Smoothing pigs growth data

Søren Højsgaard

September 26, 2008

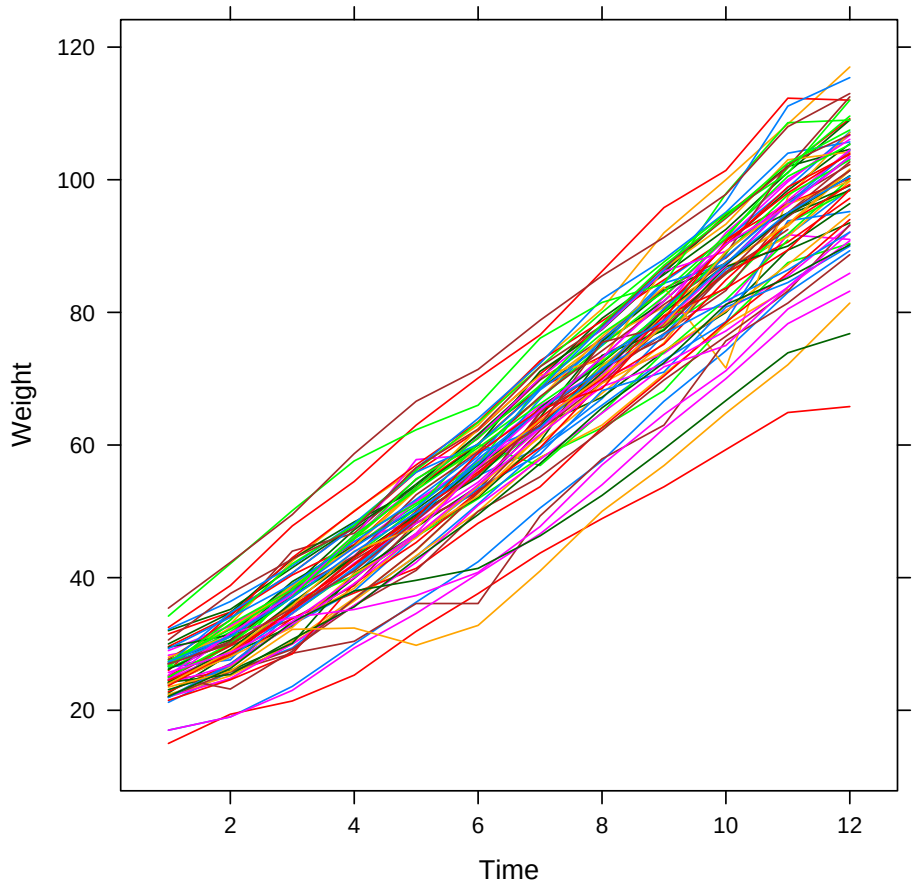
Contents

1 Growth of slaughter pigs

Pigs have been weighed every week during a 12–week period. There are some treatments involved, but these are ignored for now.

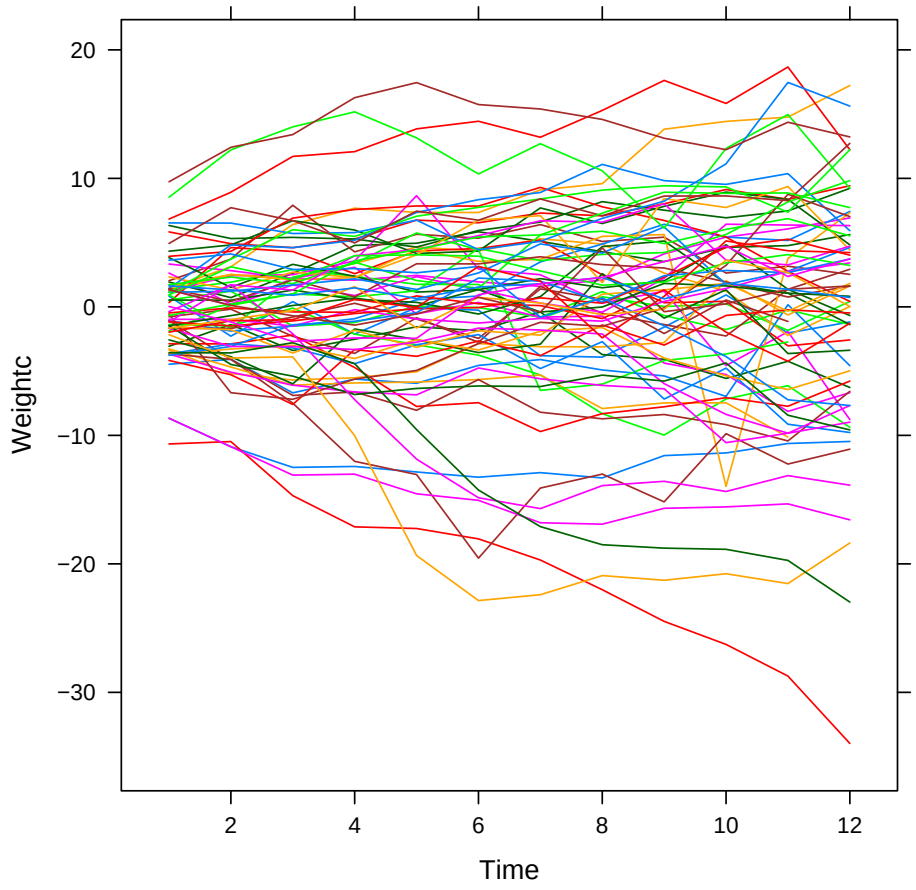
```
> library(fda)
> library(lattice)
> library(doBy)
> data(dietox)

> print(xyplot(Weight ~ Time, group = Pig, data = dietox,
+     type = "l"))
```



Hard to see structure so we center data by subtracting time–wise mean:

```
> grandmean <- fitted(lm(Weight ~ factor(Time), data = dietox))
> dietox$Weightc <- dietox$Weight - grandmean
> print(xyplot(Weightc ~ Time, group = Pig, data = dietox,
+      type = "l"))
```



Actually, for some pigs there were only measuremens on 11 weeks. We discard these. To simplify graphics we only look at those pigs with no vitamin E added to their feed.

```
> s <- splitBy(~Pig, data = dietox)
> usepig <- as.numeric(names(which(unlist(lapply(s, nrow)) ==
+     12)))
> dietox <- subset(dietox, Pig %in% usepig & Evit == 1)
> wgt <- matrix(dietox$Weight, nrow = 12)
```

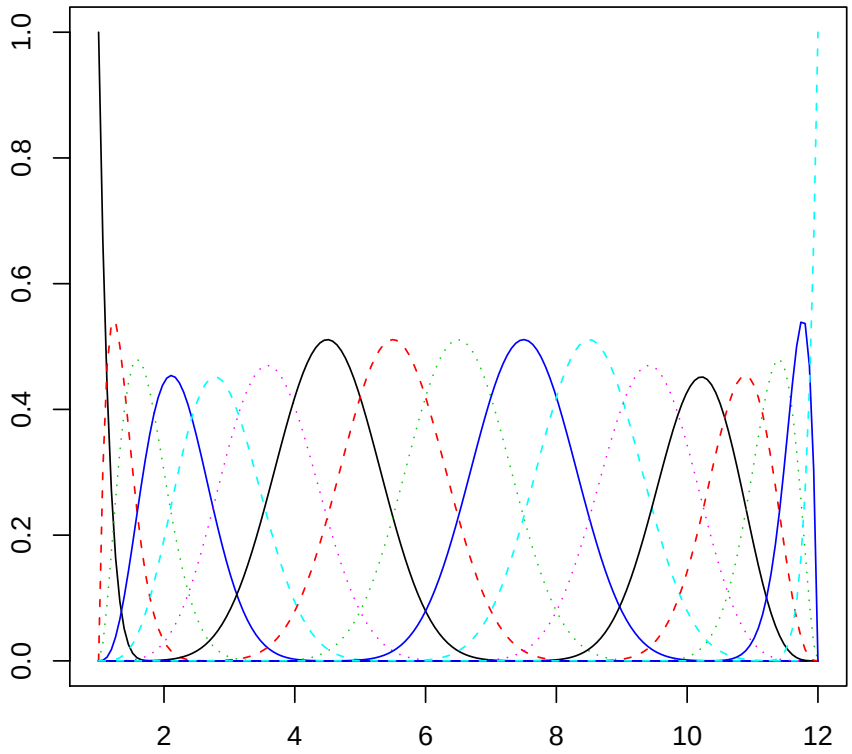
Define generic variables

```
> TVAR <- 1:12
> YVAR <- wgt
```

Create B-spline basis: [[JIM; there is a limit on the number of basis functions...]]

```
> norder <- 5
> nbasis <- length(TVAR) + norder
> rng <- range(TVAR)
```

```
> rngfine <- seq(rng[1], rng[2], length = 100)
> knots <- TVAR
> wbasis <- create.bspline.basis(rng, nbasis, norder, knots)
> plot(wbasis)
```



Create initial vector:

```
> cvec0 <- matrix(0, nbasis, 1)
```

Create functional data object:

```
> Wfd0 <- fd(cvec0, wbasis)
```

Penalize curvature of acceleration:

```
> Lfdobj <- 3
```

Set smoothing parameter:

```
> lambda <- 5
```

Define functional parameter object:

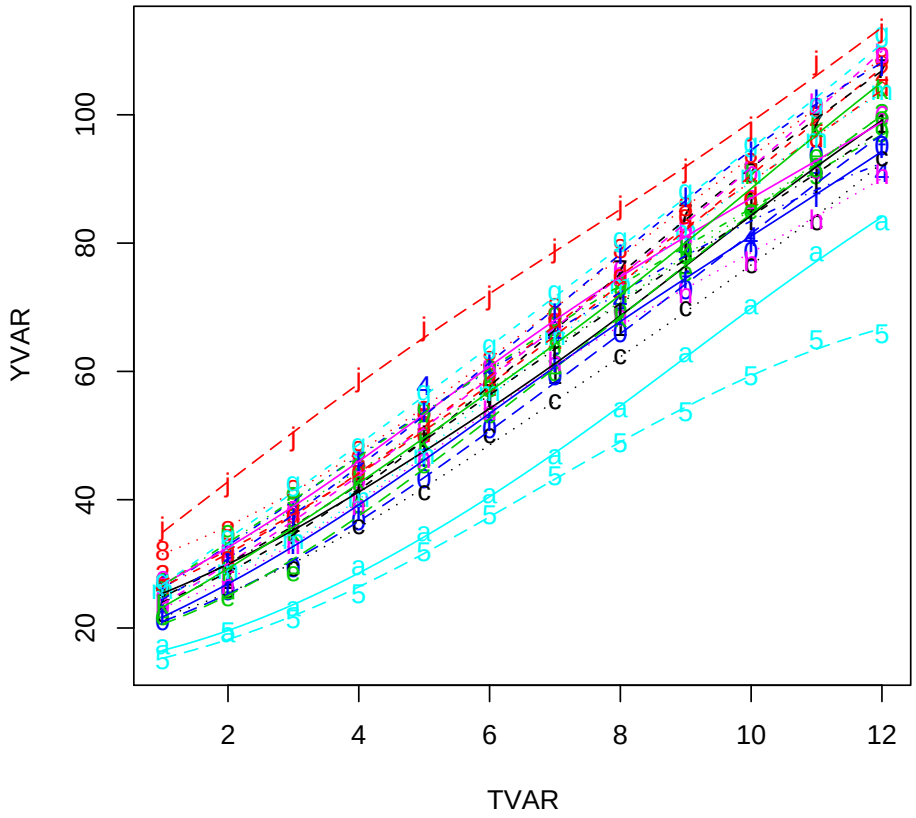
```
> xxxfdPar <- fdPar(Wfd0, Lfdobj, lambda)
```

Do the smoothing:

```
> fdobj <- smooth.basis(TVAR, YVAR, xxxfdPar)$fd
```

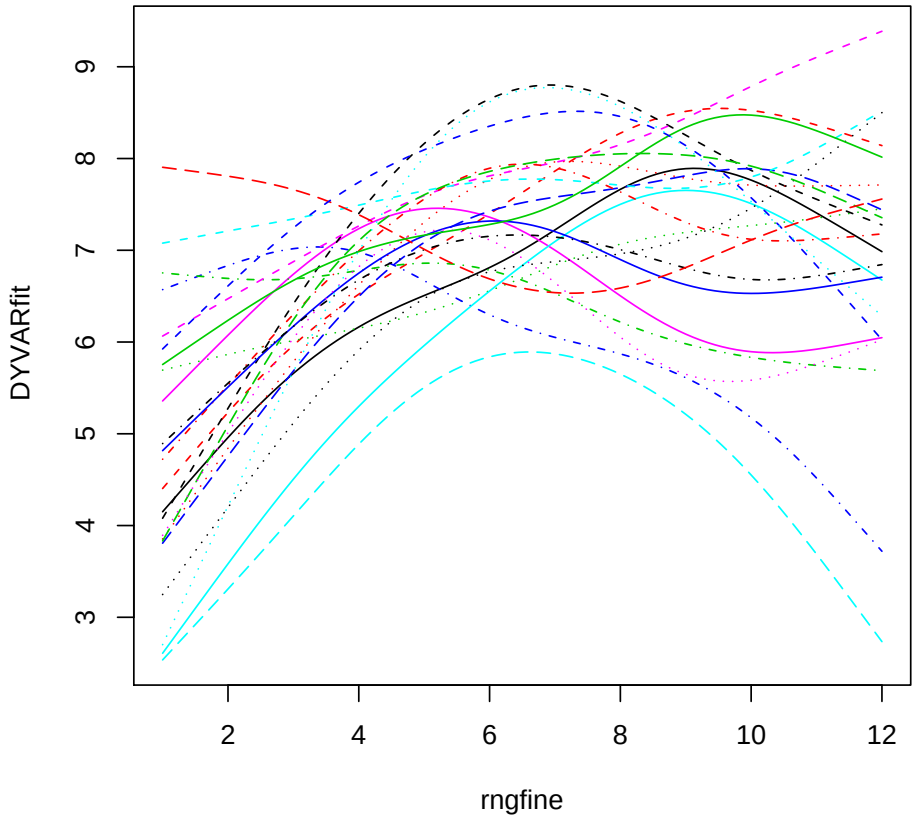
Smoothed values:

```
> YVARfit <- eval.fd(rngfine, fdobj)
> matplot(TVAR, YVAR, type = "p")
> matlines(rngfine, YVARfit)
```



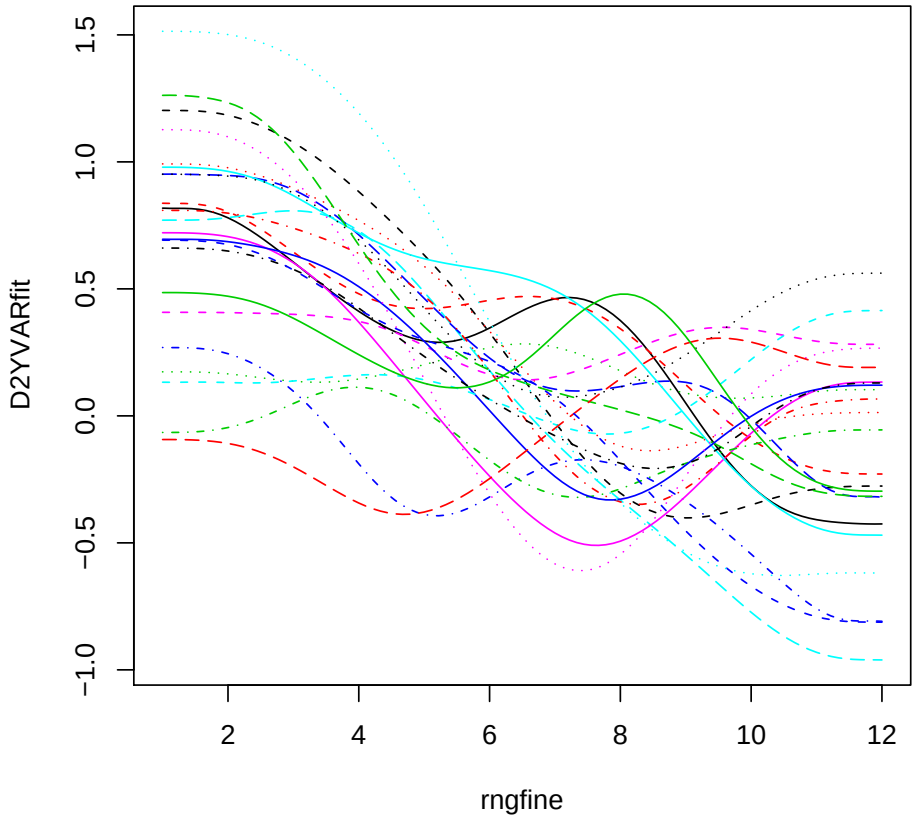
First derivative:

```
> DYVARfit <- eval.fd(rngfine, fdobj, 1)
> matplot(rngfine, DYVARfit, type = "l")
```



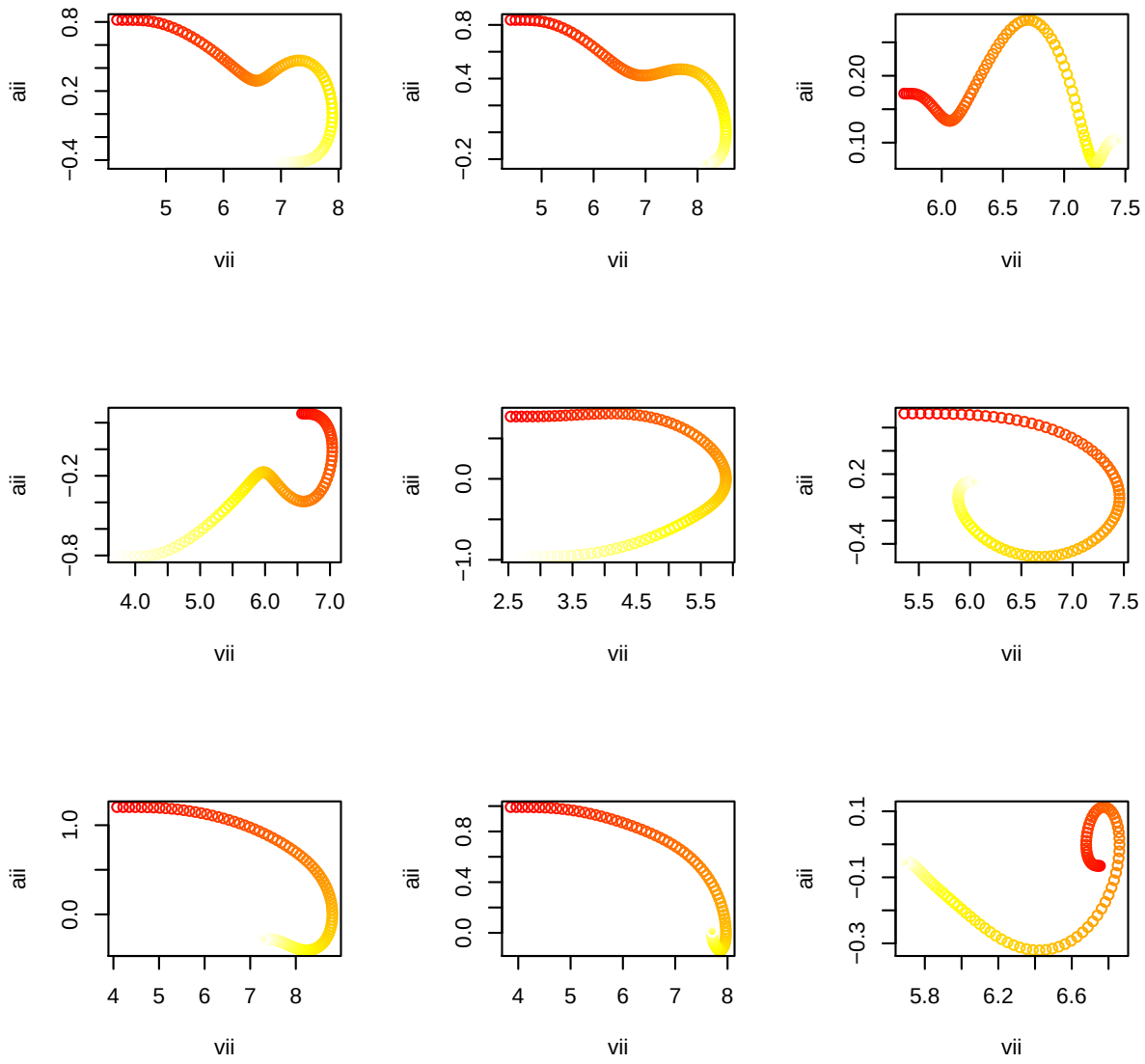
Second derivative:

```
> D2YVARfit <- eval.fd(rngfine, fdobj, 2)
> matplot(rngfine, D2YVARfit, type = "l")
```

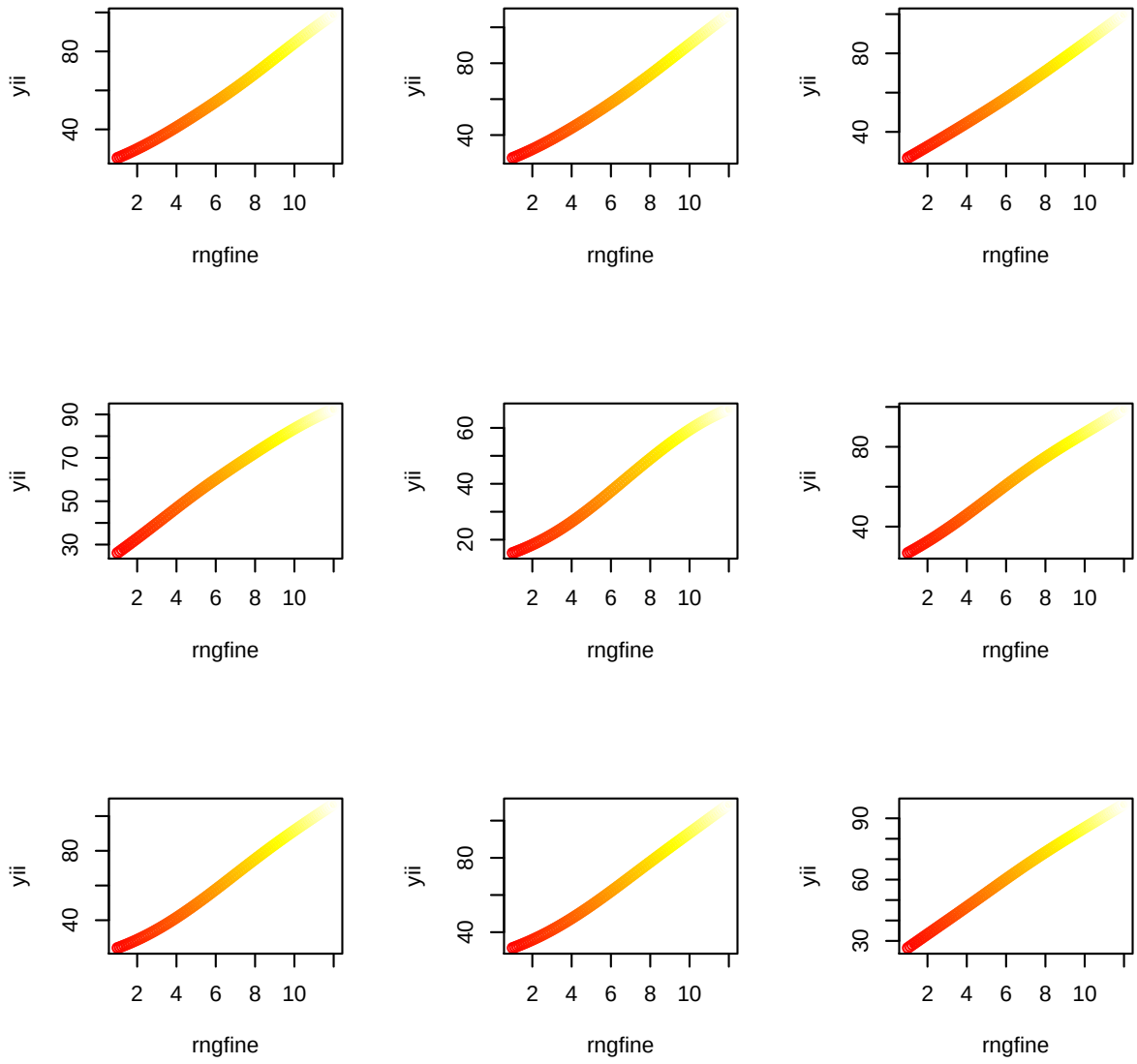


Plot acceleration against velocity for some pigs

```
> par(mfrow = c(3, 3))
> for (ii in 1:9) {
+     vii <- DYVARfit[, ii]
+     aii <- D2YVARfit[, ii]
+     cv <- heat.colors(length(aii))
+     plot(vii, aii, col = cv)
+ }
```




```
> par(mfrow = c(3, 3))
> for (ii in 1:9) {
+     yii <- YVARfit[, ii]
+     cv <- heat.colors(length(aii))
+     plot(rngfine, yii, col = cv)
+ }
```



2 Low hanging fruits

JIM; there are some low-hanging apples in terms of streamlining the functions:
The fdSmooth object should be equipped with the data as well

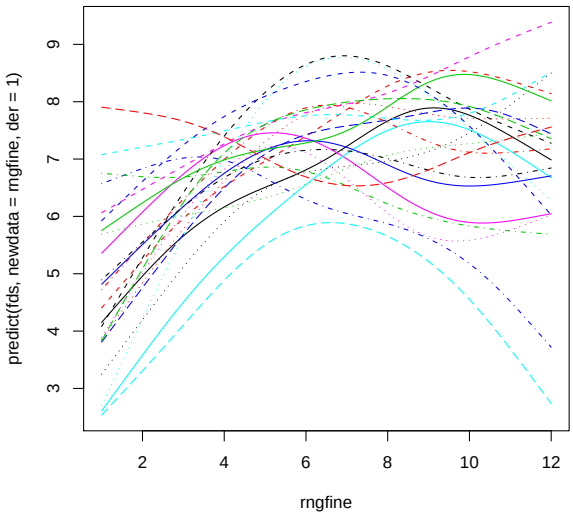
```
> fds <- smooth.basis(TVAR, YVAR, xxxfdPar)
> fds$TVAR <- TVAR
> fds$YVAR <- YVAR
```

Define

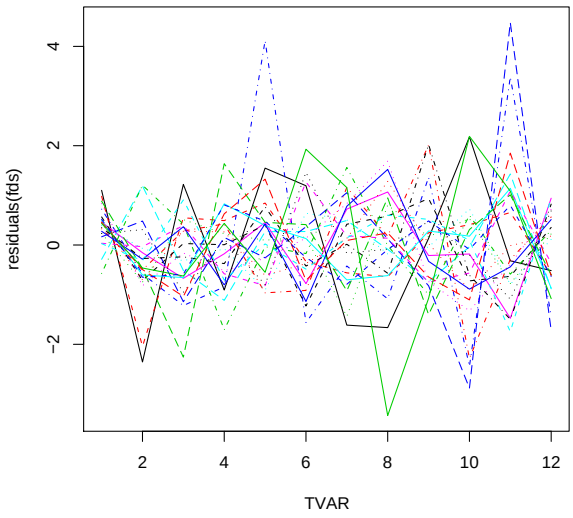
```
> predict.fdSmooth <- function(object, newdata, derivative = 0,
+     ...) {
+     if (missing(newdata))
+         newdata <- object$TVAR
+     eval.fd(newdata, object$fd, derivative)
+ }
> fitted.fdSmooth <- function(object, ...) {
+     eval.fd(object$TVAR, object$fd)
+ }
> residuals.fdSmooth <- function(object, ...) {
+     object$YVAR - eval.fd(object$TVAR, object$fd)
+ }
```

Now we can do:

```
> matplot(rngfine, predict(fds, newdata = rngfine, der = 1),
+     type = "l")
```




```
> matplot(TVAR, residuals(fds), type = "l")
```



The following model-building-steps could conveniently be wrapped into a single function

```
> norder <- 5
> nbasis <- length(TVAR) + norder
> rng <- range(TVAR)
> knots <- TVAR
> wbasis <- create.bspline.basis(rng, nbasis, norder, knots)
> cvec0 <- matrix(0, nbasis, 1)
> Wfd0 <- fd(cvec0, wbasis)
```

For example:

```
> fdmodel <- function(YVAR, TVAR, norder = 5, nbasis = length(TVAR) +
+     norder, knots = TVAR, start = rep(0, nbasis)) {
+     rng <- range(TVAR)
+     wbasis <- create.bspline.basis(rng, nbasis, norder,
+         knots)
+     cvec0 <- as.matrix(start, nrow = nbasis)
+     Wfd0 <- fd(cvec0, wbasis)
+     Wfd0$YVAR <- YVAR
+     Wfd0$TVAR <- TVAR
+     return(Wfd0)
+ }
```

So we can do:

```
> fdm <- fdmodel(YVAR, TVAR, norder = 8)
> fdm

Functional data object:

 Dimensions of the data:
```

```
    args
     reps
     funs

Basis object:

  Type:   bspline

  Range:  1  to  12

  Number of basis functions:  20

  Order of spline:  10
[1] "  Interior knots"
 [1]  2  3  4  5  6  7  8  9 10 11
```

The same applies to the model-fitting-steps:

```
> Lfdobj <- 3
> lambda <- 5
> xxxfdPar <- fdPar(Wfd0, Lfdobj, lambda)
> fdobj <- smooth.basis(TVAR, YVAR, xxxfdPar)$fd
```

For example:

```
> calibratefd <- function(object, derpenalty = 3, lambda = 1) {
+     xxxfdPar <- fdPar(object, derpenalty, lambda)
+     ans <- smooth.basis(object$TVAR, object$YVAR, xxxfdPar)
+     ans$YVAR <- object$YVAR
+     ans$TVAR <- object$TVAR
+     return(ans)
+ }
```

The above is not entirely general; the specific smoothing method in mind should go in somehere...
So now we can do

```
> fdmc <- calibratefd(fdm)
```

Thereb model specification and model fitting has been separated into two different steps...

Of course, TVAR and YVAR should not be stored twice...