Notes on Penalized Estimation and GAMs

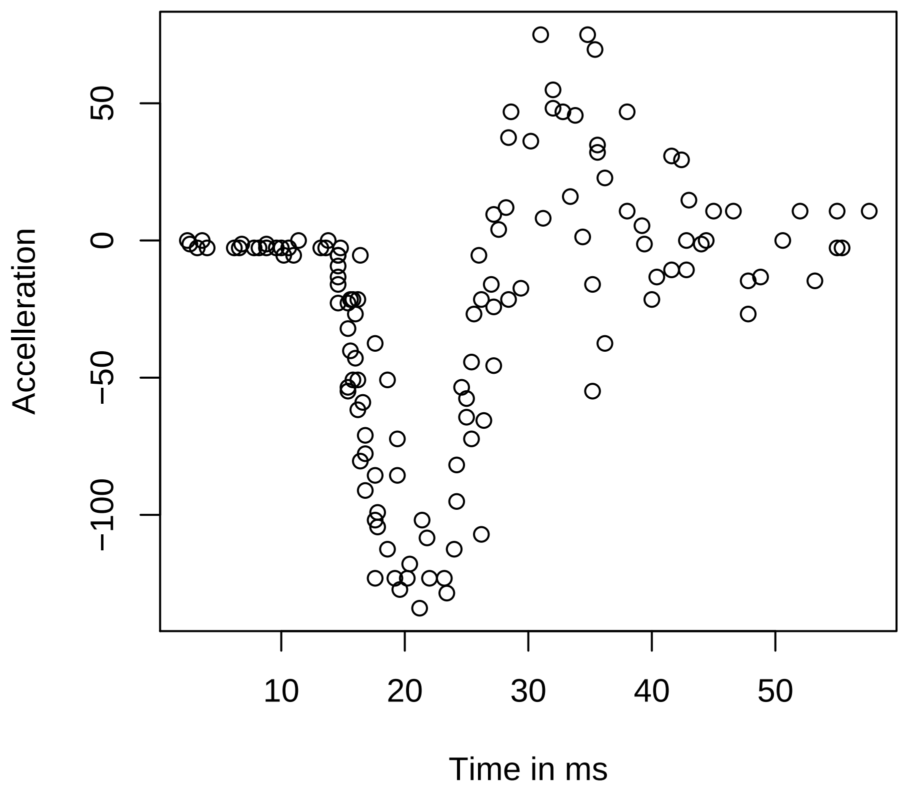
Introduction

Generalized additive models (GAMs) extend generalized linear models by allowing terms in the linear predictor that are *smooth* functions of the predictors. Estimation is then not my maximum likelihood, but by *penalized* maximum likelihood: the quantity maximised is the log-likelihood with an added *penalty* term which increases as complexity of the model increases. In these notes we look a little more closely at this idea in action.

0.1 The motor-cycle crash data

The `MASS` library contains a data set, `mcycle`, from a simulated motor-cycle crash. The data shows the accelleration, `accel` at the base of the skull of the crash dummy, at a sequence of times, `times`, in milliseconds, slightly before and after the crash incident.

```
> library(MASS)
> with(mcycle, plot(times, accel,
      ylab = "Accelleration", xlab = "Time in ms"))
```



We use this example to look at some smoothing ideas.

Consider finding a smooth function which describes the mean as a function of time. The first point to make is that polynomials are not very effective for this, since the data set has an abrupt change at the time of impact. To see this consider trying to capture the mean with some reasonably high order polynomial regressions.

Figure 1 shows how this method can lead to unsatisfactory results. The main problem is keeping the function still in the initial and final time segments.

```
> p_05 <- lm(accel ~ poly(times, 05), mcycle)
> p_10 <- lm(accel ~ poly(times, 10), mcycle)
> p_15 <- lm(accel ~ poly(times, 15), mcycle)
> p_20 <- lm(accel ~ poly(times, 20), mcycle)
> with(mcycle, plot(times, accel, ylab = "Accelleration",
                    xlab = "Time in ms"))
> dat <- with(mcycle,  data.frame(times =
        seq(min(times), max(times), len = 1000)))
> with(dat, {
      lines(times, predict(p_05, dat), col = "red")
      lines(times, predict(p_10, dat), col = "blue")
      lines(times, predict(p_15, dat), col = "green4")
      lines(times, predict(p_20, dat), col = "gold")
    })
> legend("bottomleft", c("p_05","p_10","p_15","p_20"), lty = 1,
           col = c("red","blue","green4","gold"), lwd = 2,
           bty = "n", cex = 0.75)
```

B−spline bases

The orthogonal polynomials used above are called the *basis* functions used for the *smoother*. The entire set is called the *base*. One way round the problem of polynomials is to use base functions that are zero, except for a small part of the range. One such base is the set of cubic B−splines. In this discussion we only consider the special case of *equally spaced knots*, but this is enough to give the idea.

To define a function in a cubic spline base, first define the truncated polynomials

$$P_j(x) = (x-j)_+^3 = \begin{cases} 0 & \text{if } x < j \\ (x-j)^3 & \text{if } x \geq j \end{cases}, \quad j = 0, 1, 2, \ldots$$

The B−spline basis functions on the integers are then defined by the differene:

$$B_j(x) = P_j(x) - 4P_{j+1}(x) + 6P_{j+2}(x) - 4P_{j+3}(x) + P_{j+4}(x)$$

which is a continuous piecewise cubic polynomial inside $j < x < (j+4)$ and zero outside this range.

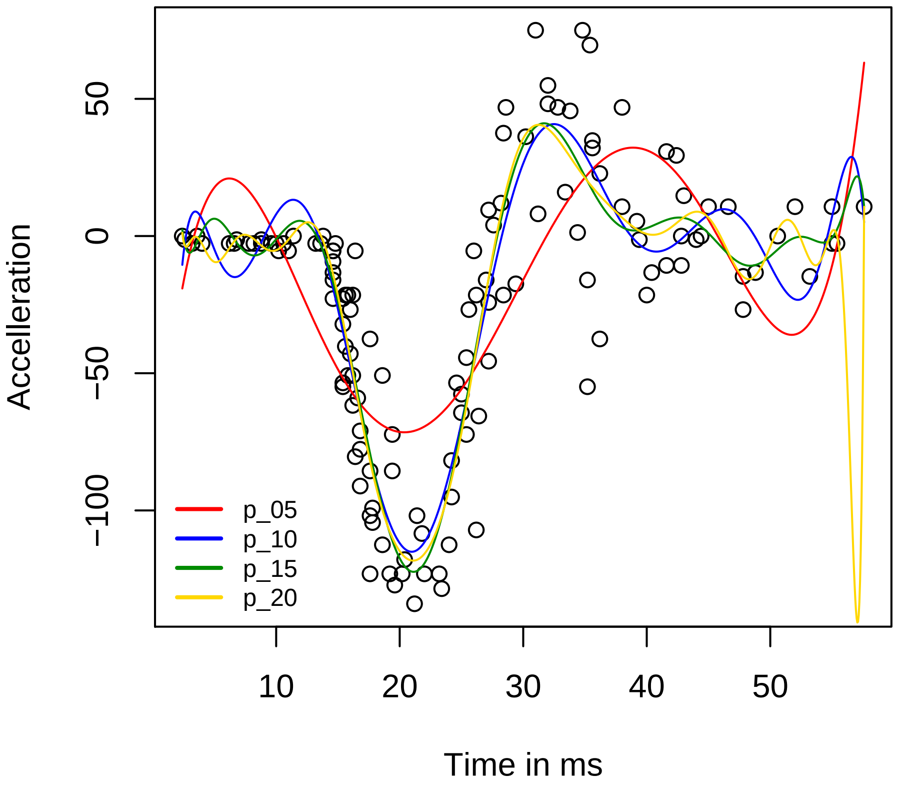
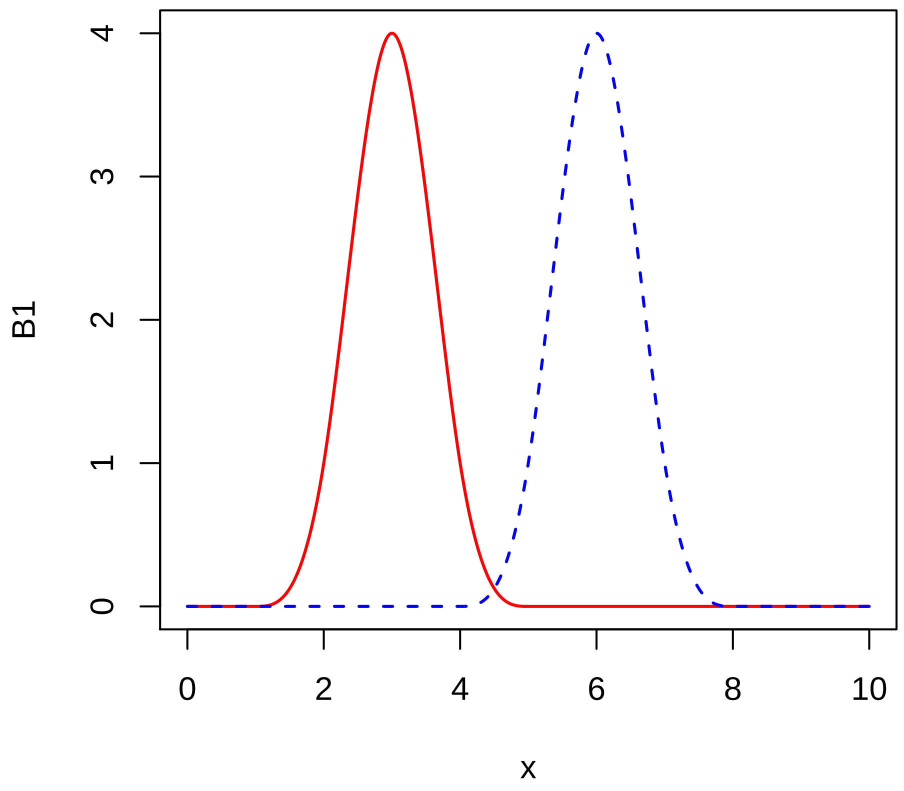


Figure 1: Smoothing with polynomials of high degree

```
> x <- seq(0, 10, len = 1001)
> P <- function(j, x) pmax(0, (x - j)^3)
> B1 <- P(1,x) - 4*P(2,x) + 6*P(3,x) - 4*P(4,x) + P(5,x)
> B4 <- P(4,x) - 4*P(5,x) + 6*P(6,x) - 4*P(7,x) + P(8,x)
> plot(x, B1, type = "l", col = "red", lwd = 1.5)
> lines(x, B4, col = "blue", lwd = 1.5, lty = "dashed")
```

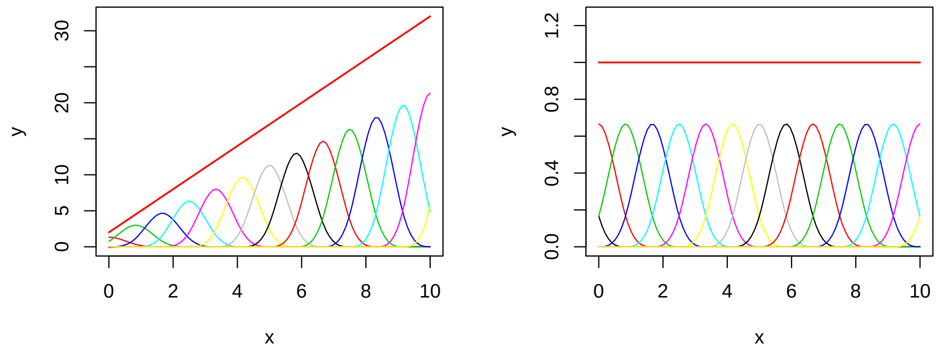


A simple function to calculate a B−spline basis on a finite x−range is as follows:

```
> Bspline <- function(x, k = 10) { ### local B spline basis, equally spaced
      rx <- range(x)
      k <- max(4, k)
      z <- 2*(x - rx[1])/(rx[2] - rx[1]) - 1
      k <- k-1
      del <- 2/(k-2)
      zs <- seq(-1 - 3*del, 1 + 4*del, by = del)
      B <- matrix(0, length(x), k+5)
      for(i in 1:(k+5))
        B[, i] <- pmax(0, (z-zs[i])^3)
      k <- k+1
      for(j in 1:k)
        B[, j] <- B[,j] - 4*B[,j+1] + 6*B[,j+2] - 4*B[,j+3] + B[,j+4]
      B[, 1:k]
    }
```

Note that the constant function and straight lines can both be written as linear combinations of $\boldsymbol{B}$−splines:

```
> par(mfrow=c(1,2))
> x <- seq(0, 10, len = 100)
> y <- 2 + 3*x
> B <- Bspline(x, 15)
> b <- lsfit(B, y, int = FALSE)$coef
> plot(x, y, type = "n", ylim = c(0, 32))
> lines(x, B %*% b, col = "red", lwd = 1.5)
> for(j in 1:15)
      lines(x, B[, j]*b[j], col = j)
> y <- rep(1, 100)
> b <- lsfit(B, y, int = FALSE)$coef
> plot(x, y, type = "n", ylim = c(0,1.25))
> lines(x, B %*% b, col = "red", lwd = 1.5)
> for(j in 1:15)
      lines(x, B[, j]*b[j], col = j)
```

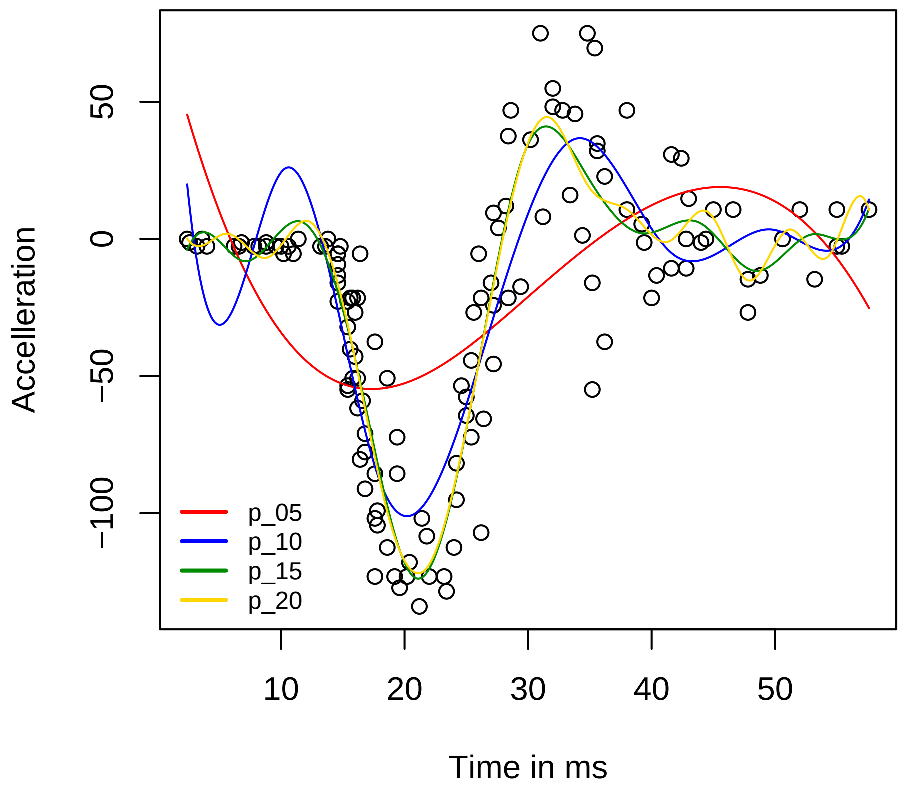


Hence, when we fit models with $\boldsymbol{B}$−splines defined in this simple way, both the intercept term and a linear term are implicitly included. In the examples that follows, we remove the intercept term for this reason.

Fitting regressions with $\boldsymbol{B}$−splines and penalties

We now return to the motor-cycle data example. Consider fitting regressions not with polynomial terms but with $\boldsymbol{B}$−splines.

```
> p_05 <- lm(accel ~ Bspline(times, 05)-1, mcycle)
> p_10 <- lm(accel ~ Bspline(times, 10)-1, mcycle)
> p_15 <- lm(accel ~ Bspline(times, 15)-1, mcycle)
> p_20 <- lm(accel ~ Bspline(times, 20)-1, mcycle)
> with(mcycle, plot(times, accel, ylab = "Accelleration",
                    xlab = "Time in ms"))
> dat <- with(mcycle,  data.frame(times =
        seq(min(times), max(times), len = 1000)))
> with(dat, {
      lines(times, predict(p_05, dat), col = "red")
      lines(times, predict(p_10, dat), col = "blue")
      lines(times, predict(p_15, dat), col = "green4")
      lines(times, predict(p_20, dat), col = "gold")
    })
> legend("bottomleft", c("p_05","p_10","p_15","p_20"), lty = 1,
           col = c("red","blue","green4","gold"), lwd = 2,
           bty = "n", cex = 0.75)
```



We can already start to see some improvement in that the fit with 20 basis functions remains much more controlled than with the polynomial of degree 20. Nevertheless we can do better still with penalization.

First look at the coefficients of the regression with 20 basis functions above:

```
> b <- structure(coef(p_20), names=paste("B",1:20,sep=""))
```

```
        B1         B2         B3         B4         B5         B6         B7
  5722.786  -1697.378   1183.833  -1859.267   2084.813  -2452.655 -11304.875
        B8         B9        B10        B11        B12        B13        B14
-13743.256  -7108.284   1580.309   6175.780   1004.567   1652.424  -1230.497
       B15        B16        B17        B18        B19        B20
  2672.527  -3524.809   2113.977  -2848.223   5527.631 -12780.414
```

```
> c(Roughness1 = sum(diff(b)^2), Roughness2 = sum(diff(b, diff = 2)^2))
```

```
Roughness1 Roughness2
 884577155 1857215564
```

Notice that there are some rapid changes between adjacent basis functions. This is basically why the curve starts to oscillate rapidly in some regions. The sum of squared differences, and sum of squared second differences, of he coefficients are two measures of 'roughness' of the fitted curve.

The idea behind penalization is as follows:

- If we use a large number of basis functions, the fitted function will be very flexible, but will reproduce a lot of the random variation, or 'noise', in the data and the curve will not be very smooth.
- Most of the excess variability in the curve will be caused by rapid changes between the coefficients of adjacent basis functions.
- We can measure the fit of the curve by the residual sum of squares. We can also measure the 'roughness' of the curve by the sum of squared second differences of adjacent coefficients.
- Why not use a large number of basis functions, but rather than fit by simple least squares, fit by minimising a quantity that quantity that increases *either* if the fit becomes poor *or* the curve becomes excessively rough?

To make this proposal concrete, note that:

$$\text{RSS} = \|\mathbf{y} - \boldsymbol{B}\boldsymbol{\beta}\|^2 \qquad \text{R} = \sum_j \left(\beta_j - 2\beta_{j+1} + \beta_{j+2}\right)^2 = \boldsymbol{\beta}^\mathsf{T}\boldsymbol{D}^\mathsf{T}\boldsymbol{D}\boldsymbol{\beta}$$

where

$$\boldsymbol{D} = \begin{bmatrix} 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

which can be easily calculated in R as `D = diff(diag(k), diff=2)`. The composite measure we minimise is then

$$\text{RSS} + \lambda\text{R} = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}\|^2 + \lambda\boldsymbol{\beta}^\mathsf{T}\mathbf{D}^\mathsf{T}\mathbf{D}\boldsymbol{\beta}$$

where λ is a tuning constant. Large values of λ penalise roughness excessively, and small values allow too much flexibility into the smoother.

It can easily be shown that the penalized model cam be fitted by regression a modified response vector on a modified model matrix, as follows

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{z} \qquad \text{where} \quad \mathbf{z} = \left[\begin{array}{c}\mathbf{y}\\ \mathbf{0}\end{array}\right] \quad \mathbf{X} = \left[\begin{array}{c}\mathbf{B}\\ \sqrt{\lambda}\mathbf{D}\end{array}\right]$$

This idea is used in the penalised fitting function below. The matrix

$$\mathbf{S}(\lambda) = \mathbf{B}\left(\mathbf{B}^{\mathsf{T}}\mathbf{B} + \lambda\mathbf{D}^{\mathsf{T}}\mathbf{D}\right)^{-1}\mathbf{B}$$

is called the *smoothing matrix*. It maps the observation vector into the smoothed curve. The trace of the smoothing matrix and be shown to be an approximate equivalent degrees of freedom for the fitted smooth function, that is, a measure of complexity of the fitted curve.

$$\tilde{v}(\lambda) = \operatorname{tr}\mathbf{S}(\lambda)$$

The equivalent degrees of freedom need not be an integer. For such models the usual measures of fit are then

$$\text{AIC} = b\log(\text{RSS}(\lambda)/n) + 2\tilde{v}(\lambda) \qquad \text{BIC} = b\log(\text{RSS}(\lambda)/n) + \log n\tilde{v}(\lambda)$$

Fitting with a prescribed λ

Consider now fitting penalized B−spline models to the motor-cycle data. A simple function to fit such a model with a prescribed λ is as follows:

```
> ### fit a penalized regression
>
> fitPenalized <- function(x, y,
          k = length(unique(x)), lambda, d = 2) {
      B <- Bspline(x, k)
      D <- diff(diag(k), diff = d)
      X <- rbind(B, sqrt(lambda) * D)
      Y <- c(y, rep(0, nrow(D)))
      fit <- lsfit(X, Y, int = FALSE)
      n <- length(y)
      rs <- fit$resid[1:n]
      dfr <- sum(rowSums((X %*% solve(crossprod(X) +
              lambda * crossprod(D))) * X))
      AIC <- n*log(sum(rs^2)/n) + 2*dfr
      BIC <- AIC + (log(n) - 2)*dfr
      structure(list(coefficients = fit$coef, dfr = dfr,
        residuals = rs, fitted = y - rs,
        AIC = AIC, BIC = BIC), class = "fitPenalized")
    }
```

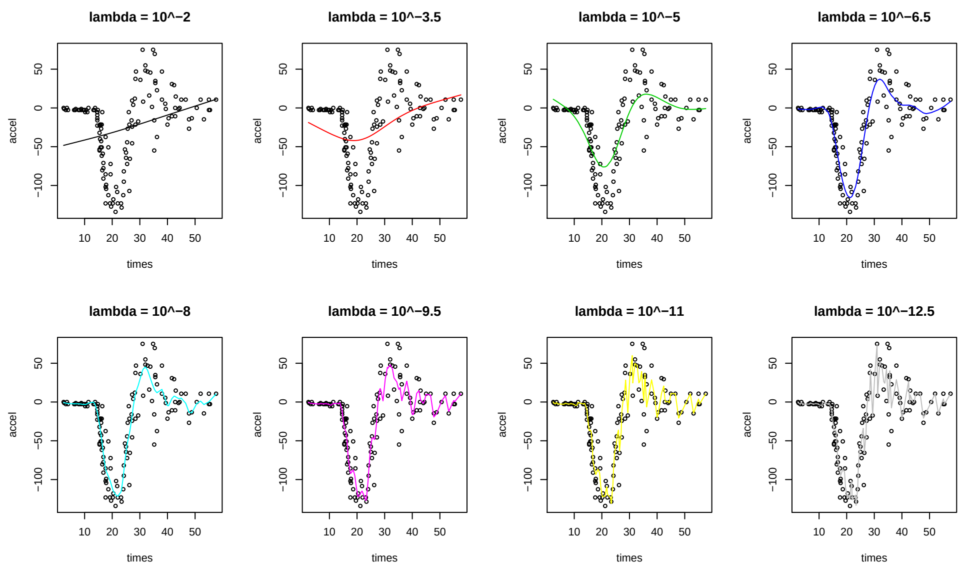
Note that by default this uses as many B−splines as there are distinct values of x, so penalization will certainly be needed in this default case.

It is usual to vary λ uniformly on a log-scale.

```
> with(mcycle, {
      par(mfrow = c(2,4))
      rf <- sqrt(1000)  ## reduction factor
      lambda <- 1/100

      for(j in 1:8) {
        plot(times, accel, cex = 0.7)
        ft <- fitPenalized(times, accel, lambda = lambda)
        with(ft, cat(j, "  df:", round(dfr, 3),
                     " log10(lambda):", log10(lambda),
                     "  AIC:", AIC,
                     "  BIC:", BIC, "\n"))
        lines(times, fitted(ft), col = j)
        title(main = paste("lambda = 10^", log10(lambda), sep=""), cex = 0.7)
        lambda <- lambda/rf
      }
    })
```

```
1   df: 48.018  log10(lambda): -2   AIC: 1112.556   BIC: 1251.345
2   df: 48.344  log10(lambda): -3.5   AIC: 1085.923   BIC: 1225.654
3   df: 49.554  log10(lambda): -5   AIC: 997.6657   BIC: 1140.895
4   df: 52.281  log10(lambda): -6.5   AIC: 921.596   BIC: 1072.705
5   df: 58.212  log10(lambda): -8   AIC: 919.1351   BIC: 1087.388
6   df: 68.355  log10(lambda): -9.5   AIC: 921.3492   BIC: 1118.919
7   df: 78.873  log10(lambda): -11   AIC: 923.2274   BIC: 1151.199
8   df: 84.281  log10(lambda): -12.5   AIC: 928.606   BIC: 1172.207
```



We can see that decreasing λ allows greater flexibility to the point where the curve can become very rough, although it remains close to the data, unlike the case of polynomials.

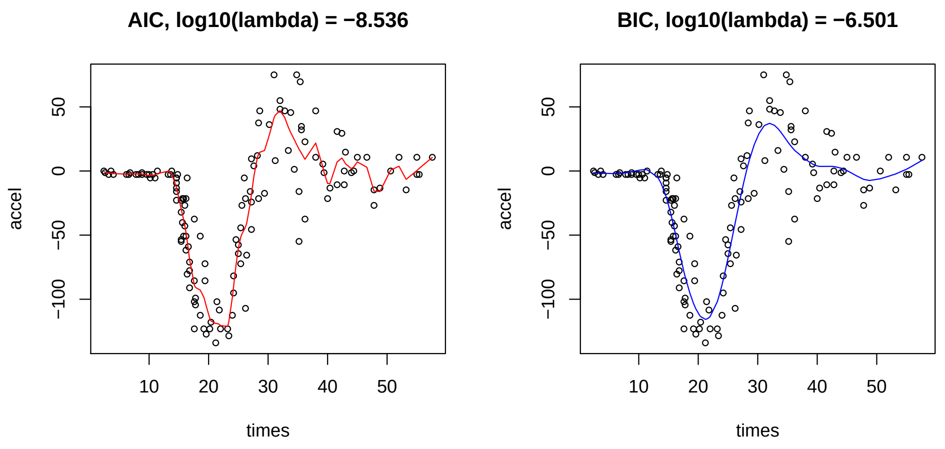
Choosing λ

The appropriate value of λ is usually assessed by *cross-validation*, that is by looking at the predictive performance of the model from internal evidence of the fitting data. We do not discuss this here, though. See the description, for example, in the MASS book.

A simplere way to choose an optimal λ is to minimise either AIC or BIC. We can do this easily here as follows.

```
> ## find best lambda by two crieria
>
> fAIC <- function(loglambda)
      with(mcycle, fitPenalized(times, accel,
                                    lambda = 10^loglambda)$AIC)
> fBIC <- function(loglambda)
      with(mcycle, fitPenalized(times, accel,
                                    lambda = 10^loglambda)$BIC)
> lAIC <- optimize(fAIC, -c(4,9))
> lBIC <- optimize(fBIC, -c(4,9))
> par(mfrow=c(1,2))
> with(mcycle, {
      plot(times, accel, cex = 0.7)
      ft <- fitPenalized(times, accel, lambda = 10^lAIC$minimum)
      lines(times, fitted(ft), col = "red")
      title(main = paste("AIC, log10(lambda) =",
                         round(lAIC$min, 3)), cex = 0.5)

      plot(times, accel, cex = 0.7)
      ft <- fitPenalized(times, accel, lambda = 10^lBIC$minimum)
      lines(times, fitted(ft), col = "blue")
      title(main = paste("BIC, log10(lambda) =",
                         round(lBIC$min, 3)), cex = 0.5)
    })
```

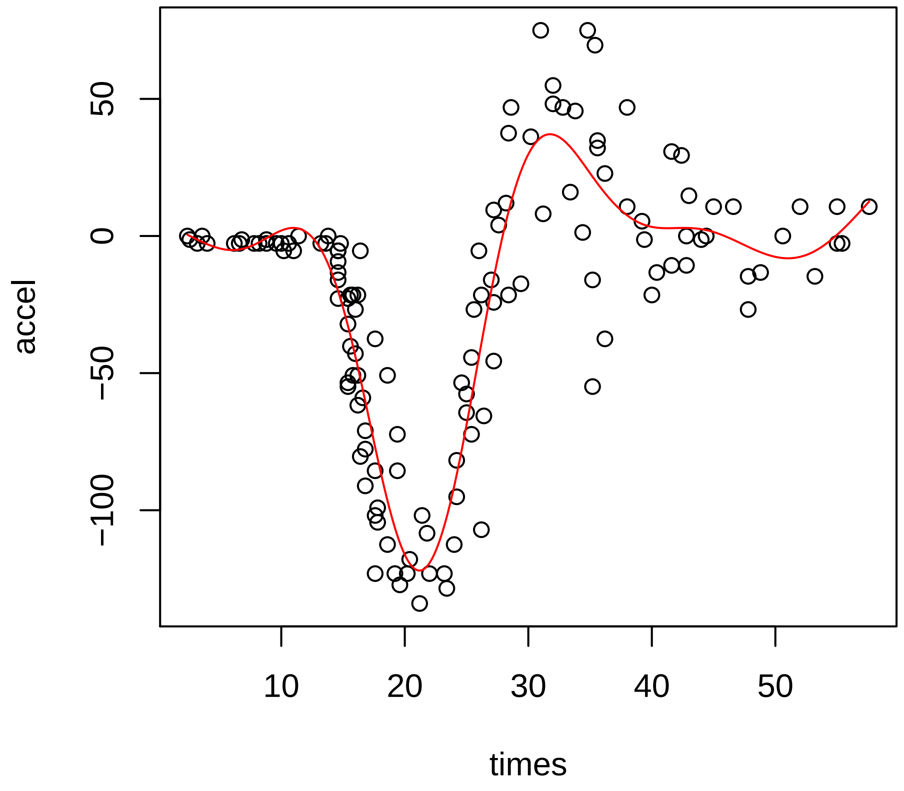


Finally, we compare this with the standard software result, that is, using generalized cross-validation to choose λ.

```
> library(mgcv)

This is mgcv 1.3-26

> p_gam <- gam(accel ~ s(times), data = mcycle)
> with(mcycle, plot(times, accel))
> dat <- with(mcycle, data.frame(times =
           seq(min(times), max(times), len = 1000)))
> with(dat,
      lines(times, predict(p_gam, dat), col = "red"))
```



Extensions to the real world

In practice, penalised estimation with GAMs is somewhat more complex than this simple demonstration might suggest. We make some notes here, which may be followed up with examples.

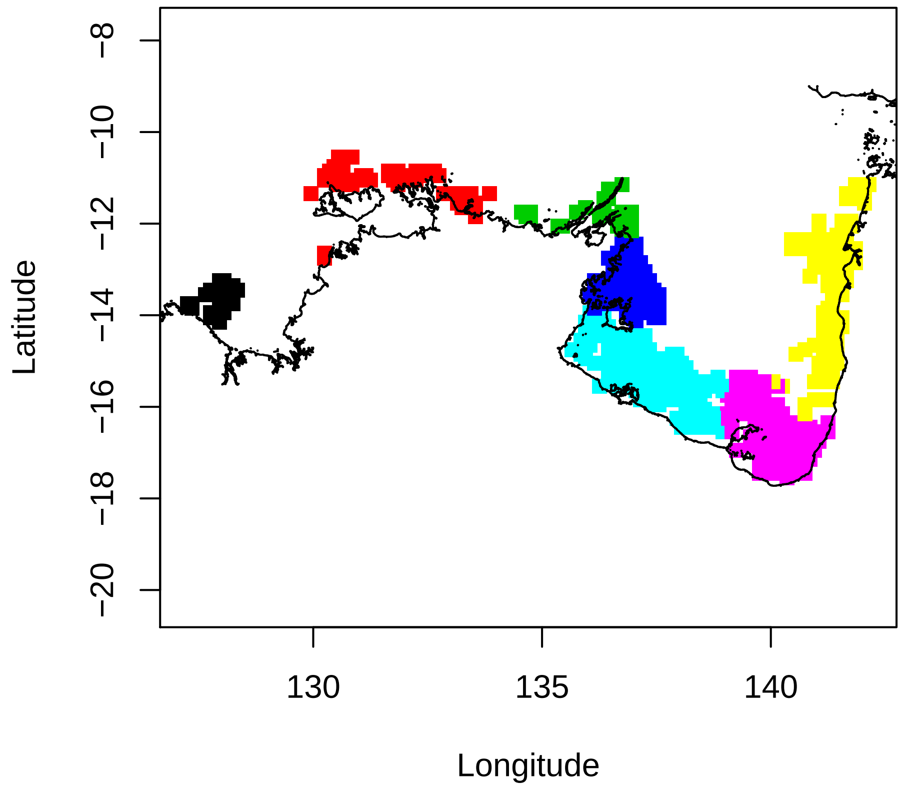
- In practice there are usually several smooth terms, each of which may involve one or more predictors. In these cases several tuning constants must be chosen and this is usually done by generalized cross-validation.

- $B-$spline bases are only one of a number of local bases that can be chosen. It is not usually the case that basis functions need be equally spaced, but rather linked to the actual $x-$values present in the data.
- In two or more dimensions, a wide variety of basis functions is available. For example, bivariate basis functions may be *isotropic*, or they may be *tensor splines*, or *thin-plate splines*, an so on. They may also be *periodic* in either or both directins.

A real example

Tiger prawns caught in Northern Australia are a mixture of two biological species, *Penaeus semisulcatus* and *P. esculentus*. This example uses data from an historical series of research surveys to predict the proportions of the two species as a function of spatial location, time of year, and possibly a long-term time trend.

```
> library(ASOR)
> load("TigersW.RData")
> Store(TigersW, Aus)
> with(TigersW, plot(Longitude, Latitude, asp = 1,
                     pch = 15, col = as.numeric(StockTig)))
> Aus(add = T)  ## coastline
```



```
> names(TigersW)

 [1] "Survey"    "Station"   "Survey_ID" "Pesc"      "Psem"      "GTag"
 [7] "Date"      "Longitude" "Latitude"  "Tiger"     "Fsemi"     "Year"
[13] "PDay"      "Time"      "cosP"      "sinP"      "Rdist"     "Rland"
[19] "Depth_av"  "Mud_av"    "Sand_av"   "MGS_av"    "Gravel_av" "Stress_av"
[25] "StockTig"
```

The fitted model will be a `quasibinomial` model with the proportion, by weight, of *P. semisulcatus* as the response. The total weight of the trawl will be the weight, namely the variable `Tiger`.

```
> library(mgcv)
> t_gam <- gam(Fsemi ~ s(Longitude,Latitude) +
             te(PDay, Depth_av) + te(PDay, Rland) +
             s(Mud_av), quasibinomial, TigersW,
             weight = Tiger, trace = T)
```

A further model was used to check for evidence of a long-term trend.

```
> t_gam_t <- update(t_gam, . ~ . + s(Time))
```

These models can be checked graphically using `plot` and `vis.gam`.

Acknowledgement

In preparing these notes I have made use of some ideas presented in more detail in the booklet: *Splines, Knots, and Penalties. The Craft of Smoothing* by Brian Marx and Paul Eilers. 9^a Escola de Modelos de Regressão, 21 a 23 de fevereiro de 2005, São Pedro, SP. Readers are referred to this booklet for a more complete account.