FDA: Smoothing

TOPICS:

- From rough data to smooth functions

```
# Upload noisy data
noisycurve <- read.table("noisycurvebis.txt",header=T)
head(noisycurve)
```

```
##   Abscissa        X0
## 1     0.00 0.7058236
## 2     0.01 0.7062072
## 3     0.02 0.6966732
## 4     0.03 0.6924990
## 5     0.04 0.6915923
## 6     0.05 0.6828690
```

```
dim(noisycurve)
```

```
## [1] 101   2
```

101 evaluations of the function

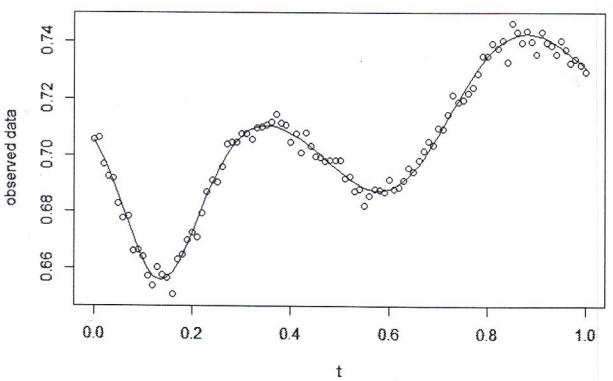
```
Xobs0 <- noisycurve$X0
abscissa <- noisycurve$Abscissa
NT <- length(abscissa)

plot(abscissa,Xobs0,xlab="t",ylab="observed data")

# Upload true data
truecurve <- read.table("truecurve.txt",header=T)
head(truecurve)
```

true evaluations of the curve

```
##   Abscissa    X0vera    X1vera    X2vera
## 1     0.00 0.7052652 -0.2727032 -4.598676
## 2     0.01 0.7023075 -0.3188459 -4.603647
## 3     0.02 0.6988909 -0.3641894 -4.432447
## 4     0.03 0.6950328 -0.4067724 -4.044348
## 5     0.04 0.6907726 -0.4442018 -3.393694
## 6     0.05 0.6861755 -0.4736025 -2.429753
```

first and second derivative

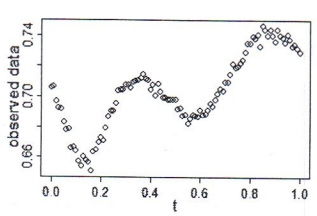
```
points(abscissa,truecurve$X0vera,type="l")
```

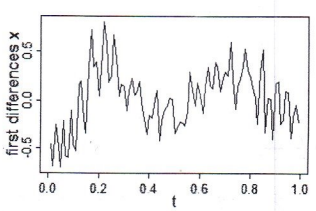


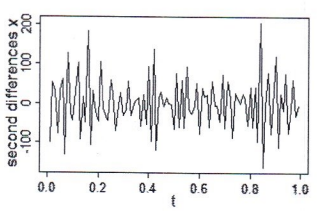
To see what happens to the 1st and 2nd derivative when we add the noise we compute the central finite differences:

```
# compute the central finite differences
rappincX1 <- (Xobs0[3:NT]-Xobs0[1:(NT-2)])/(abscissa[3:NT]-abscissa[1:(NT-2)])
rappincX2 <- ((Xobs0[3:NT]-Xobs0[2:(NT-1)])/(abscissa[3:NT]-abscissa[2:(NT-1)])-(Xobs0[2:(NT-1)]-Xobs0[1:(NT-2)])/(abscissa[
2:(NT-1)]-abscissa[1:(NT-2)]))*2/(abscissa[3:(NT)]-abscissa[1:(NT-2)])
```

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
```



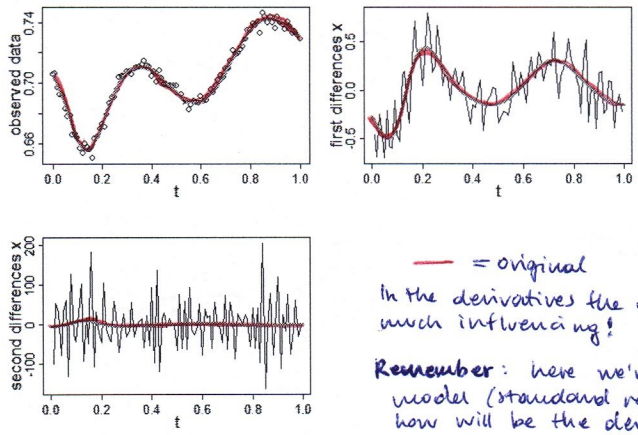




we see a noise on the original data that is almost neglectable, but the effects of the noise on the first and second derivative are so much influencing (not neglectable)

We make a comparison with the original function : (the one that generated data)

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(truecurve$Abscissa,truecurve$X0vera,type='l',col="orange",lwd=3)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(truecurve$Abscissa,truecurve$X1vera,type='l',col="orange",lwd=3)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(truecurve$Abscissa,truecurve$X2vera,type='l',col="orange",lwd=3)
```



— = original

In the derivatives the noise is so much influencing!

Remember: here we've not just trying to fit a linear model (standard regression), we're considering also how will be the derivatives!

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### REGRESSION SPLINES
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------

# Load package fda
library(fda)
```

= fitting of the model with least squares (without penalization)

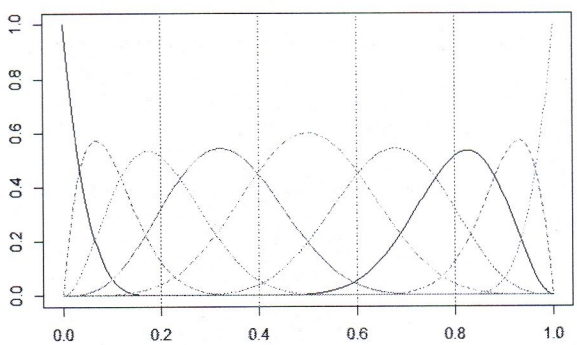
```
# Set parameters
m <- 5          # spline order
degree <- m-1   # spline degree

nbasis <- 9

# Create the basis
help(create.bspline.basis)
```

range of the ascissa

```
basis <- create.bspline.basis(rangeval=c(0,1), nbasis=nbasis, norder=m)
# If breaks are not provided, equispaced knots are created
names(basis)
```

```
##  [1] "call"        "type"        "rangeval"    "nbasis"      "params"
##  [6] "dropind"     "quadvals"    "values"      "basisvalues" "names"
```

```
x11()
plot(basis)
```



```
# Evaluate the basis on the grid of abscissa
basismat<- eval.basis(abscissa, basis)
dim(basismat)
```

```
## [1] 101   9
```

since we had 101 evaluations (101 couples (ascissa, ordinate) for the same function)

```
head(basismat)
```

```
##         bspl5.1   bspl5.2     bspl5.3      bspl5.4      bspl5.5 bspl5.6 bspl5.7
## [1,] 1.0000000 0.0000000 0.000000000 0.000000e+00 0.000000e+00       0       0
## [2,] 0.8145062 0.1783633 0.007049045 8.116319e-05 2.604167e-07       0       0
## [3,] 0.6561000 0.3168125 0.026451389 6.319444e-04 4.166667e-06       0       0
## [4,] 0.5220063 0.4201758 0.055722656 2.074219e-03 2.109375e-05       0       0
## [5,] 0.4096000 0.4930000 0.092555556 4.777778e-03 6.666667e-05       0       0
## [6,] 0.3164063 0.5395508 0.134819878 9.060330e-03 1.627604e-04       0       0
##      bspl5.8 bspl5.9
## [1,]       0       0
## [2,]       0       0
## [3,]       0       0
## [4,]       0       0
## [5,]       0       0
## [6,]       0       0
```

```
# Fit via LS
help(lsfit)

lsfit(basismat, Xobs0, intercept=FALSE)$coef
```

we provide:
- evaluation of the basis of the ascissa grid
- response: X_0 from "noisy curve" ($= Y$)

```
##   bspl5.1   bspl5.2   bspl5.3   bspl5.4   bspl5.5   bspl5.6   bspl5.7   bspl5.8
## 0.7086585 0.6849816 0.6052928 0.7716465 0.6639224 0.6976876 0.7622365 0.7338739
##   bspl5.9
## 0.7311345
```

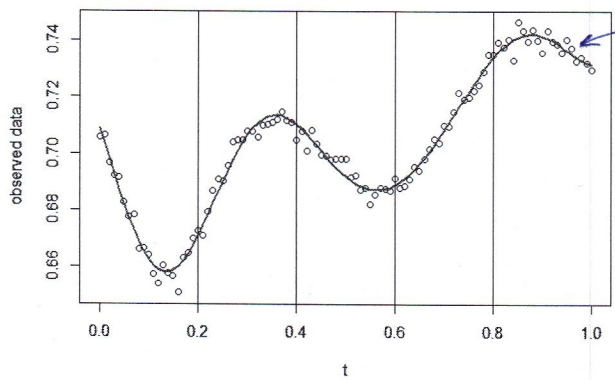
→ coefficients for the 9 basis of the spline basis set

```
Xsp0 <- basismat %*% lsfit(basismat, Xobs0, intercept=FALSE)$coef
```

to get to the evaluation of the regression spline we multiply the two matrices

[evaluations of the basis system] · [coefficients (vector)]

```
x11()
par(mfrow=c(1,1))
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0 ,type="l",col="blue",lwd=2)
abline(v=basis$params)
```

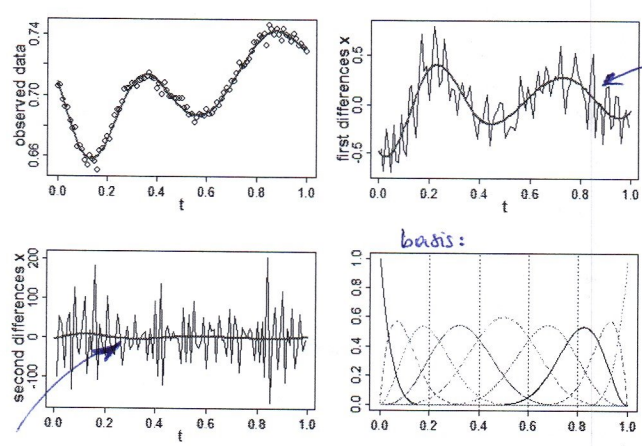


regression spline obtained

```
# to obtain the first derivative (argument Lfdobj=1)
basismat1<- eval.basis(abscissa, basis, Lfdobj=1)
Xsp1 <- basismat1 %*% lsfit(basismat, Xobs0, intercept=FALSE)$coef

# to obtain the second derivative (argument Lfdobj=2)
basismat2<- eval.basis(abscissa, basis, Lfdobj=2)
Xsp2 <- basismat2 %*% lsfit(basismat, Xobs0, intercept=FALSE)$coef
```

evaluation of the basis on the derivative (1st derivative)

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsp1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsp2 ,type="l",col="blue",lwd=2)
plot(basis)
```



1st derivative regression spline

They're good! They're taking out the noise!

2nd derivative regression spline

```
### ---------------------------------------------------------------------------
# alternative code
help(smooth.basis)
Xsp <- smooth.basis(argvals=abscissa, y=Xobs0, fdParobj=basis)
Xsp0bis <- eval.fd(abscissa, Xsp$fd) #  the curve smoothing the data
Xsp1bis <- eval.fd(abscissa, Xsp$fd, Lfd=1) # first derivative
Xsp2bis <- eval.fd(abscissa, Xsp$fd, Lfd=2) # second derivative
df <- Xsp$df   #  the degrees of freedom in the smoothing curve
df
```

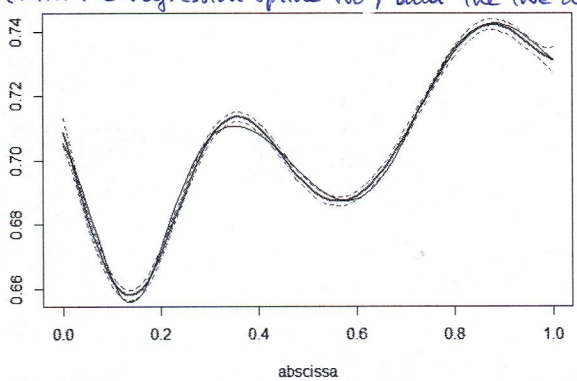
```
## [1] 9
```

```
### ---------------------------------------------------------------------------

# Approximate pointwise confidence intervals
# As in linear models, we can estimate the variance of x(t) as
# sigma^2*diag[phi*(phi'phi)^{-1}(phi)']
S <- basismat%*%solve(t(basismat)%*%basismat)%*%t(basismat) #projection operator
sigmahat <- sqrt(sum((Xsp0-Xobs0)^2)/(NT-nbasis)) #estimate of sigma
lb <- Xsp0-qnorm(0.975)*sigmahat*sqrt(diag(S))
ub <- Xsp0+qnorm(0.975)*sigmahat*sqrt(diag(S))
```

```
x11()
plot(abscissa,Xsp0,type="l",col="blue",lwd=2,ylab="")
points(abscissa,lb,type="l",col="blue",lty="dashed")
points(abscissa,ub,type="l",col="blue",lty="dashed")
points(abscissa,truecurve$X0vera,type="l")
```

Remember again: these are 1 at the time CI (we use the normal instead of t because the sample size is very high, CLT says that when we estimate the mean we get a gaussian distr.)

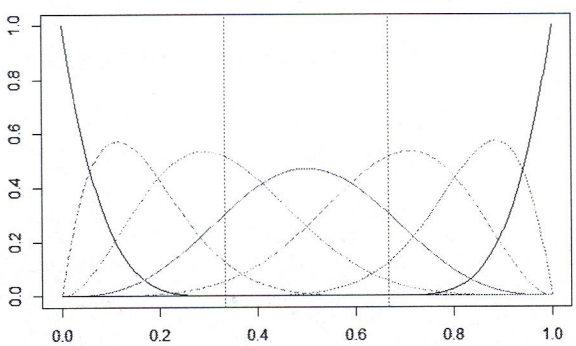
Graphical representation of the CI (---):
(with the regression spline too, and the true curve too)



What happens if we change the number of basis?

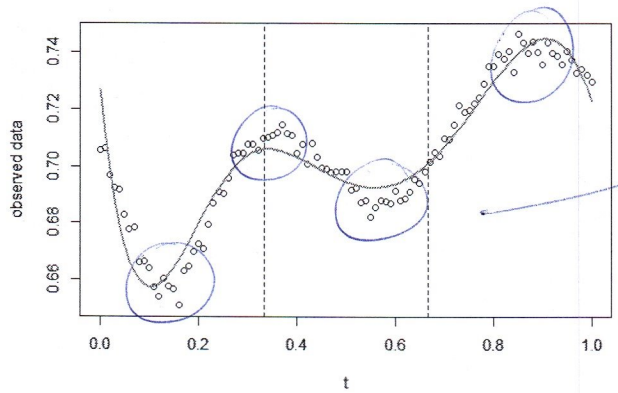
```
# Oversmoothing
nbasis <- 7

basisbis <- create.bspline.basis(c(0,1), nbasis, m)
```

```
x11()
par(mfrow=c(1,1))
plot(basisbis)
```




```
basismatbis <- eval.basis(abscissa, basisbis)
Xsp0bis <- basismatbis %*% lsfit(basismatbis, Xobs0, intercept=FALSE)$coef

basismat1bis <- eval.basis(abscissa, basisbis,Lfdobj=1)
Xsp1bis <- basismat1bis %*% lsfit(basismatbis, Xobs0, intercept=FALSE)$coef

basismat2bis <- eval.basis(abscissa, basisbis,Lfdobj=2)
Xsp2bis <- basismat2bis %*% lsfit(basismatbis, Xobs0, intercept=FALSE)$coef
```

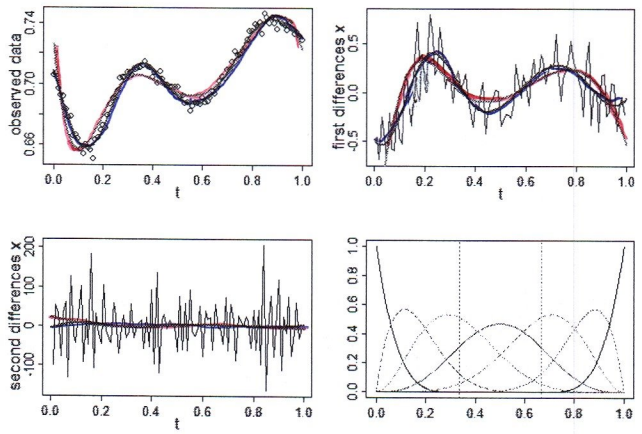
```
x11()
par(mfrow=c(1,1))
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0bis ,type="l",col="green",lwd=2)
abline(v=basisbis$params,lty=2)
```



we're missing some characteristics of the real function (we see it especially near the concavities)

⟹ OVERSMOOTHING

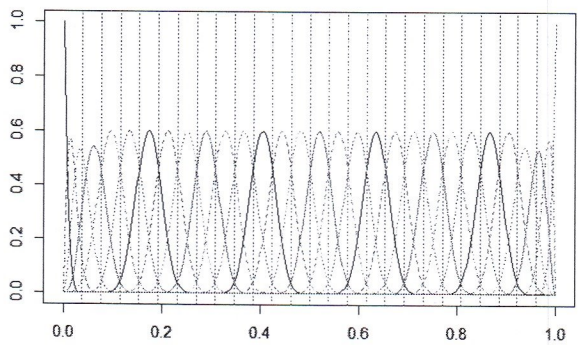
```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsp1bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsp2bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp2 ,type="l",col="blue",lwd=2)
plot(basisbis)
```



— = 7 basis
— = 9 basis

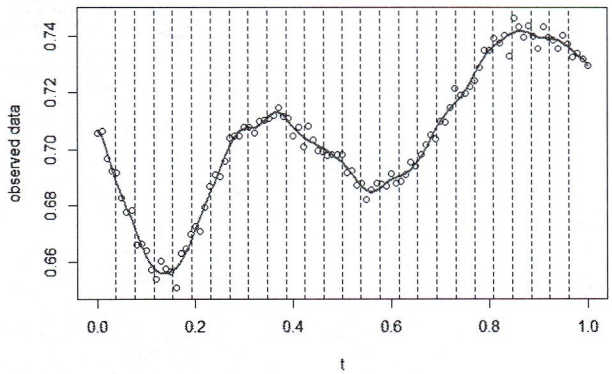
```
# Undersmoothing
nbasis <- 30

basister <- create.bspline.basis(c(0,1), nbasis, m)
```

```
x11()
par(mfrow=c(1,1))
plot(basister)
```



```
basismatter <- eval.basis(abscissa, basister)
Xsp0ter <- basismatter %*% lsfit(basismatter, Xobs0, intercept=FALSE)$coef

basismat1ter <- eval.basis(abscissa, basister,Lfdobj=1)
Xsp1ter <- basismat1ter %*% lsfit(basismatter, Xobs0, intercept=FALSE)$coef

basismat2ter <- eval.basis(abscissa, basister,Lfdobj=2)
Xsp2ter <- basismat2ter %*% lsfit(basismatter, Xobs0, intercept=FALSE)$coef
```

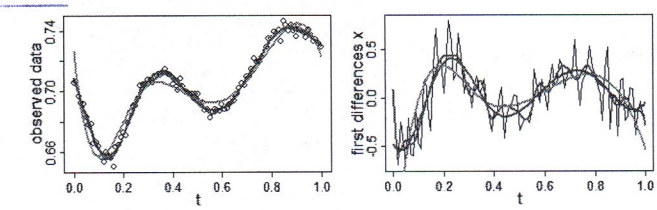
```
x11()
par(mfrow=c(1,1))
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0ter ,type="l",col="red",lwd=2)
abline(v=basister$params,lty=2)
```



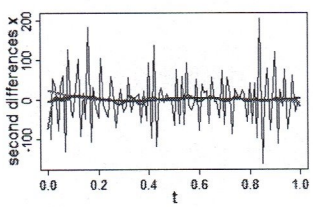

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsp0ter ,type="l",col="red",lwd=2)
points(abscissa,Xsp0bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsp1ter ,type="l",col="red",lwd=2)
points(abscissa,Xsp1bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsp2ter ,type="l",col="red",lwd=2)
points(abscissa,Xsp2bis ,type="l",col="green",lwd=2)
points(abscissa,Xsp2 ,type="l",col="blue",lwd=2)
```

How to chose the correct number of basis? Cross-validation!

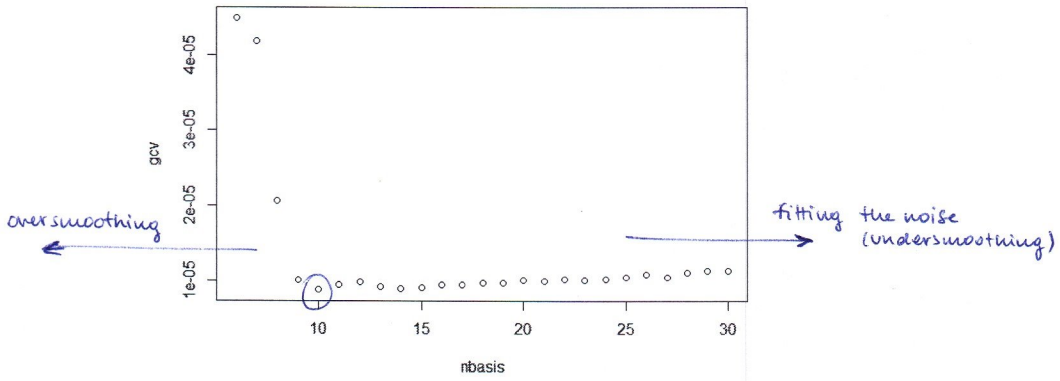
```
# generalized cross-validation
nbasis <- 6:30
gcv <- numeric(length(nbasis))
for (i in 1:length(nbasis)){
  basis <- create.bspline.basis(c(0,1), nbasis[i], m)
  gcv[i] <- smooth.basis(abscissa, Xobs0, basis)$gcv
}
```



(see colors)
the new one is terrible!
(is much noisier : is increasing the noise in the curve)




```
x11()
par(mfrow=c(1,1))
plot(nbasis,gcv)
```



oversmoothing

fitting the noise (undersmoothing)

```
nbasis[which.min(gcv)]
```

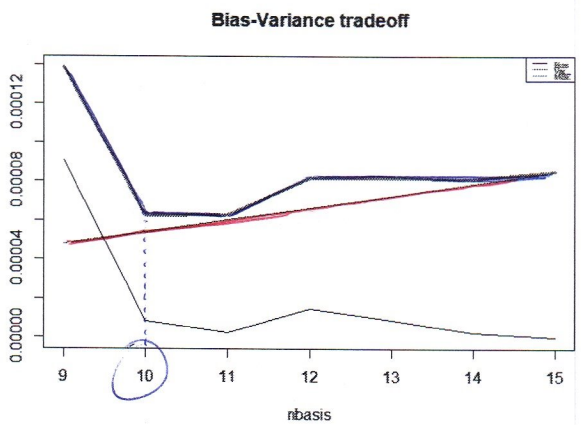
```
## [1] 10
```

```
### ------------------------------------------------------------------------
### Bias-Variance tradeoff
### ------------------------------------------------------------------------
sigma <- 0.003 # True sigma. Estimated before as sigmahat
nbasis <- 9:15
integrationinterval <- 11:90
bias <- rep(NA,len=length(nbasis))
var <- rep(NA,len=length(nbasis))
for (j in 1:length(nbasis)){
  basis <- create.bspline.basis(c(0,1), nbasis[j], m)
  basismat <- eval.basis(abscissa, basis)
  S <- basismat%*%solve(t(basismat)%*%basismat)%*%t(basismat)
  bias[j] <- sum((truecurve$X0vera-S%*%truecurve$X0vera)[integrationinterval])
  var[j] <- (sigma^2)*sum(diag(S[integrationinterval,integrationinterval]))
}
mse <- var+bias^2
```

we exclude the boundaries (because of problems)

we create a for-loop and we consider a number of basis from 9 to 15:

```
x11()
par(mfrow=c(1,1))
plot(nbasis,bias^2,ylim=c(0,max(mse)),type="l",ylab="",main="Bias-Variance tradeoff")
points(nbasis,var,col="red",type="l")
points(nbasis,mse,col="green",type="l",lwd=3)
legend('topright', c("Bias","Var","MSE"), col=c("black","red","green"),
       lty=1, cex=.5)
```



: MSE
: Var
: Bias

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### SMOOTHING SPLINES
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------

breaks <- abscissa[((0:50)*2)+1]

basis <- create.bspline.basis(breaks, norder=m)
functionalPar <- fdPar(fdobj=basis, Lfdobj=3, lambda=1e-8)
# functional parameter, having arguments:
# basis, order of the derivative to be penalized, smoothing parameter.

Xss <- smooth.basis(abscissa, Xobs0, functionalPar)

Xss0 <- eval.fd(abscissa, Xss$fd, Lfd=0)
Xss1 <- eval.fd(abscissa, Xss$fd, Lfd=1)
Xss2 <- eval.fd(abscissa, Xss$fd, Lfd=2)

df <- Xss$df   # the degrees of freedom in the smoothing curve
df
```

we add a penalization term

this time we specify where we want the breaks

defines the penalization: we specify:
- the basis
- order of the derivative that we want to penalize
- lambda (multiplying factor of the penalization term)

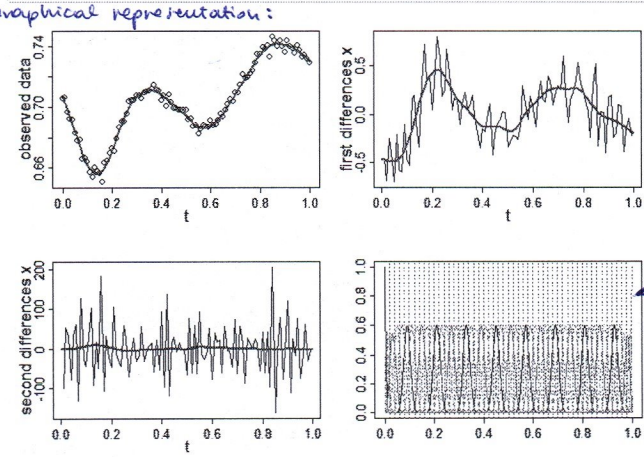
```
## [1] 17.08055
```

```
gcv <- Xss$gcv  # the value of the gcv statistic
gcv
```

```
##         rep1
## 9.016861e-06
```

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xss0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xss1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xss2 ,type="l",col="blue",lwd=2)
plot(basis)
```

Graphical representation:



! very fine grid, without a penalization we would get an overfitting curve

```
# change lambda: 1e-5
functionalParbis <- fdPar(fdobj=basis, Lfdobj=3, lambda=1e-5)

Xssbis <- smooth.basis(abscissa, Xobs0, functionalParbis)

Xss0bis <- eval.fd(abscissa, Xssbis$fd, Lfd=0)
Xss1bis <- eval.fd(abscissa, Xssbis$fd, Lfd=1)
Xss2bis <- eval.fd(abscissa, Xssbis$fd, Lfd=2)

dfbis <- Xssbis$df   #  the degrees of freedom in the smoothing curve
dfbis
```

```
## [1] 6.439156
```

← a stronger penalization (higher λ) reduces the degrees of freedom

```
gcvbis <- Xssbis$gcv  #  the value of the gcv statistic
gcvbis
```

```
##         rep1
## 3.478863e-05
```

```
# change lambda: 1e-12
functionalParter <- fdPar(fdobj=basis, Lfdobj=3, lambda=1e-12)

Xsster <- smooth.basis(abscissa, Xobs0, functionalParter)

Xss0ter <- eval.fd(abscissa, Xsster$fd, Lfd=0)
Xss1ter <- eval.fd(abscissa, Xsster$fd, Lfd=1)
Xss2ter <- eval.fd(abscissa, Xsster$fd, Lfd=2)

dfter <- Xsster$df   #  the degrees of freedom in the smoothing curve
dfter
```

```
## [1] 51.14587
```

```
gcvter <- Xsster$gcv  #  the value of the gcv statistic
gcvter
```

```
##         rep1
## 1.522785e-05
```

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xss0ter ,type="l",col="red",lwd=2)
points(abscissa,Xss0bis ,type="l",col="green",lwd=2)
points(abscissa,Xss0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xss1ter ,type="l",col="red",lwd=2)
points(abscissa,Xss1bis ,type="l",col="green",lwd=2)
points(abscissa,Xss1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xss2ter ,type="l",col="red",lwd=2)
points(abscissa,Xss2bis ,type="l",col="green",lwd=2)
points(abscissa,Xss2 ,type="l",col="blue",lwd=2)
```

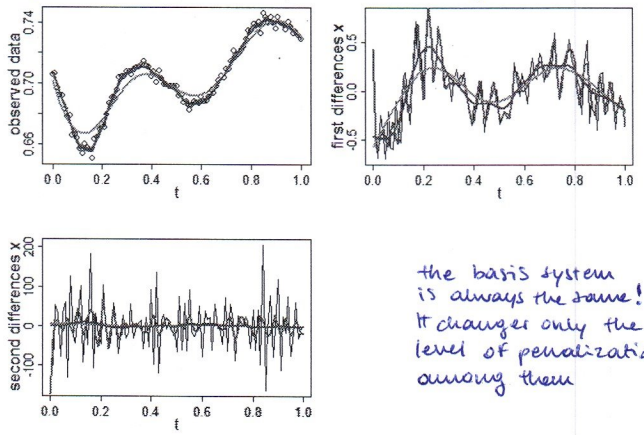
(*) →

```
# Recommendation: when choosing the smoothing parameter, look at the
# derivatives vs central finite differences
```

```
# generalized cross-validation
lambda <- c(1e-6,1e-7,1e-8,1e-9,1e-10,1e-11,1e-12)
gcv <- numeric(length(lambda))
for (i in 1:length(lambda)){
  functionalPar <- fdPar(fdobj=basis, Lfdobj=3, lambda=lambda[i])
  gcv[i] <- smooth.basis(abscissa, Xobs0, functionalPar)$gcv
}
```

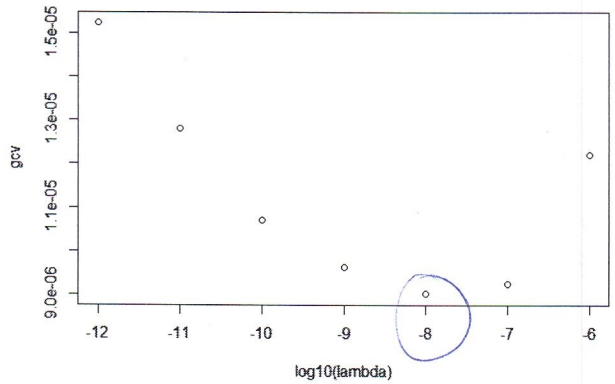
← for the optimal λ

Graphical representation of the changes of λ: (look at t with colors)



the basis system is always the same! It changes only the level of penalization (λ) among them

```
x11()
par(mfrow=c(1,1))
plot(log10(lambda),gcv)
```

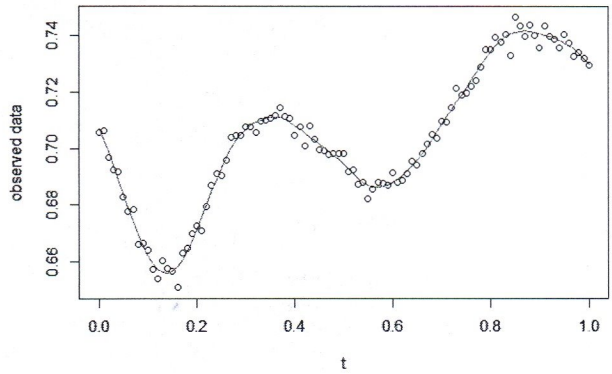



```
lambda[which.min(gcv)]
```

```
## [1] 1e-08
```

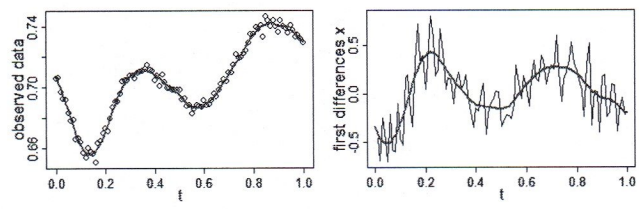
```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### LOCAL POLYNOMIAL REGRESSION
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------

library(KernSmooth)
```

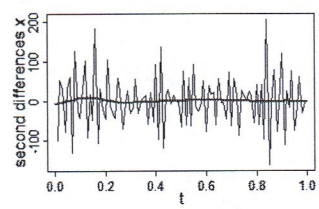
```
help(locpoly)

m <- 5          # order of the polynomial
degree <- m-1   # degree of the polynomial

bw <- 0.05 # bandwidth

Xsm0 <- locpoly(abscissa, Xobs0, degree=degree,
                bandwidth=bw, gridsize=length(abscissa),
                range.x=range(abscissa))
Xsm0 <- Xsm0$y

x11()
par(mfrow=c(1,1))
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsm0 ,type="l",col="blue")
```



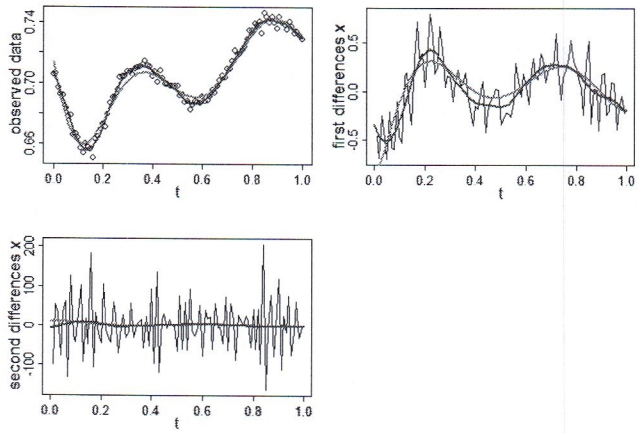

```
Xsm1 <- locpoly(abscissa,Xobs0,drv=1,degree=degree,bandwidth=bw,
                gridsize=length(abscissa), range.x=range(abscissa))
Xsm1 <- Xsm1$y

Xsm2 <- locpoly(abscissa,Xobs0,drv=2,degree=degree,bandwidth=bw,
                gridsize=length(abscissa), range.x=range(abscissa))
Xsm2 <- Xsm2$y
```

```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsm0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsm1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsm2 ,type="l",col="blue",lwd=2)

bw <- 0.15

Xsm0bis <- locpoly(abscissa,Xobs0,drv=0,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm0bis <- Xsm0bis$y

Xsm1bis <- locpoly(abscissa,Xobs0,drv=1,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm1bis <- Xsm1bis$y

Xsm2bis <- locpoly(abscissa,Xobs0,drv=2,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm2bis <- Xsm2bis$y
```

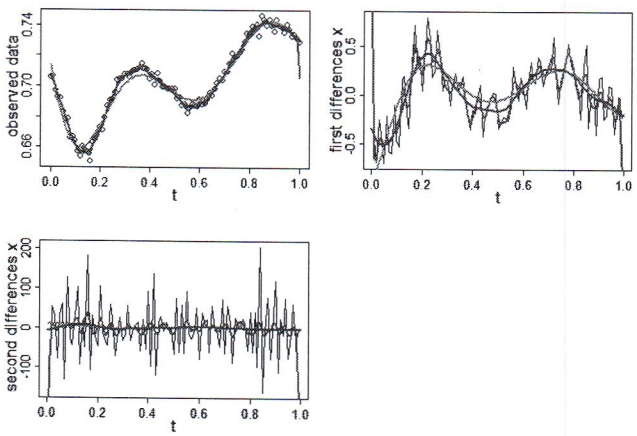





```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsm0bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsm1bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsm2bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm2 ,type="l",col="blue",lwd=2)

# a too large bandwidth leads to oversmoothing

# a too small bandwidth leads to undersmoothing
bw <- 0.015

Xsm0ter <- locpoly(abscissa,Xobs0,drv=0,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm0ter <- Xsm0ter$y

Xsm1ter <- locpoly(abscissa,Xobs0,drv=1,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm1ter <- Xsm1ter$y

Xsm2ter <- locpoly(abscissa,Xobs0,drv=2,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm2ter <- Xsm2ter$y
```




```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(abscissa,Xobs0,xlab="t",ylab="observed data")
points(abscissa,Xsm0ter ,type="l",col="red",lwd=2)
points(abscissa,Xsm0bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm0 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="t",ylab="first differences x",type="l")
points(abscissa,Xsm1ter ,type="l",col="red",lwd=2)
points(abscissa,Xsm1bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm1 ,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="t",ylab="second differences x",type="l")
points(abscissa,Xsm2ter ,type="l",col="red",lwd=2)
points(abscissa,Xsm2bis ,type="l",col="green",lwd=2)
points(abscissa,Xsm2 ,type="l",col="blue",lwd=2)

# Recommendation: when choosing the bandwidth, look at the
# derivatives vs central finite differences
```

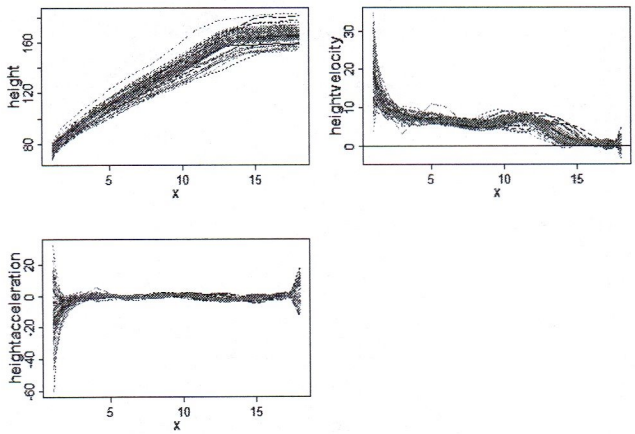



```
### ---------------------------------------------------------------------------
### Constrained functions
### ---------------------------------------------------------------------------
### SMOOTHING for positive curves
### ---------------------------------------------------------------------------
# y_j = exp(w(t_j)) + e_j

# f(t) = exp(w(t))

# The function w(t) is unconstrained
# The function f(t) is monotone increasing

# w(t) is modeled via a basis expansion
# numerical methods are used to compute the coefficients of the
# basis expansion

help(smooth.pos)

### ---------------------------------------------------------------------------
### SMOOTHING for monotone curves
### ---------------------------------------------------------------------------
# Example: Berkeley growth data

help(growth)
names(growth)
```

```
## [1] "hgtm" "hgtf" "age"
```

```
# If we neglet considering that the curves must be monotone...

age <- growth$age
heightbasis12 <- create.bspline.basis(rangeval = c(1,18), nbasis = 12, norder = 6)
basismat <- eval.basis(evalarg = growth$age, basisobj = heightbasis12)
heightmat <- growth$hgtf
heightcoef <- lsfit(x = basismat, y = heightmat, intercept=FALSE)$coef

height <- basismat %*% lsfit(basismat, heightmat, intercept=FALSE)$coef

basismat1 <- eval.basis(evalarg = growth$age, basisobj = heightbasis12,
                        Lfdobj=1)
heightvelocity <- basismat1 %*% lsfit(x = basismat, y = heightmat,
                                      intercept=FALSE)$coef

basismat2 <- eval.basis(evalarg = growth$age, basisobj = heightbasis12,Lfdobj=2)
heightacceleration <- basismat2 %*% lsfit(x=basismat, y= heightmat, intercept=FALSE)$coef
```

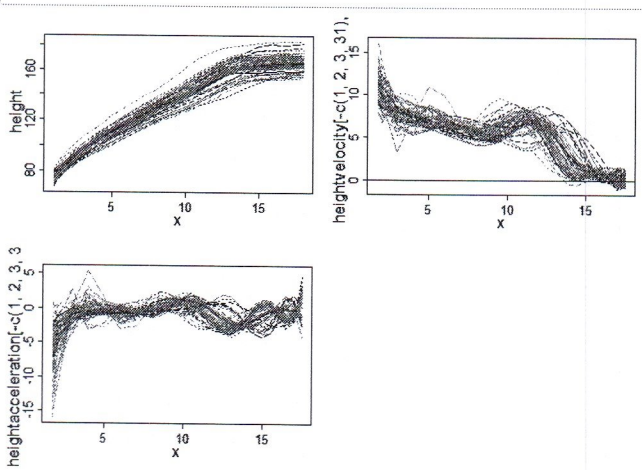
```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
matplot(age,height,type="l" )
matplot(age,heightvelocity,type="l" )
abline(h=0)
matplot(age,heightacceleration,type="l")
```




```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
matplot(age,height,type="l" )
matplot(age[-c(1,2,3,31)],heightvelocity[-c(1,2,3,31),],type="l" )
abline(h=0)
matplot(age[-c(1,2,3,31)],heightacceleration[-c(1,2,3,31),],type="l")

# a negative velocity does not make any sense (girls height does not
# decrease over this age interval)

# A model for monotone curves

# f(t) = int_(t_0)^t exp(w(u)) du
# y_j = b_0 + b_1 * f(t_j) + e_j

# The function w(t) is unconstrained
# The function f(t) is monotone increasing
# b_1>0 for monotone increasing functions
# b_1<0 for monotone decreasing functions
# b_0 is the value of the function at t_0

# w(t) is modeled via a basis expansion
# numerical methods are used to compute the coefficients of the
# basis expansion, as well as b_0, b_1

nage <- length(age)
ageRng <- range(age)
nfine <- 101
agefine <- seq(ageRng[1], ageRng[2], length=nfine)

# Let's consider only the first 5 girls

hgtf <- growth$hgtf[,1:5]
ncasef <- dim(hgtf)[2]

# We set up an order 6 spline basis with knots at ages of observations

norder <- 6
nbasis <- nage - 2 + norder
wbasis <- create.bspline.basis(rangeval = ageRng, nbasis = nbasis,
                               norder = norder, breaks = age)

# We construct the functional parameter with penalization of the third
# derivative

Lfdobj <- 3
lambda <- 10^(-0.5)
cvecf <- matrix(0, nbasis, ncasef) # this is used as initial value
                                   # for the numerical techniques
Wfd0 <- fd(coef = cvecf, basisobj = wbasis)
growfdPar <- fdPar(fdobj = Wfd0, Lfdobj = Lfdobj, lambda = lambda)

# We carry out a monotone smoothing
help(smooth.monotone)

growthMon <- smooth.monotone(argvals = age, y = hgtf, WfdParobj = growfdPar)
```

```
##
##
##  Results for curve 1
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        49.7544      36.7926      87.6342      5.0963
## 1        5.5785      4.0678      83.0646      4.613
## 2        0.4035      1.3208      76.591      4.4491
## 3        0.1899      0.3375      76.6529      4.4333
## 4        0.1527      0.1251      76.7122      4.4239
## 5        0.1479      0.0449      76.7118      4.4234
## 6        0.1478      0.0044      76.7079      4.4237
##
##
##  Results for curve 2
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        36.0196      32.9827      85.3473      5.606
## 1        5.008      1.5033      81.7036      5.2088
## 2        0.6301      2.273      74.3752      5.0816
## 3        0.3729      0.1844      74.2868      5.0734
## 4        0.3681      0.0561      74.2572      5.0749
## 5        0.3676      0.0431      74.1785      5.0801
## 6        0.3675      0.012      74.1721      5.0806
##
##
##  Results for curve 3
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        82.053      50.2742      92.0217      5.1538
## 1        6.8776      3.9204      84.4092      4.225
## 2        1.2419      3.7958      79.0688      3.977
## 3        0.4031      0.4305      78.9335      3.9435
## 4        0.3605      0.1483      78.9343      3.9384
## 5        0.3522      0.1167      78.9177      3.9382
## 6        0.3512      0.0192      78.8967      3.9392
## 7        0.3511      0.0316      78.8756      3.9404
##
##
##  Results for curve 4
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        39.5042      34.6945      88.1391      5.5471
## 1        4.7497      2.4998      83.2311      5.1368
## 2        0.4734      2.3284      77.3968      4.9863
## 3        0.1575      0.3493      77.4024      4.9703
## 4        0.127      0.0414      77.4343      4.9651
## 5        0.1266      0.0092      77.4302      4.9654
##
##
##  Results for curve 5
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        29.6498      30.5824      85.4322      5.7836
## 1        4.1793      4.4275      82.7034      5.5988
## 2        0.2755      0.3147      76.2669      5.5647
## 3        0.2318      0.1816      76.2504      5.5636
## 4        0.2242      0.0683      76.2109      5.5657
## 5        0.2237      0.0505      76.1757      5.5682
## 6        0.2235      0.0308      76.1481      5.5702
```

```
Wfd <- growthMon$Wfd
betaf <- growthMon$beta
hgtfhatfd <- growthMon$yhatfd

velocfdUN <- deriv.fd(expr = hgtfhatfd, Lfdobj = 1)
velocmeanfdUN <- mean.fd(velocfdUN)

accelfdUN <- deriv.fd(expr = hgtfhatfd, Lfdobj = 2)
accelmeanfdUN <- mean.fd(accelfdUN)
```

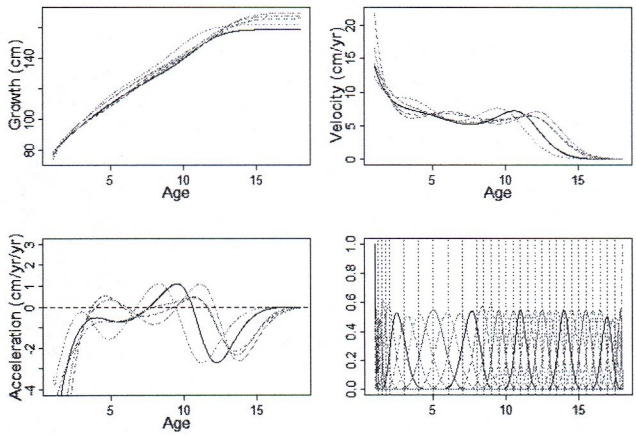



```
x11()
par(mfrow=c(2,2),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)
plot(hgtfhatfd, xlim=c(1,18), lty=1, lwd=2,
     cex=2, xlab="Age", ylab="Growth (cm)")
```

```
plot(velocfdUN, xlim=c(1,18),  lty=1, lwd=2,
     cex=2, xlab="Age", ylab="Velocity (cm/yr)")
```

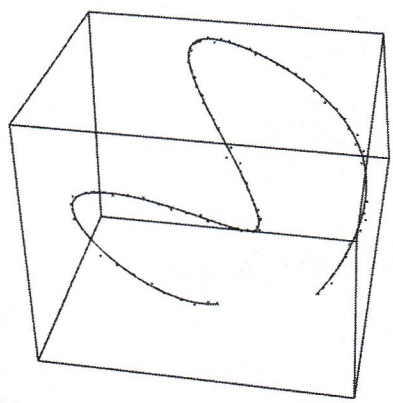
```
plot(accelfdUN, xlim=c(1,18), ylim=c(-4,3), lty=1, lwd=2,
     cex=2, xlab="Age", ylab="Acceleration (cm/yr/yr)")
```

```
plot(wbasis)
```



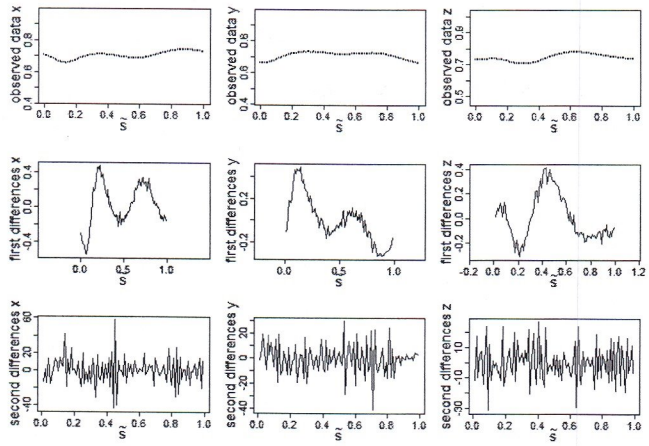

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Extension to multidimensional curves
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------

noisycurve3D <- read.table("noisycurve3D.txt",header=T)
Xobs0 <- noisycurve3D$X0
Yobs0 <- noisycurve3D$Y0
Zobs0 <- noisycurve3D$Z0
obs0 <- rbind(Xobs0,Yobs0,Zobs0)
abscissa <- noisycurve3D$Abscissa
NT <- length(abscissa)

truecurve3D <- read.table("truecurve3D.txt",header=T)
Xtrue0 <- truecurve3D$X0
Ytrue0 <- truecurve3D$Y0
Ztrue0 <- truecurve3D$Z0
true0 <- rbind(Xtrue0,Ytrue0,Ztrue0)

library(rgl)

open3d()
```

```
lines3d(t(true0[1,]),t(true0[2,]),t(true0[3,]),xlab="",ylab="",zlab="",size=3,axes=F)
points3d(t(obs0[1,]),t(obs0[2,]),t(obs0[3,]),xlab="",ylab="",zlab="",size=2,axes=F,pch=19,cex=2)
box3d()

a = scene3d()
rgl.close()
x11()
rglwidget(a)
```



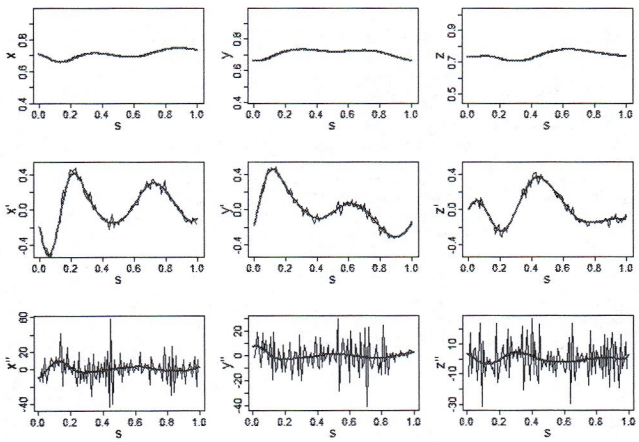
```
# compute the difference quotient
rappincX1 <- (Xobs0[3:NT]-Xobs0[1:(NT-2)])/(abscissa[3:NT]-abscissa[1:(NT-2)])
rappincY1 <- (Yobs0[3:NT]-Yobs0[1:(NT-2)])/(abscissa[3:NT]-abscissa[1:(NT-2)])
rappincZ1 <- (Zobs0[3:NT]-Zobs0[1:(NT-2)])/(abscissa[3:NT]-abscissa[1:(NT-2)])

rappincX2 <- ((Xobs0[3:NT]-Xobs0[2:(NT-1)])/(abscissa[3:NT]-abscissa[2:(NT-1)])-(Xobs0[2:(NT-1)]-Xobs0[1:(NT-2)])/(abscissa[
2:(NT-1)]-abscissa[1:(NT-2)]))*2/(abscissa[3:(NT)]-abscissa[1:(NT-2)])
rappincY2 <- ((Yobs0[3:NT]-Yobs0[2:(NT-1)])/(abscissa[3:NT]-abscissa[2:(NT-1)])-(Yobs0[2:(NT-1)]-Yobs0[1:(NT-2)])/(abscissa[
2:(NT-1)]-abscissa[1:(NT-2)]))*2/(abscissa[3:(NT)]-abscissa[1:(NT-2)])
rappincZ2 <- ((Zobs0[3:NT]-Zobs0[2:(NT-1)])/(abscissa[3:NT]-abscissa[2:(NT-1)])-(Zobs0[2:(NT-1)]-Zobs0[1:(NT-2)])/(abscissa[
2:(NT-1)]-abscissa[1:(NT-2)]))*2/(abscissa[3:(NT)]-abscissa[1:(NT-2)])
```

```
x11()
par(mfrow=c(3,3),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)

plot(abscissa,obs0[1,],xlab=expression(tilde(s)),ylab="observed data x",cex=0.1,asp=1)
plot(abscissa,obs0[2,],xlab=expression(tilde(s)),ylab="observed data y",cex=0.1,asp=1)
plot(abscissa,obs0[3,],xlab=expression(tilde(s)),ylab="observed data z",cex=0.1,asp=1)
plot(abscissa[2:(NT-1)],rappincX1,xlab=expression(tilde(s)),ylab="first differences x",type="l",asp=1)
plot(abscissa[2:(NT-1)],rappincY1,xlab=expression(tilde(s)),ylab="first differences y",type="l",asp=1)
plot(abscissa[2:(NT-1)],rappincZ1,xlab=expression(tilde(s)),ylab="first differences z",type="l",asp=1)
plot(abscissa[2:(NT-1)],rappincX2,xlab=expression(tilde(s)),ylab="second differences x",type="l")
plot(abscissa[2:(NT-1)],rappincY2,xlab=expression(tilde(s)),ylab="second differences y",type="l")
plot(abscissa[2:(NT-1)],rappincZ2,xlab=expression(tilde(s)),ylab="second differences z",type="l")
```



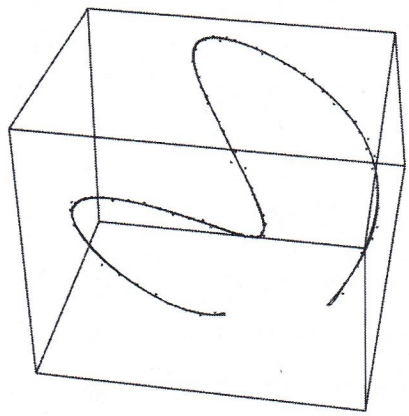

```
bw <- 0.05

Xsm0 <- locpoly(abscissa,Xobs0,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm0 <- Xsm0$y

Xsm1 <- locpoly(abscissa,Xobs0,drv=1,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm1 <- Xsm1$y

Xsm2 <- locpoly(abscissa,Xobs0,drv=2,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Xsm2 <- Xsm2$y

Ysm0 <- locpoly(abscissa,Yobs0,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Ysm0 <- Ysm0$y

Ysm1 <- locpoly(abscissa,Yobs0,drv=1,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Ysm1 <- Ysm1$y

Ysm2 <- locpoly(abscissa,Yobs0,drv=2,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Ysm2 <- Ysm2$y

Zsm0 <- locpoly(abscissa,Zobs0,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Zsm0 <- Zsm0$y

Zsm1 <- locpoly(abscissa,Zobs0,drv=1,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Zsm1 <- Zsm1$y

Zsm2 <- locpoly(abscissa,Zobs0,drv=2,degree=degree,bandwidth=bw,gridsize=length(abscissa), range.x=range(abscissa))
Zsm2 <- Zsm2$y
```

```
x11()
par(mfrow=c(3,3),mar=c(6,5,2,1),mex=0.6, mgp=c(2.2,0.7,0),pty="m", font.main=1,font.lab=1, font.axis=1,cex.lab=1.3,cex.axis=
1)

plot(abscissa,obs0[1,],xlab="s",ylab="x",cex=0.1,asp=1,xlim=c(0,1))
points(abscissa,Xsm0,type="l",col="blue",lwd=2)
plot(abscissa,obs0[2,],xlab="s",ylab="y",cex=0.1,asp=1,xlim=c(0,1))
points(abscissa,Ysm0,type="l",col="blue",lwd=2)
plot(abscissa,obs0[3,],xlab="s",ylab="z",cex=0.1,asp=1,xlim=c(0,1))
points(abscissa,Zsm0,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX1,xlab="s",ylab="x'",type="l",ylim=c(-0.5,0.5),xlim=c(0,1))
points(abscissa,Xsm1,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincY1,xlab="s",ylab="y'",type="l",ylim=c(-0.5,0.5),xlim=c(0,1))
points(abscissa,Ysm1,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincZ1,xlab="s",ylab="z'",type="l",ylim=c(-0.5,0.5),xlim=c(0,1))
points(abscissa,Zsm1,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincX2,xlab="s",ylab="x''",type="l")
points(abscissa,Xsm2,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincY2,xlab="s",ylab="y''",type="l")
points(abscissa,Ysm2,type="l",col="blue",lwd=2)
plot(abscissa[2:(NT-1)],rappincZ2,xlab="s",ylab="z''",type="l")
points(abscissa,Zsm2,type="l",col="blue",lwd=2)
```




```
open3d()
```

```
lines3d(t(true0[1,]),t(true0[2,]),t(true0[3,]),xlab="",ylab="",zlab="",size=3,axes=F)
points3d(t(obs0[1,]),t(obs0[2,]),t(obs0[3,]),size=2,pch=19,cex=2)
lines3d(t(Xsm0),t(Ysm0),t(Zsm0),size=3,col="blue")
box3d()
a = scene3d()
rgl.close()
x11()
rglwidget(a)
```



```
# or we may use another among the techniques seen above
```

FDA: Functional PCA

TOPICS:

- Functional Principal Component Analysis

```
library(fda)
```

```
###
### First dataset: canadian weather
###
# daily temperatures recorded in 35 weather stations of Canada
# (data are averages over 35 years - 1960 to 1994)
help(CanadianWeather)
```

(smoothing) →

Remember: smoothing is not trying to solve prediction problem, it's not trying to solve time series analysis, is just smoothing:
we believe that there is a continuous process that generates data observations over time.
It's sort of pre-processing data.

```
data_W <- CanadianWeather$dailyAv[,,1]
head(data_W)
```

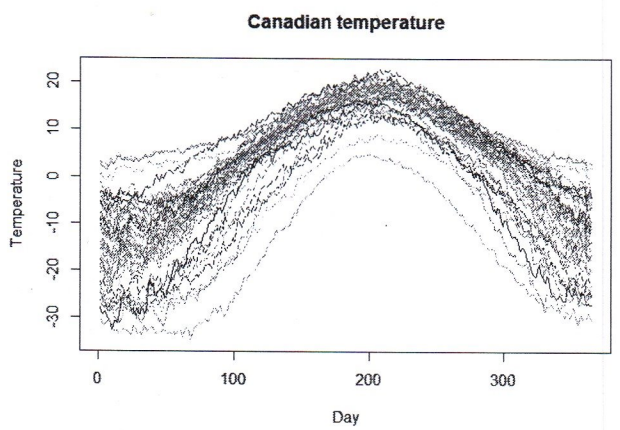
```
##        St. Johns Halifax Sydney Yarmouth Charlottvl Fredericton Scheffervll
## jan01      -3.6    -4.4   -3.8     -1.4       -5.8        -7.9       -22.5
## jan02      -3.1    -4.2   -3.5     -1.6       -5.6        -7.5       -23.0
## jan03      -3.4    -5.3   -4.6     -2.5       -7.3        -9.3       -23.0
## jan04      -4.4    -5.4   -5.0     -2.3       -7.0        -8.7       -21.8
## jan05      -2.9    -5.6   -4.1     -2.4       -6.7        -9.1       -23.5
## jan06      -4.5    -7.1   -6.1     -3.7       -8.9       -10.9       -24.4
##        Arvida Bagottville Quebec Sherbrooke Montreal Ottawa Toronto London
## jan01  -14.1       -14.6 -10.8      -10.1     -8.7   -9.4    -5.4   -5.2
## jan02  -14.4       -14.7 -11.4      -10.5     -9.9  -10.4    -5.4   -5.7
## jan03  -15.0       -15.5 -12.5      -11.4     -9.8   -9.9    -5.0   -5.3
## jan04  -14.3       -14.3 -11.2      -10.2     -9.0   -9.2    -5.9   -5.8
## jan05  -16.2       -15.8 -12.0      -11.3     -9.8  -10.1    -5.7   -6.5
## jan06  -16.3       -16.9 -13.2      -12.6    -10.7  -11.0    -5.6   -5.6
##        Thunder Bay Winnipeg The Pas Churchill Regina Pr. Albert Uranium City
## jan01       -14.0    -17.9  -20.5    -24.9  -15.7       -19.3        -23.5
## jan02       -14.0    -16.9  -19.6    -25.0  -15.0       -18.0        -23.6
## jan03       -13.5    -17.7  -20.6    -26.5  -15.8       -18.5        -24.6
## jan04       -13.9    -18.5  -21.0    -26.7  -17.3       -19.4        -25.5
## jan05       -14.4    -18.2  -21.7    -26.7  -17.0       -20.3        -26.5
## jan06       -14.5    -18.4  -22.6    -28.5  -17.1       -20.3        -27.9
##        Edmonton Calgary Kamloops Vancouver Victoria Pr. George Pr. Rupert
## jan01    -12.6    -8.6     -5.3       2.3       3.0      -10.5        0.4
## jan02    -12.7    -8.4     -5.6       2.1       2.7      -11.4        0.5
## jan03    -14.2    -9.2     -6.5       1.9       2.7      -11.0       -0.2
## jan04    -14.8   -10.8     -6.8       2.0       2.9      -11.9       -0.6
## jan05    -15.4   -12.0     -7.3       1.6       2.6      -13.5       -1.0
## jan06    -15.1   -11.7     -7.9       1.4       2.1      -13.2       -0.6
##        Whitehorse Dawson Yellowknife Iqaluit Inuvik Resolute
## jan01      -17.6  -28.0       -24.5   -23.3  -26.3    -30.7
## jan02      -18.6  -28.9       -25.3   -24.0  -27.1    -30.6
## jan03      -18.6  -29.5       -26.1   -24.4  -27.8    -31.4
## jan04      -20.0  -29.0       -27.7   -24.7  -28.2    -31.9
## jan05      -20.9  -28.8       -27.8   -25.3  -27.2    -31.5
## jan06      -21.0  -29.9       -28.1   -26.3  -27.6    -31.1
```

```
dim(data_W)
```

```
## [1] 365  35
```

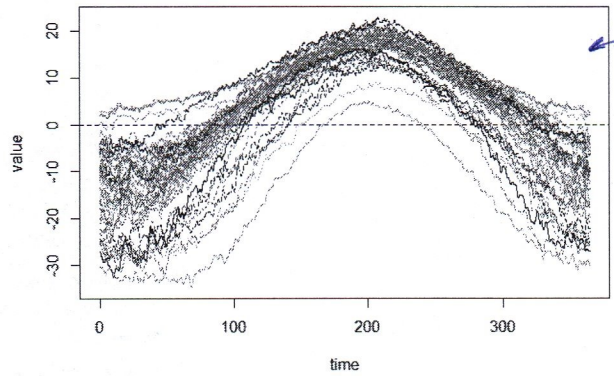
we consider 35 stations (n = 35 = # curves) and we measure the temperature over a year (365 days) in these stations

```
matplot(data_W,type='l',main='Canadian temperature',xlab='Day',ylab='Temperature')
```




```
# First of all we smooth the data. We choose a Fourier basis
# (periodic). We need to set the dimension of the basis

time <- 1:365

# Choice 1: we set a high dimensional basis (interpolating)
# Pros: no loss of information
# Cons: possible overfitting
basis.1 <- create.fourier.basis(rangeval=c(0,365),nbasis=365)
data_W.fd.1 <- Data2fd(y = data_W,argvals = time,basisobj = basis.1)
plot.fd(data_W.fd.1)
```

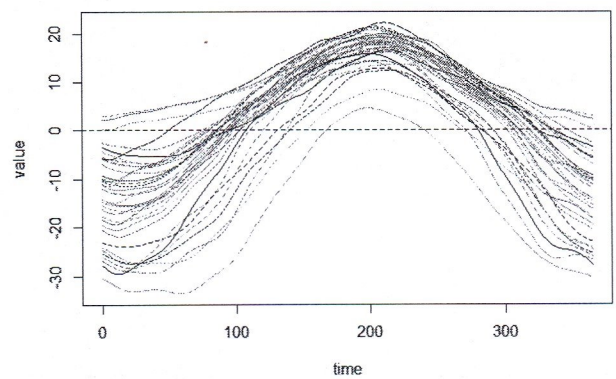
we've smoothing the whole dataset one curve at the time → we'll obtain 35 curves
(UNDERSMOOTHING)



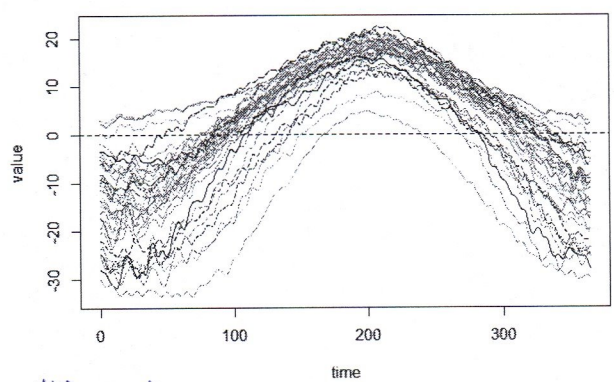
very close to the original data

```
# Choice 2: reduced dimensionality (we set a low dimensional basis)
# Pros: the data are much smoother and the measurement error is filtered
# Cons: I could have lost important information
basis.2 <- create.fourier.basis(rangeval=c(0,365),nbasis=21)
data_W.fd.2 <- Data2fd(y = data_W,argvals = time,basisobj = basis.2)
plot.fd(data_W.fd.2)
```

(OVERSMOOTHING)




```
# Choice 3: compromise between 1 and 2
basis.3 <- create.fourier.basis(rangeval=c(0,365),nbasis=109)
data_W.fd.3 <- Data2fd(y = data_W,argvals = time,basisobj = basis.3)
plot.fd(data_W.fd.3)
```



end of the smoothing part

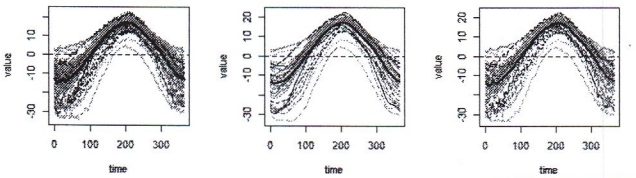
```
# estimate of the mean and of the covariance kernel
library(fields)
```

```
x11(width=10)
par(mfrow=c(2,3))

#mean
plot.fd(data_W.fd.1)
```

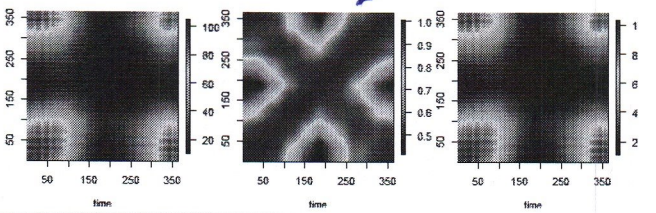
```
lines(mean.fd(data_W.fd.1),lwd=3)

plot.fd(data_W.fd.2)
```

```
lines(mean.fd(data_W.fd.2),lwd=2)

plot.fd(data_W.fd.3)
```

```
lines(mean.fd(data_W.fd.3),lwd=2)

# covariance
eval.1 <- eval.fd(time,data_W.fd.1)
image.plot(time,time,(cov(t(eval.1))[1:365,]))

eval.2 <- eval.fd(time,data_W.fd.2)
image.plot(time,time,(cor(t(eval.2))[1:365,]))

eval.3 <- eval.fd(time,data_W.fd.3)
image.plot(time,time,(cov(t(eval.3))[1:365,]))
```



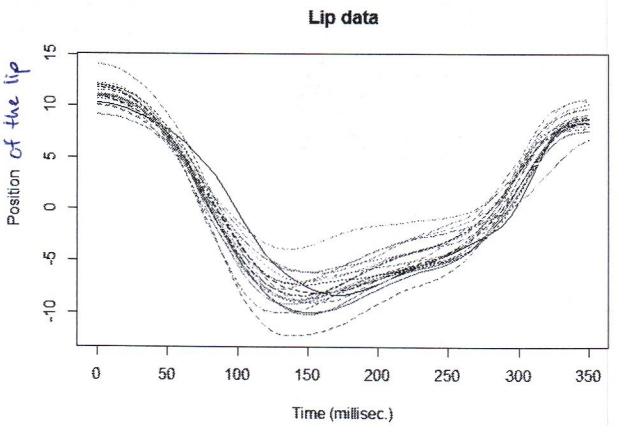
← the solid line is the pointwise mean



much more variability

smoothing affects the measure of covariability (3 different smoothings → 3 different images for the covariances)

```
###
### Second dataset: lip                    (smoothing)
###
# 51 measurements of the position of the lower lip every 7
# milliseconds for 20 repetitions of the syllable 'bob'.
help(lip)

data_L <- lip

time <- seq(0,350,by=7)
matplot(time,data_L,type='l',main='Lip data',ylab='Position',
        xlab='Time (millisec.)')
```




```
# smoothing. In this case, the data are already quite smooth.
# The smoothing does not pose problems

# Since the Fourier basis creates periodical data, it modifies the
# data near the boundaries of the domain, to make them periodical

basis <- create.fourier.basis(rangeval=c(0,350),nbasis=51)
data_L.fd <- Data2fd(data_L,time,basis)
```

What can go wrong?

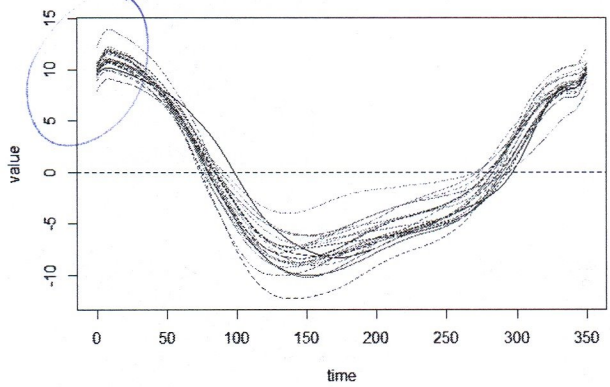
the boundaries!

```
## Swapping 'y' and 'argvals', because 'y' is  simpler,
##   and 'argvals' should be;  now  dim(argvals) =  51 ;  dim(y) =  51 x 20
```

```
plot.fd(data_L.fd, main="Fourier")
```

this happens because we're adapting boundaries for PERFECT PERIODICITY

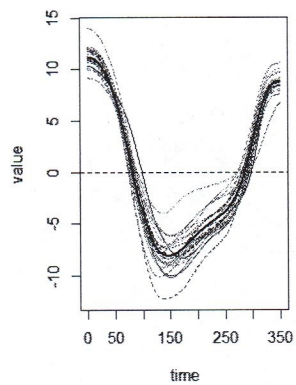
the effect of smoothing with fourier basis introduces some artificial constraints along the boundaries (constraints that are not satisfied by our data)

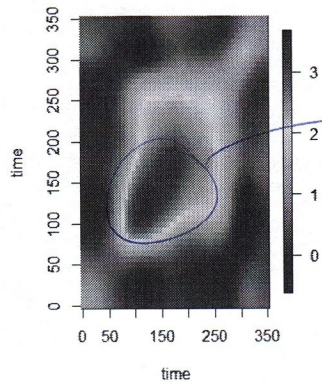



```
# Better to use a b-spline basis
basis <- create.bspline.basis(rangeval=c(0,350),nbasis=21)
data_L.fd <- Data2fd(y = data_L,argvals = time,basisobj = basis)
plot.fd(data_L.fd, main="B-splines")
```

```
# Estimate of the mean and of the covariance kernel

layout(cbind(1,2))
plot.fd(data_L.fd,xaxs='i')
```

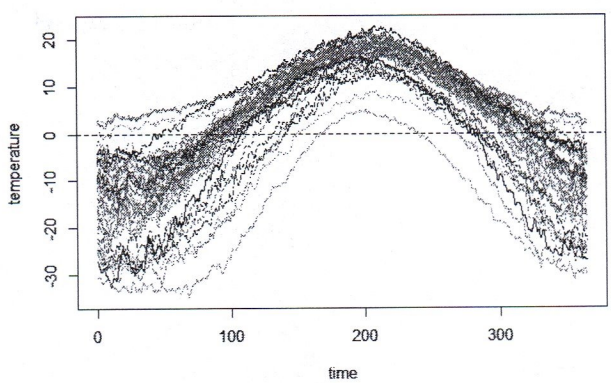
```
lines(mean.fd(data_L.fd),lwd=2)
eval <- eval.fd(time,data_L.fd)
image.plot(time, time, (cov(t(eval))[1:51,]))
```





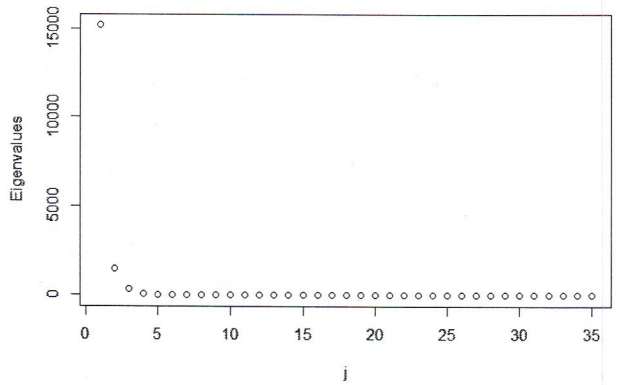
where there is higher variability we see more discrepancies among the functions

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### Functional PCA
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
help(pca.fd)

### ------------------------------------------------------------------------
### First dataset: canadian weather
### ------------------------------------------------------------------------
# interpolated data (Choice 1)
plot.fd(data_W.fd.1,ylab='temperature')
```



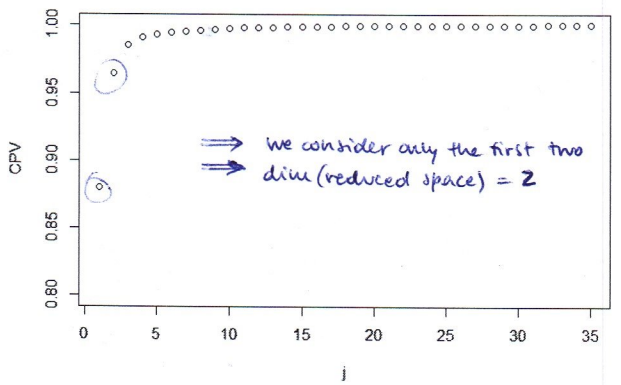
it computes the first 5 principal components

```
pca_W.1 <- pca.fd(data_W.fd.1,nharm=5,centerfns=TRUE)

# scree plot
# pca.fd computes all the 365 eigenvalues, but only the first
# N-1=34 are non-null
plot(pca_W.1$values[1:35],xlab='j',ylab='Eigenvalues')
```




```
plot(cumsum(pca_W.1$values)[1:35]/sum(pca_W.1$values),xlab='j',ylab='CPV',ylim=c(0.8,1))
```

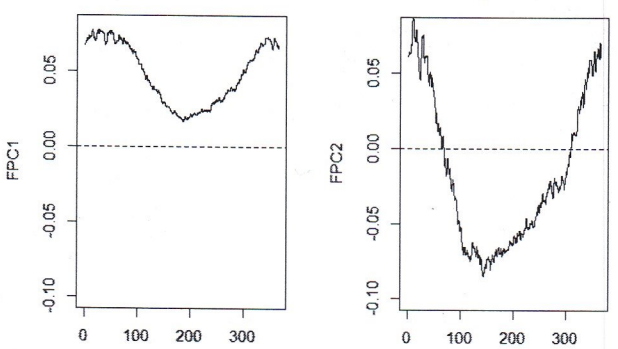
Cumulative proportion of variability:




```
# first two FPCs
layout(cbind(1,2))
plot(pca_W.1$harmonics[1,],col=1,ylab='FPC1',ylim=c(-0.1,0.08))
```

```
abline(h=0,lty=2)
plot(pca_W.1$harmonics[2,],col=2,ylab='FPC2',ylim=c(-0.1,0.08))
```

Loadings of the two first principal components:

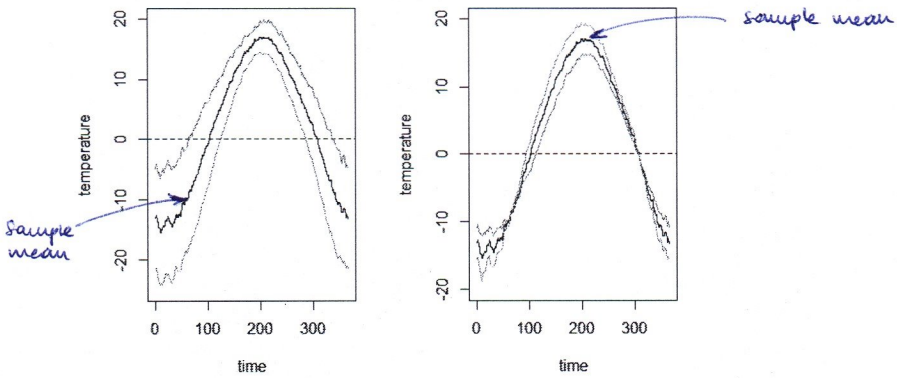



```
# plot of the FPCs as perturbation of the mean
media <- mean.fd(data_W.fd.1)

plot(media,lwd=2,ylim=c(-25,20),ylab='temperature',main='FPC1')
```

(more easy to interprete)

```
lines(media+pca_W.1$harmonics[1,]*sqrt(pca_W.1$values[1]), col=2)
lines(media-pca_W.1$harmonics[1,]*sqrt(pca_W.1$values[1]), col=3)
# variation in amplitude (more in winter than in summer)

plot(media,lwd=2,ylim=c(-20,20),ylab='temperature',main='FPC2')
```

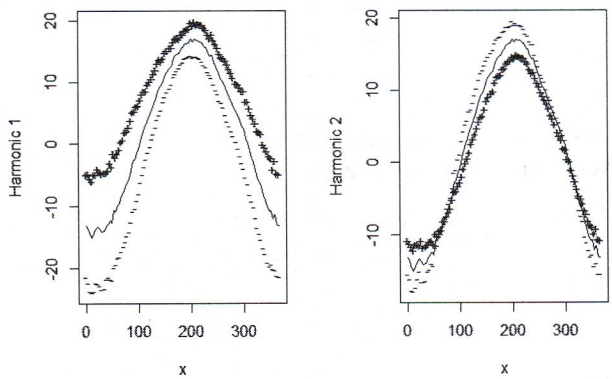
```
lines(media+pca_W.1$harmonics[2,]*sqrt(pca_W.1$values[2]), col=2)
lines(media-pca_W.1$harmonics[2,]*sqrt(pca_W.1$values[2]), col=3)
```




```
# temperate climate or not
```

```
# Command of the library fda that automatically does these plots
par(mfrow=c(1,2))
plot.pca.fd(pca_W.1, nx=100, pointplot=TRUE, harm=c(1,2), expand=0, cycle=FALSE)
```

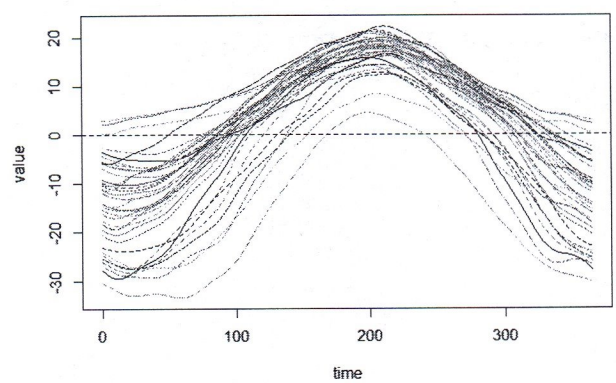
PCA function 1 (Percentage of variabilCA function 2 (Percentage of variability)



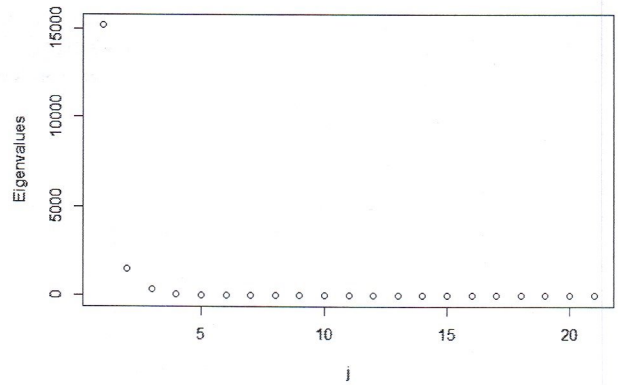
same plots as we saw in the theory

```
# ------------------------------------------------------------------------
# smooth data (Choice 2)
plot.fd(data_W.fd.2)
```

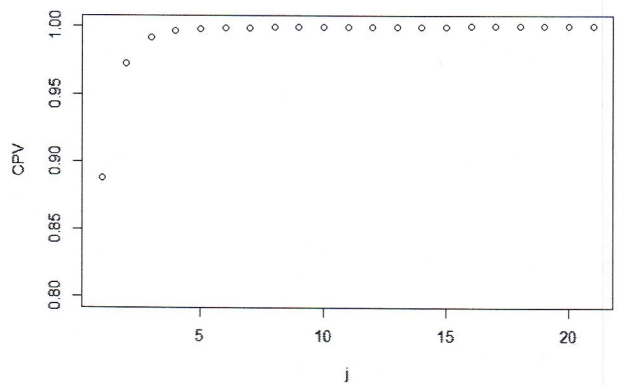
(same procedure)




```
pca_W.2 <- pca.fd(data_W.fd.2,nharm=5,centerfns=TRUE)

# scree plot
plot(pca_W.2$values,xlab='j',ylab='Eigenvalues')
```



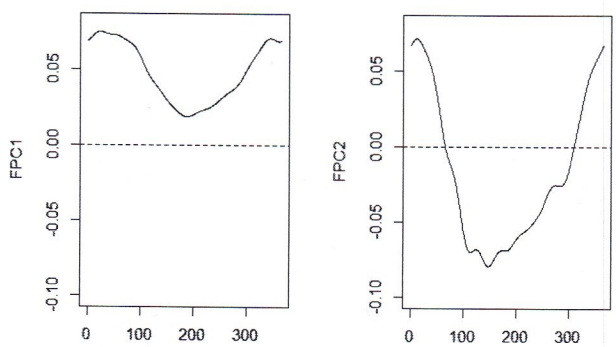

```
plot(cumsum(pca_W.2$values)/sum(pca_W.2$values),xlab='j',ylab='CPV',ylim=c(0.8,1))
```



similar behaviour as before
⟹ reduced space of dimension 2

```
# first two FPCs
layout(cbind(1,2))
plot(pca_W.2$harmonics[1,],col=1,ylab='FPC1',ylim=c(-0.1,0.08))
```

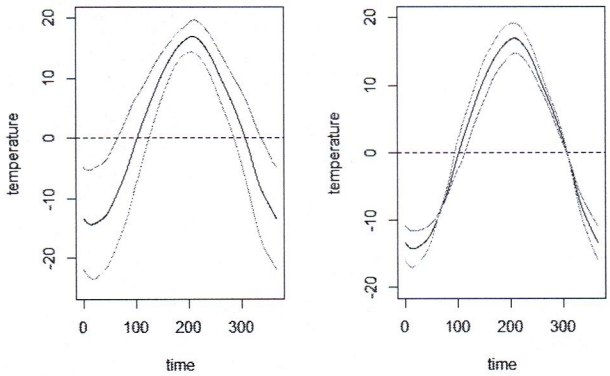
```
abline(h=0,lty=2)
plot(pca_W.2$harmonics[2,],col=2,ylab='FPC2',ylim=c(-0.1,0.08))
```




```
# plot of the FPCs as perturbation of the mean
media <- mean.fd(data_W.fd.2)

plot(media,lwd=2,ylim=c(-25,20),ylab='temperature',main='PC1')
```

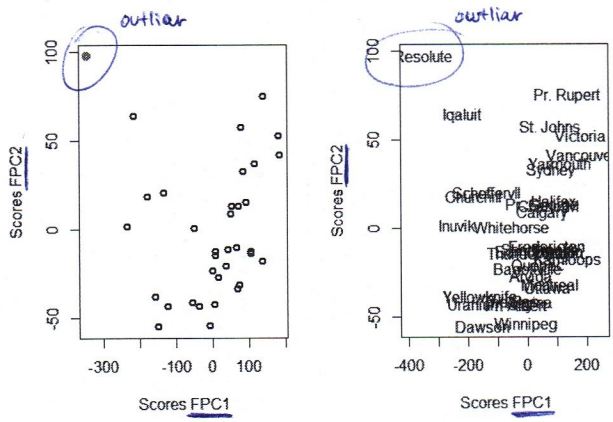
```
lines(media+pca_W.2$harmonics[1,]*sqrt(pca_W.2$values[1]), col=2)
lines(media-pca_W.2$harmonics[1,]*sqrt(pca_W.2$values[1]), col=3)

plot(media,lwd=2,ylim=c(-20,20),ylab='temperature',main='PC2')
```

```
lines(media+pca_W.2$harmonics[2,]*sqrt(pca_W.2$values[2]), col=2)
lines(media-pca_W.2$harmonics[2,]*sqrt(pca_W.2$values[2]), col=3)
```

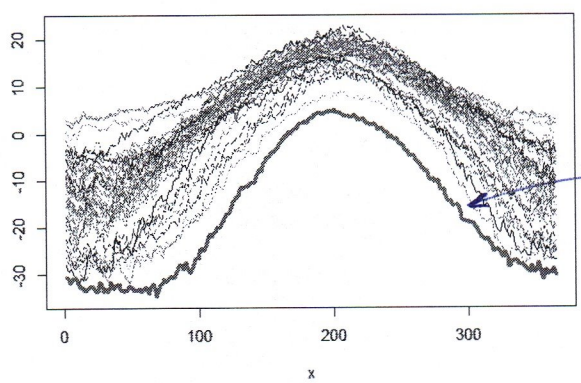


← similar but more smoothed (that's because here we oversmoothed)

```
# similar interpretations as before

# scatter plot of the scores
par(mfrow=c(1,2))
plot(pca_W.1$scores[,1],pca_W.1$scores[,2],xlab="Scores FPC1",ylab="Scores FPC2",lwd=2)
points(pca_W.1$scores[35,1],pca_W.1$scores[35,2],col=2, lwd=4)

plot(pca_W.1$scores[,1],pca_W.1$scores[,2],type="n",xlab="Scores FPC1",
     ylab="Scores FPC2",xlim=c(-400,250))
text(pca_W.1$scores[,1],pca_W.1$scores[,2],dimnames(data_W)[[2]], cex=1)
```

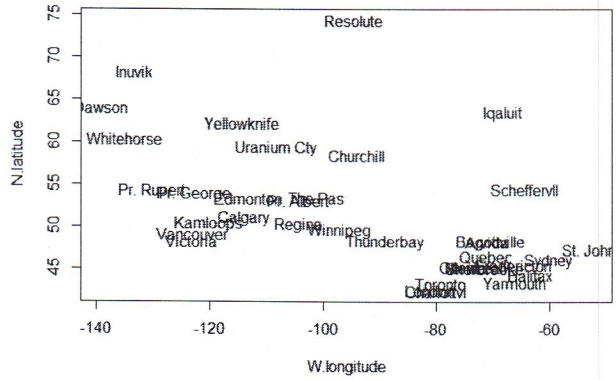



```
# outlier: Resolute (35)

layout(1)
matplot(eval.1,type='l')
lines(eval.1[,35],lwd=4, col=2) #temperature profile for Resolute
```

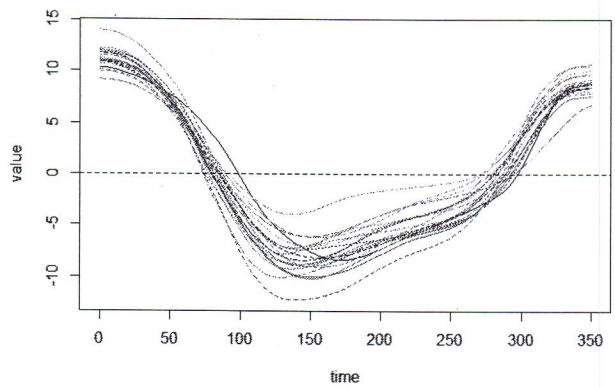


that's the curve of the outliar zone (also from the original representation we could notice)

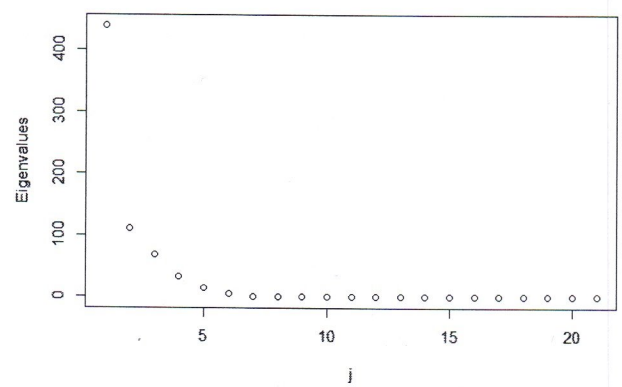
```
coord <- CanadianWeather$coordinates
coord[,2] <- -coord[,2]
plot(coord[,2:1],col=0)
text(coord[,2:1],rownames(coord))
```



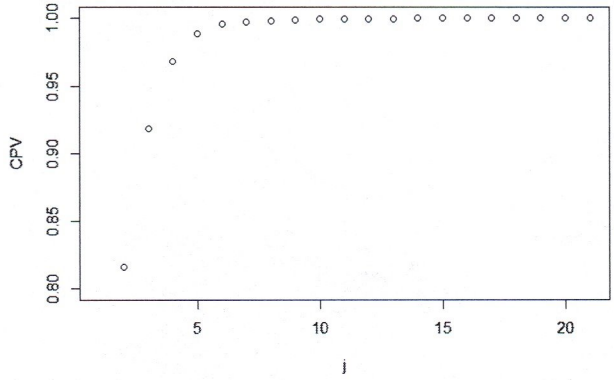

```
# Exercise: perform FPCA with Choice 3 of smoothing

### ------------------------------------------------------------------------
### Second dataset: Lip
### ------------------------------------------------------------------------
plot.fd(data_L.fd)
```




```
pca_L <- pca.fd(data_L.fd,nharm=5,centerfns=TRUE)

# scree plot
plot(pca_L$values,xlab='j',ylab='Eigenvalues')
```



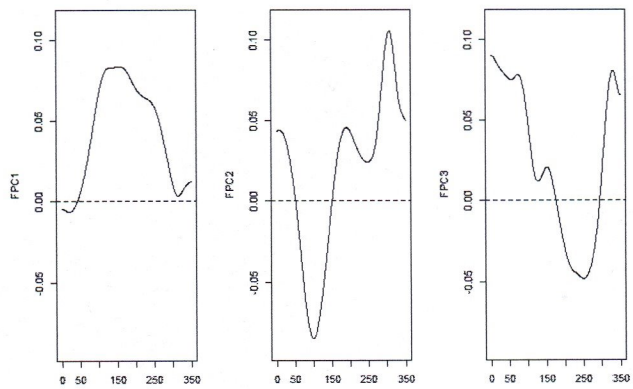

```
plot(cumsum(pca_L$values)/sum(pca_L$values),xlab='j',ylab='CPV',ylim=c(0.8,1))
```




```
# First three FPCs
layout(cbind(1,2,3))
plot(pca_L$harmonics[1,],col=1,ylab='FPC1',ylim=c(-0.09,0.11))
```

```
plot(pca_L$harmonics[2,],col=2,ylab='FPC2',ylim=c(-0.09,0.11))
```

```
plot(pca_L$harmonics[3,],col=3,ylab='FPC3',ylim=c(-0.09,0.11))
```

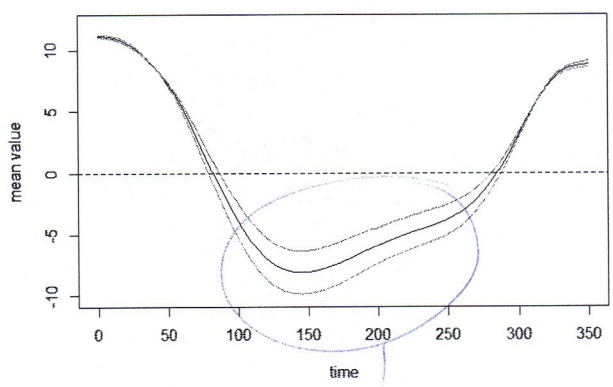



```
# plot of the principal components as perturbation of the mean
media <- mean.fd(data_L.fd)
```

```
plot(media,lwd=2,ylim=c(-10,12),main='FPC1')
```

```
lines(media+pca_L$harmonic[1,]*sqrt(pca_L$values[1]), col=2)
lines(media-pca_L$harmonic[1,]*sqrt(pca_L$values[1]), col=3)
```

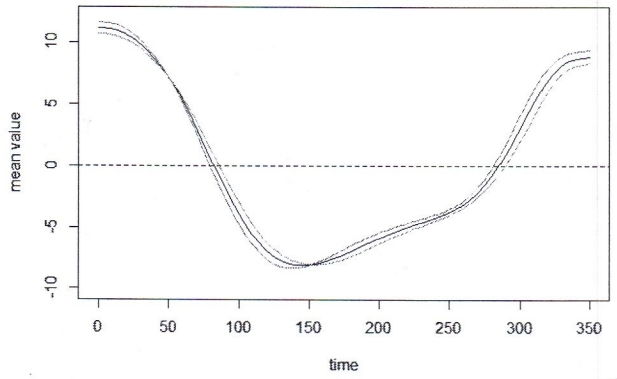
PC1:




```
# variation in amplitude in the centre
plot(media,lwd=2,ylim=c(-10,12),main='FPC2')
```

```
lines(media+pca_L$harmonic[2,]*sqrt(pca_L$values[2]), col=2)
lines(media-pca_L$harmonic[2,]*sqrt(pca_L$values[2]), col=3)
```

PC2:

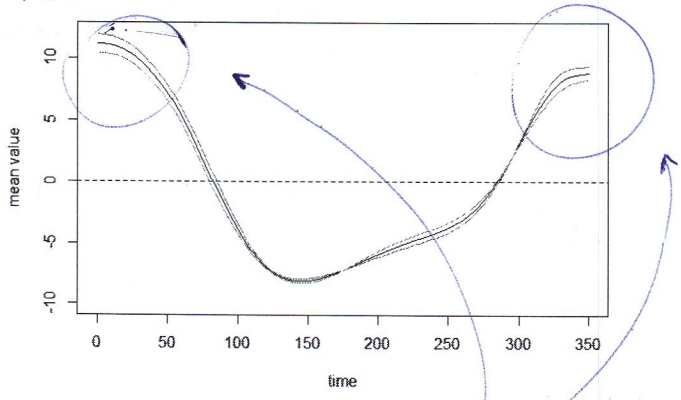


in colors: the red curve comes before the others, the green one comes as last

```
# horizontal translation - difference in timing

plot(media,lwd=2,ylim=c(-10,12),main='FPC3')
```

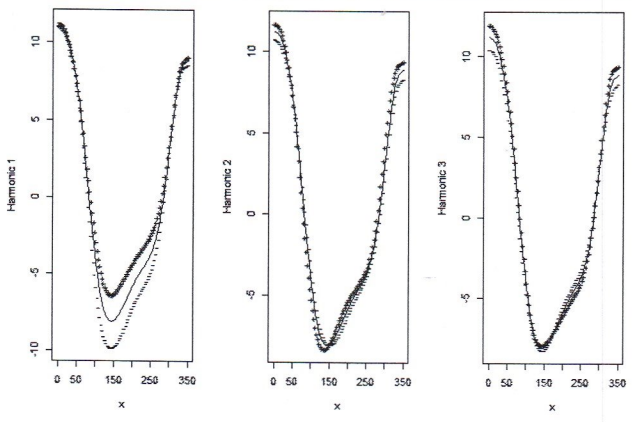
```
lines(media+pca_L$harmonic[3,]*sqrt(pca_L$values[3]), col=2)
lines(media-pca_L$harmonic[3,]*sqrt(pca_L$values[3]), col=3)
```

PC3:



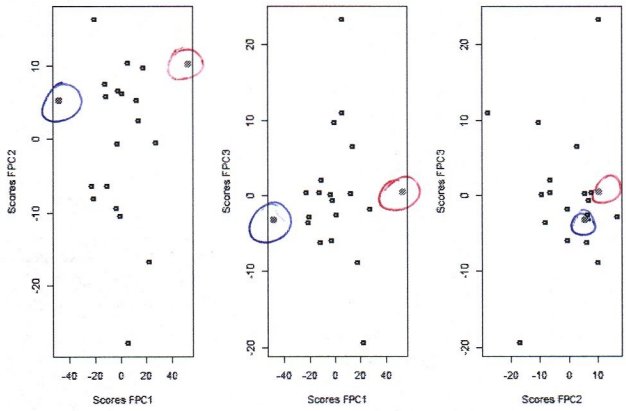

```
# variation in amplitude at the boundaries of the domain

# Command of the library fda that automatically does these plots
par(mfrow=c(1,3))
plot.pca.fd(pca_L, nx=100, pointplot=TRUE, harm=c(1,2,3), expand=0, cycle=FALSE)
```

function 1 (Percentage of variabilità function 2 (Percentage of variabilità function 3 (Percentage of variabili




```
# Scores
layout(cbind(1,2,3))
plot(pca_L$scores[,1],pca_L$scores[,2],xlab="Scores FPC1",ylab="Scores FPC2",lwd=2)
points(pca_L$scores[12,1],pca_L$scores[12,2],col=2, lwd=4)
points(pca_L$scores[9,1],pca_L$scores[9,2],col=3, lwd=4)
plot(pca_L$scores[,1],pca_L$scores[,3],xlab="Scores FPC1",ylab="Scores FPC3",lwd=2)
points(pca_L$scores[12,1],pca_L$scores[12,3],col=2, lwd=4)
points(pca_L$scores[9,1],pca_L$scores[9,3],col=3, lwd=4)
plot(pca_L$scores[,2],pca_L$scores[,3],xlab="Scores FPC2",ylab="Scores FPC3",lwd=2)
points(pca_L$scores[12,2],pca_L$scores[12,3],col=2, lwd=4)
points(pca_L$scores[9,2],pca_L$scores[9,3],col=3, lwd=4)
```

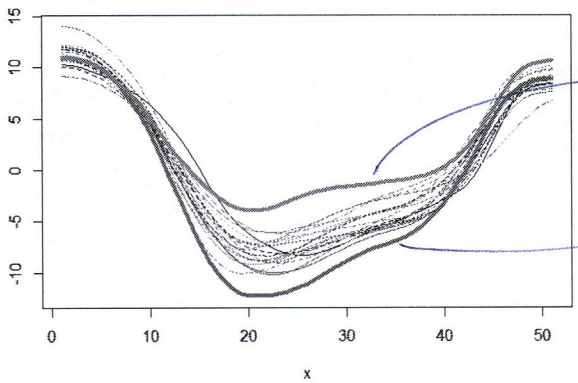
Now we're working in a 3D space:



projections of the two outliers

```
layout(1)
matplot(eval,type='l')
lines(eval[,12],lwd=4, col=2)
lines(eval[,9],lwd=4, col=3)
```

In the original data we see were are the outliers:



outlier corresponding to

outlier corresponding to

FDA: K-mean alignment

TOPICS:

- K-mean alignment

```
# Using the R package fdakma
library(fdakma)
help(kma)
```

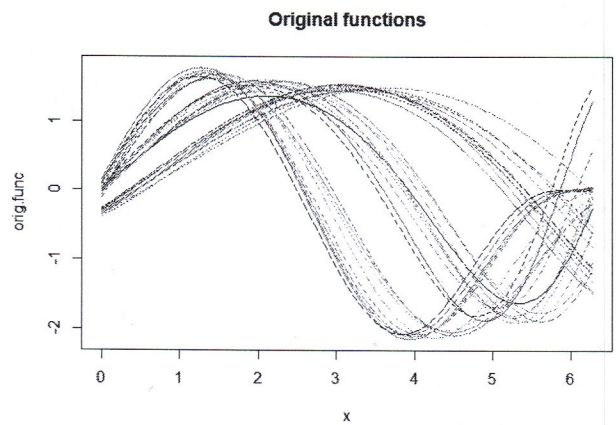
```
## starting httpd help server ... done
```

```
help(kma.data)

data(kma.data)
names(kma.data)
```

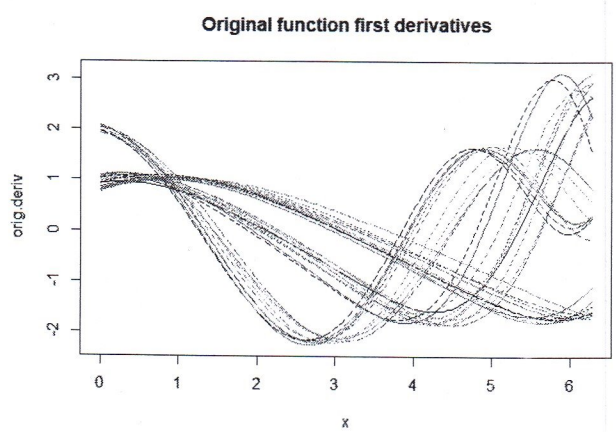
```
## [1] "x"  "y0" "y1"
```

: ascissa, value of the function, value of the first derivative of the function

```
x <- kma.data$x   # abscissas
y0 <- kma.data$y0 # evaluations of original functions
y1 <- kma.data$y1 # evaluations of original functions' first derivatives

# Plot of original functions
matplot(t(x),t(y0), type='l', xlab='x', ylab='orig.func')
title ('Original functions')
```




```
# There seems to be three clusters of functions obtained by warping the
# abscissas.

# Plot of original function first derivatives
matplot(t(x),t(y1), type='l', xlab='x', ylab='orig.deriv')
title ('Original function first derivatives')
```



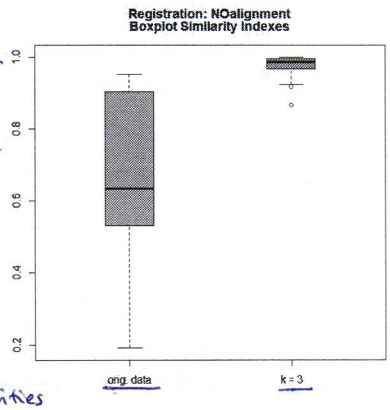

```
# Without alignment, let's try with 3 clusters

set.seed(4)
fdakma_example_noalign_0der <- kma(
  x=x, y0=y0, n.clust = 3,
  warping.method = 'NOalignment',
  similarity.method = 'd0.pearson',   # similarity computed as the cosine
                                      # between the original curves
                                      # (correlation)
  center.method = 'k-means'
  #,seeds = c(1,11,21) # you can give a little help to the algorithm...
)
```

ascissa, evaluation of the function, number of clusters

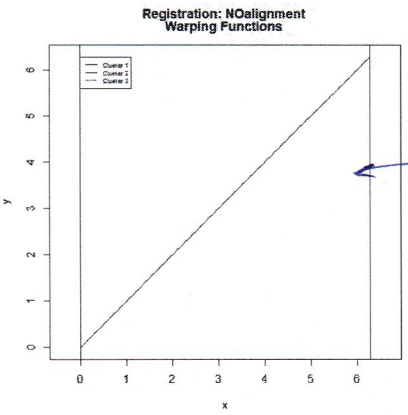
if we do not include this the algorithm will start with random generation of the centers of the clusters
Here we're providing the 3 curves (3 because we said that we wanted 3 clusters) that we want as centers of the cluster

```
kma.show.results(fdakma_example_noalign_0der)
knitr::include_graphics('./1.jpg')
```

comparison between similarities of the original data and of the result of the k-means alignment algorithm

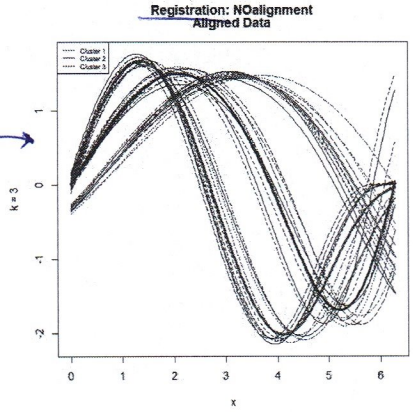


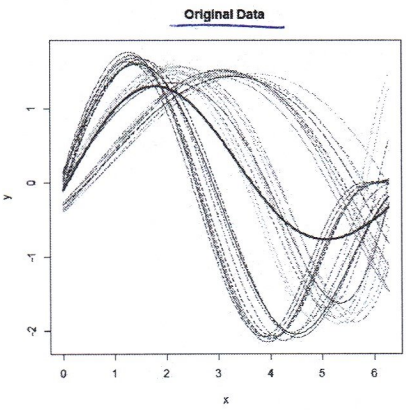
→ we can see that clustering in 3 groups provide an improvement between similarities



graphical representation of the warping functions (in this case is useless because we said "NO ALIGNMENT", so the warpings are the identity)

here we see the effect of the clustering →

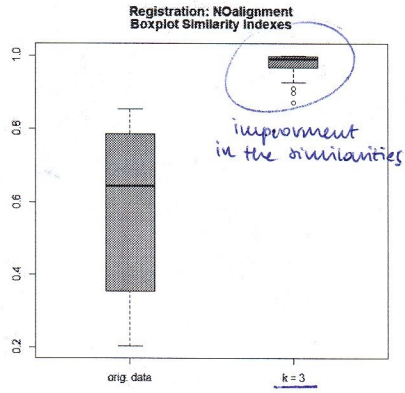




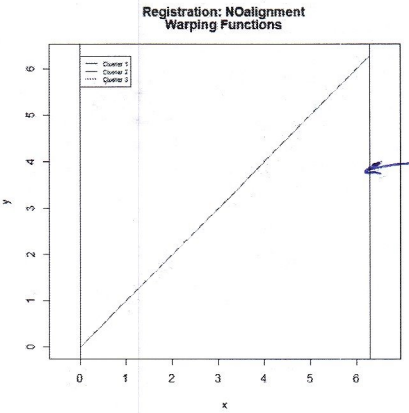

```
fdakma_example_noalign_0der$labels
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1
```

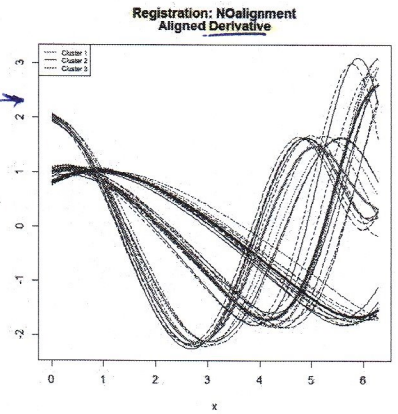
```
# We might want to use a similarity measure which considers first
# derivatives

set.seed(5)
fdakma_example_noalign_1der <- kma(
  x=x, y0=y0, y1=y1, n.clust = 3,
  warping.method = 'NOalignment',
  similarity.method = 'd1.pearson',   # similarity computed as the cosine
                                      # between the first derivatives
                                      # (correlation)
  center.method = 'k-means'
  #,seeds = c(1,11,21) # you can give a little help to the algorithm...
)

kma.show.results(fdakma_example_noalign_1der)
knitr::include_graphics('./2.jpg')
```



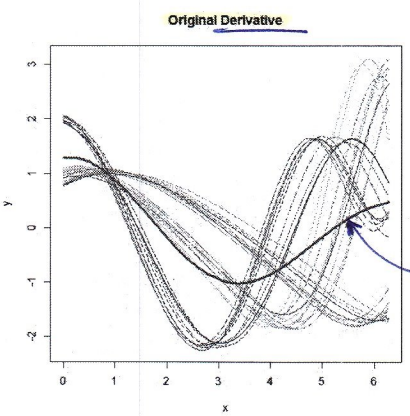
improvement in the similarities



again, we're not performing alignment

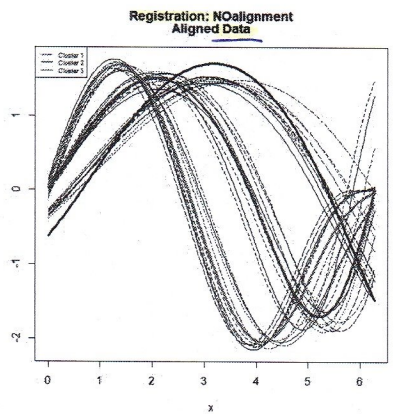


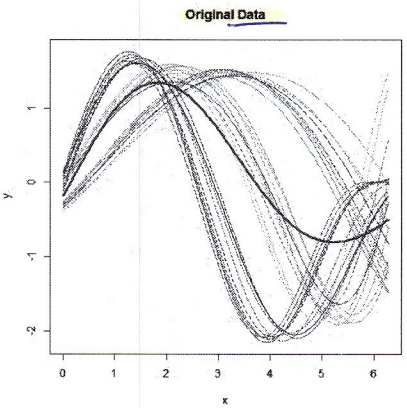
results of the clusters



the dissimilarities are computed with the first derivatives!

pointwise mean (which is not representative of the behaviour of the functions)





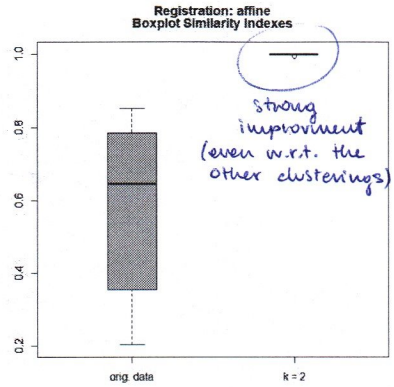

```
fdakma_example_noalign_1der$labels
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
```

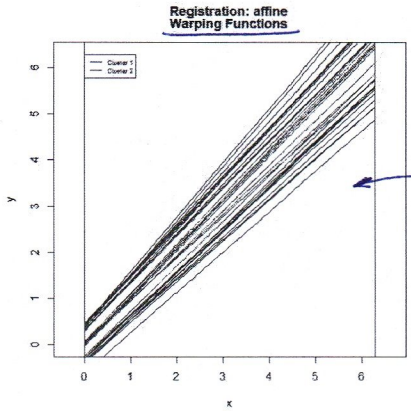
```
# Label switching aside, it is the same clustering

table(fdakma_example_noalign_0der$labels,
      fdakma_example_noalign_1der$labels, dnn=c("0der", "1der"))
```

```
##     1der
## 0der  1  2  3
##    1  0  0 10
##    2 10  0  0
##    3  0 10  0
```

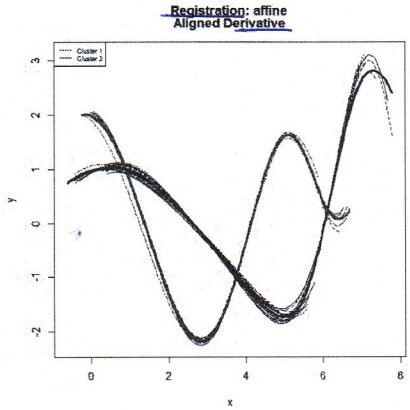
```
# Let's allow for alignment.
# Result of kma with 2 clusters,
# allowing affine transformation for the abscissas
# and considering 'd1.pearson' as similarity.method.

set.seed(1)
fdakma_example <- kma(
  x=x, y0=y0, y1=y1, n.clust = 2,
  warping.method = 'affine',
  similarity.method = 'd1.pearson',  # similarity computed as the cosine
                                     # between the first derivatives
                                     # (correlation)
  center.method = 'k-means'
  #seeds = c(1,21) # you can give a little help to the algorithm...
)

kma.show.results(fdakma_example)
knitr::include_graphics('./3.jpg')
```



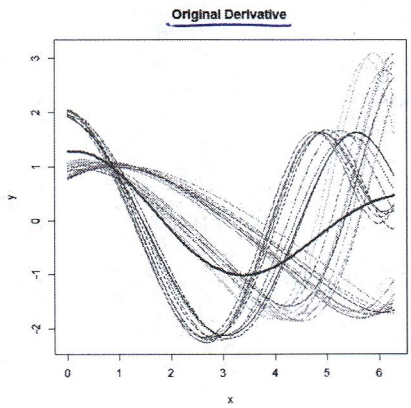
Strong improvement (even w.r.t. the other clusterings)

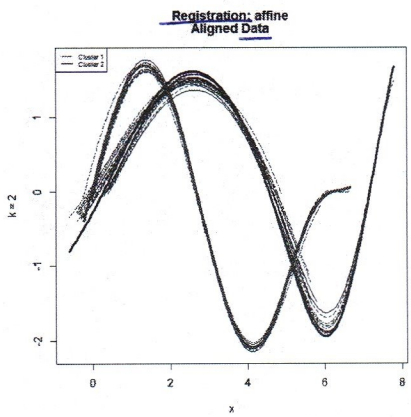


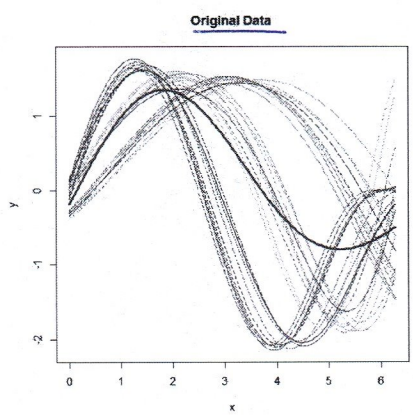
we see a structure → good (see with colors)



two means (one for cluster) are much more representative than only one for the whole dataset → it's good to cluster in two groups








```
# Much better: high within group similarity after alignment,
# the boxplot is concentrated on 1.

# Labels assigned to each function
fdakma_example$labels
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
table(fdakma_example_noalign_0der$labels,
      fdakma_example$labels,
      dnn=c("NOalign_0der_3groups", "Align_1der_2groups"))
```

```
##                    Align_1der_2groups
## NOalign_0der_3groups  1  2
##                    1 10  0
##                    2  0 10
##                    3  0 10
```

```
# Total shifts and dilations applied to the original
# abscissa to obtain the aligned abscissa
fdakma_example$shift
```

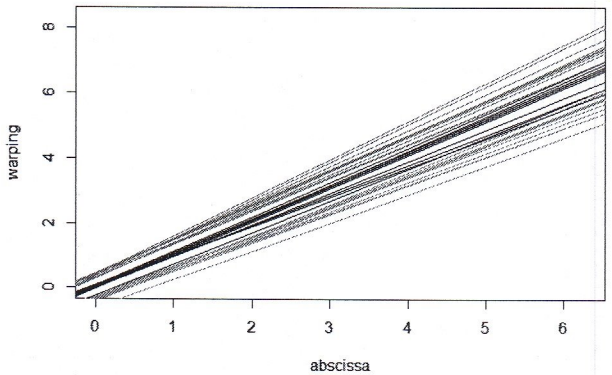
```
##  [1]  0.40993313  0.40057074  0.43414288  0.29886155  0.29556709  0.35441692
##  [7]  0.42501712  0.46387722  0.37143318  0.32735413 -0.29047380 -0.33532368
## [13] -0.41753874 -0.61190240 -0.24761012 -0.39542337 -0.30395247 -0.40057564
## [19] -0.41642727 -0.36194648  0.08353227 -0.01417597  0.05256659 -0.24455328
## [25]  0.06095024  0.07188835  0.02555842  0.01362030 -0.01847231 -0.03091461
```

```
fdakma_example$dilation
```

```
##  [1] 1.0555474 1.1095815 1.0595166 1.0922194 1.0565670 1.1639466 1.1717337
##  [8] 1.0602318 1.0114105 1.0158132 0.9397879 0.8733834 0.9607486 0.8733834
## [15] 0.9057688 0.9814290 0.9303596 0.8829472 0.9544560 0.9011687 0.9005752
## [22] 1.0224713 0.9269714 0.9746941 1.0559523 1.0276709 1.0265762 1.0305475
## [29] 0.9748072 1.0597339
```

```
# Warpings applied to the original data, colored according to
# membership to the 3 clusters obtained via k-means, no alignment,
# d0.pearson similarity
```

```
plot(x, type="n", xlim=c(min(x),max(x)), ylim=c(min(x),max(x)+2), xlab="abscissa", ylab="warping")
title("Alignment affinities")
for(i in 1:30){
  abline(a=fdakma_example$shift[i],b=fdakma_example$dilation[i],
         col=fdakma_example_noalign_0der$labels[i])
}
```



(See in colors)

⇒ 3 clusters with no alignment are coherent with 2 clusters with alignment

```
# How to choose the number of clusters and the warping method
help(kma.compare)
```

```
kma.compare_example <- kma.compare (
  x=x, y0=y0, y1=y1, n.clust = 1:3,
  warping.method = c('affine'),
  similarity.method = 'd1.pearson',
  center.method = 'k-means',
  seeds = c(1,21,30),
  plot.graph=1)
```

← the function is checking the number of clusters = 1, 2, 3

← if we say more than just "affine" then there will be more than one line (like in the theoretical slide)

```
knitr::include_graphics('./4.jpeg')
```