ.n_cols-1) =
    cubicSZ.submat(0,2,cubicSZ.n_rows-1,cubicSZ.n_cols-1);
  // Update matrix S with penalty matrix 2
  arma::mat sZ = arma::zeros(q,q);
  sZ.submat(knots_x.length()+2,knots_x.length()+2,sZ.n_cols-1,sZ.n_rows-1) =
    sCZ.submat(1,1,sCZ.n_rows-1,sCZ.n_cols-1);

  // Model matrix
  int n = x.length();
  arma::mat modelMatrix = arma::ones(n,q);
  // Cubic splines smooths over knots
  arma::mat xSmooth = cubicSpline(x,knots_x);
  arma::mat zSmooth = cubicSpline(z,knots_z);
  // Update model matrix with smooths (we remove the intercept for xSmooth
  // and zSmooth i.e.: dropping the first column)
  modelMatrix.submat(0,1,modelMatrix.n_rows-1,xSmooth.n_cols-1) =
    xSmooth.submat(0,1,xSmooth.n_rows-1,xSmooth.n_cols-1);
  modelMatrix.submat(0,zSmooth.n_cols,modelMatrix.n_rows-1,modelMatrix.n_cols-1) =
    zSmooth.submat(0,1,zSmooth.n_rows-1,zSmooth.n_cols-1);

  // Return results
  List results;
  results["modelMatrix"] = modelMatrix;
  results["sX"] = sX;
```

```
  results["sZ"] = sZ;
  results["sCX"] = sCX;
  results["sCZ"] = sCZ;
  results["cubicSX"] = cubicSX;
  results["cubicSZ"] = cubicSZ;
  results["xSmooth"] = xSmooth;
  results["zSmooth"] = zSmooth;
  return(results);
}

//[[Rcpp::export]]
List splineModelPMatrixFit(arma::vec y, NumericVector x,
                           arma::mat modelMatrix,
                           arma::mat sX,arma::mat sZ,
                           double lambda_x, double lambda_z) {
  int n = x.length();
  // Fit model
  arma::mat penaltyM = sX * lambda_x + sZ * lambda_z;
  // Sqrt of penalty matrix
  arma::vec eigval;
  arma::mat eigvec;
  arma::eig_sym(eigval, eigvec, penaltyM,"std");
  eigval = arma::pow(eigval, 0.5);
  // Replace NAN for 0
  eigval.replace(arma::datum::nan, 0);
  // Augmented model matrix
  arma::mat penaltyMSqrt = eigvec * arma::diagmat(eigval) * eigvec.t();
  // Augmented ModelMatrix: Row join between modelMatrix (with smoothings) and penalty matrix
  arma::mat augmentedModelMatrix = arma::join_cols(modelMatrix, penaltyMSqrt);
  // Augmented Y vector (with number of paramters)
  arma::vec yAugmented = arma::zeros(y.n_rows+modelMatrix.n_cols);
  yAugmented.subvec(0,y.n_rows-1) = y;
  // Solve system of equations (fast mode)
  arma::mat betas;
  try {
    betas = arma::solve(augmentedModelMatrix, yAugmented);
  } catch(...) {
    ::Rf_error("Solve can not find a solution for the system.");
  }
  // General Crodd Validation (GCV)) calculation
  arma::vec influence = hatMatrix(augmentedModelMatrix, x.length());
  // Fitted values
  arma::vec fitted = augmentedModelMatrix * betas;
  fitted = fitted.subvec(0, n-1);
  double rss = sum(arma::pow(y - fitted, 2));
  double gcv = (n*rss) / std::pow((n-sum(influence)), 2);

  // Return results
  List results;
  results["modelMatrix"] = modelMatrix;
  results["penaltyM"] = penaltyM;
  results["penaltyMSqrt"] = penaltyMSqrt;
  results["augmentedModelMatrix"] = augmentedModelMatrix;
  results["betas"] = betas;
  results["gcv"] = gcv;
  results["rss"] = rss;
  results["lambda_x"] = lambda_x;
  results["lambda_z"] = lambda_z;
  return(results);
}
```