Lecture 26 (lab 9)—Friday, March 16, 2012

Topics

R functions and commands demonstrated

- gam (from **mgcv**) for fitting generalized additive models (GAMs).
- gamm (from **mgcv**) for fitting generalized additive mixed effects models (GAMMs).
- loess fits a local polynomial regression smoother to a scatter plot of data.
- lowess fits a local polynomial regression smoother to a scatter plot of data.
- methods lists the available methods for a given function.
- plot.gam (from **mgcv**) is a method for the **plot** function for displaying **gam** objects.
- s (from **mgcv**) is the smoothing spline function used to estimate a smooth in a GAM.
- smooth.spline fits a smoothing spline to a scatter plot.

R function options

- by= (argument to **s**) is used to fit separate smooths for each level of a specified factor variable.
- f= (argument to **lowess**) is used to define the span of the **lowess** smoother.
- family= (argument to **loess**) is used to add a robust step to local polynomial regression: **family="symmetric"**.
- k= (argument to **s**) is used to specify the dimension of the basis used to represent the smooth. The default is $k = 10$.
- method= (argument to **gam** and **gamm**) is used to change the default estimation method from GCV to something else.
- new= (argument to **par**) when set to TRUE it prevents a high-level graphics command (such as **plot**) from clearing the graphics window.
- newdata= (argument to **predict**) is used to define new data values at which to obtain predictions from a fitted regression model.
- pages= (argument to **plot.gam**) is used to display all the estimated partial response functions in a single graphics window.
- select= (argument to **plot.gam**) is used to select a specific estimated partial response function to display.
- span= (argument to **loess**) is used to define the span of the **loess** smoother.

R packages used

- mgcv used for the functions **gam** and **gamm** to fit GAMs and GAMMs.

Lowess, loess, and splines

The data set beepollen.txt is taken from Harder and Thompson (1989). As part of a study to investigate reproductive strategies in plants, they recorded the proportions of pollen removed by bumblebee queens and honeybee workers pollinating a species of *Erythronium* lily and the time they spent at the flower. There are three variables in the data set.

1. removed: proportion of pollen removed
2. duration: duration of visit in seconds
3. code: 1=bumblebee queens, 2=honeybee workers

The response variable of interest is removed and the predictor is duration. The variable removed is a proportion, so it normally would offer a special challenge for analysis, but for these data the recorded proportions are far removed from the boundary values of zero and one so it is safe to treat them as being normally distributed.

```
bees <- read.table("ecol 562/beepollen.txt", header=T)
names(bees)
[1] "removed"  "duration" "code"
bees[1:8,]
  removed duration code
1    0.07        2    1
2    0.10        5    1
3    0.11        7    1
4    0.12       11    1
5    0.15       12    1
6    0.19       11    1
7    0.28        9    1
8    0.31        9    1
```

To help determine the functional form of the relationship between removed and duration we plot the data and superimpose a nonparametric smoother. The **lowess** function has an argument **f** that defines the span of the smoother, the fraction of the data that should be included in the window of the smooth. Values of **f** closer to 1 yield a smoother curve. The default setting is `f=2/3`.

```
plot(removed~duration, data=bees)
out.low <- lowess(bees$removed~bees$duration)
names(out.low)
[1] "x" "y"
```

The **lowess** function returns two vectors: the sorted values of the *x*-variable and the lowess predictions of the *y*-values sorted by the values of the *x*-variable. As a result the **lines** function can be used to add the output of **lowess** to an existing scatter plot.

```
lines(out.low, col=2)
#increase the span to 0.95
lines(lowess(bees$removed~bees$duration, f=.95), col='grey70', lwd=2)
```

```
legend('bottomright', c('f = 2/3', 'f = .95'), col=c(2,'grey70'), lwd=c(1,2), lty=1, cex=.9,
   title='span', bty='n')
```

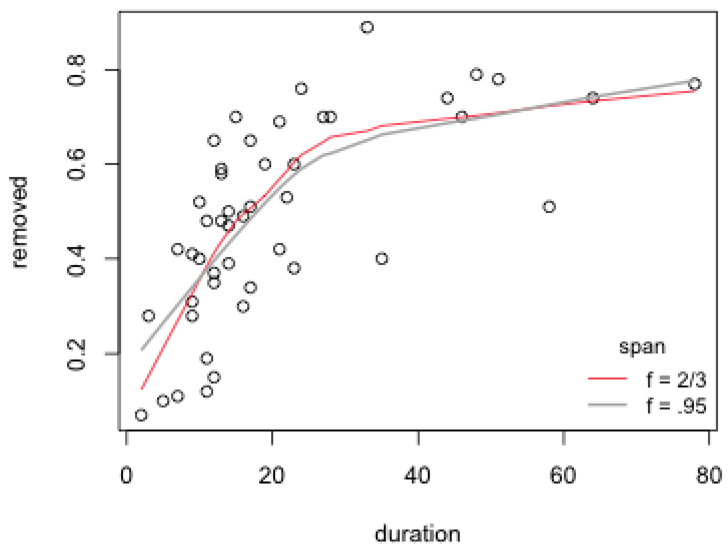


Fig. 1 Smoothers produced by the lowess function using different spans

The **loess** function of R is a second implementation of lowess. It has a **span** argument that controls the span and hence the band width. The **loess** function is more flexible than **lowess** in that it has a **data** argument, can handle missing values, and has a method for the **predict** function so that new values of the response variable can be predicted using **loess**. One drawback of **loess** is that it doesn't automatically sort the fitted values so this needs to be done before the observations can be plotted.

```
plot(removed~duration, data=bees)
out.loess <- loess(removed~duration, data=bees)
names(out.loess)
 [1] "n"          "fitted"     "residuals" "enp"        "s"          "one.delta"
 [7] "two.delta" "trace.hat" "divisor"    "pars"       "kd"         "call"
[13] "terms"      "xnames"     "x"          "y"          "weights"
```

The smoothed values are in the **"fitted"** component whereas **"x"** contains the unsorted values of the predictor.

```
lines(out.loess$x[order(out.loess$x)], out.loess$fitted[order(out.loess$x)], col=2)
```

The default settings for **loess** are different from those of **lowess**. The default span is 0.75 and the last robust step of lowess is not carried out. To include the final robust step you need to specify the argument **family="symmetric"**.

```
out.loess2 <- loess(removed~duration, data=bees, family="symmetric")
```

```
lines(out.loess2$x[order(out.loess2$x)], out.loess2$fitted[order(out.loess2$x)], col=4)
legend('bottomright',c('default loess', 'robust loess'), col=c(2,4), lty=1, bty='n', cex=.9)
```

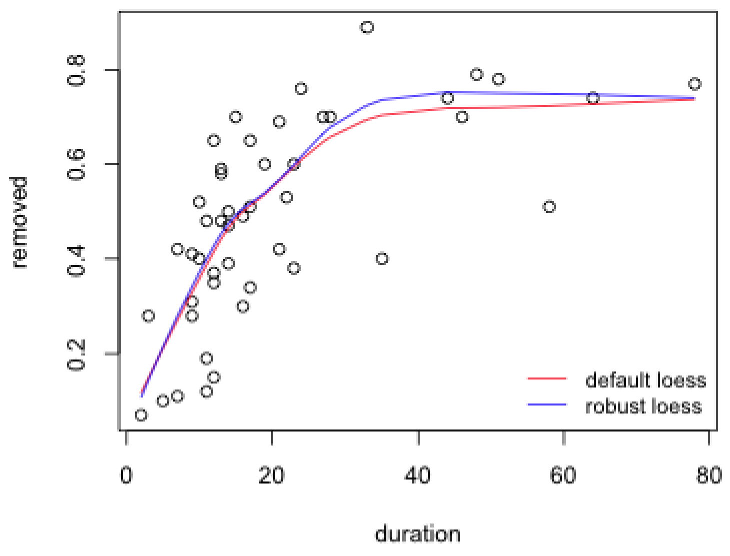


Fig. 2 Smoothers produced by the loess function

There are many ways to generate spline fits in R. Smoothing splines can be obtained with the **smooth.spline** function. Like **lowess** it returns the predictions sorted in order of the *x*-variable and so the spline fit can be plotted directly with the **lines** function.

```
plot(removed~duration, data=bees)
out.sp <- smooth.spline(bees$duration, bees$removed)
names(out.sp)
 [1] "x"        "y"        "w"        "yin"      "data"     "lev"      "cv.crit"
 [8] "pen.crit" "crit"     "df"       "spar"     "lambda"   "iparms"   "fit"
[15] "call"


lines(out.sp, lwd=2, col='grey70')
```

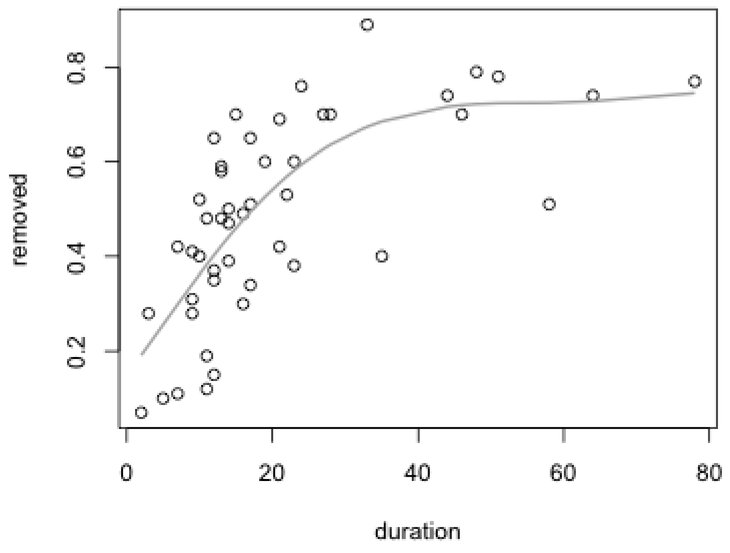


Fig. 3 Smoothing spline

An identical smoothing spline fit can also be obtained by fitting a generalized additive model using the **gam** function of the **mgcv** package. The **gam** function supports ordinary formula notation except that we need to use the **s** function on those predictors for which we wish to estimate smoothers. The primary reference for the **mgcv** package is Wood (2006).

```
library(mgcv)
model0 <- gam(removed~s(duration), data=bees)
```

To add the spline model to a plot we need to first use the **predict** function to generate predicted values and then sort the predictions in the order of the *x*-variable before plotting them.

```
model0.p <- predict(model0)
lines(bees$duration[order(bees$duration)], model0.p[order(bees$duration)], col=3)
```

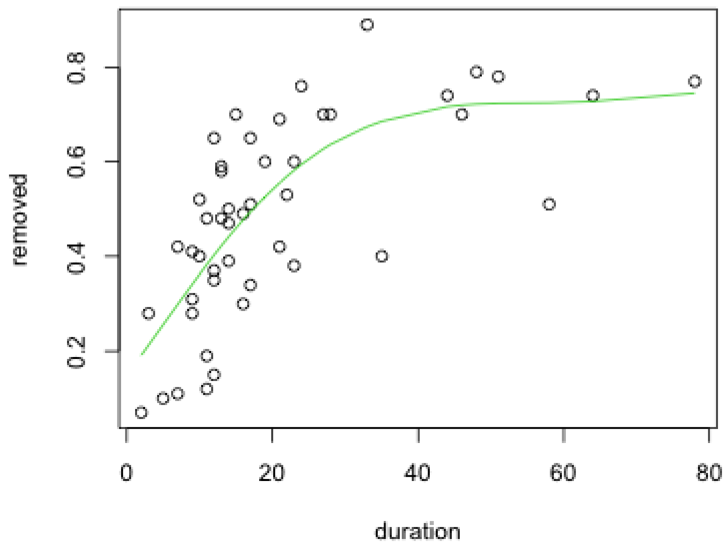


Fig. 4 Smoothing spline obtained from fitting a GAM

Based on the estimated smoother it appears that an increasing function with an asymptote would be a reasonable model. A simple choice is the two-parameter exponential model.

$$f(x) = b\left[1 - \exp(ax)\right]$$

Here b is the asymptote and a is a negative number that controls the rate of approach to the asymptote. The **nls** function can be used to fit nonlinear models to data. (We previously used **nls** to fit a nonlinear Arrhenius model in Assignment 4.) The **nls** function requires a formula for the nonlinear function with the variables specified as well as a **start** argument that provides initial values for the model parameters, here a and b. From the plot we estimate that the asymptote b should be around 0.7. For a I choose a small negative number.

```
myfunc <- function(x,a,b) b*(1-exp(a*x))
out.nls <- nls(removed~myfunc(duration,a,b), data=bees, start=list(a=-.1, b=.7))
summary(out.nls)
Formula: removed ~ myfunc(duration, a, b)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a -0.06858    0.01209  -5.674  9.5e-07 ***
b  0.74021    0.05826  12.705  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1388 on 45 degrees of freedom
```

```
Number of iterations to convergence: 4
Achieved convergence tolerance: 4.353e-06
```

I use the **curve** function to add the estimated exponential curve to Fig. 3. The fitted exponential curve provides a close approximation to the smoothing spline (Fig. 5).

```
plot(removed~duration, data=bees)
lines(bees$duration[order(bees$duration)], model0.p[order(bees$duration)], col='grey70', lwd=2)
curve(myfunc(x, coef(out.nls)[1], coef(out.nls)[2]), add=T, col=2, lty=2)
legend('bottomright', c('exponential', 'spline'), col=c(2,'grey70'), lwd=c(1,2), lty=c(2,1), cex=.9,
    bty='n')
```

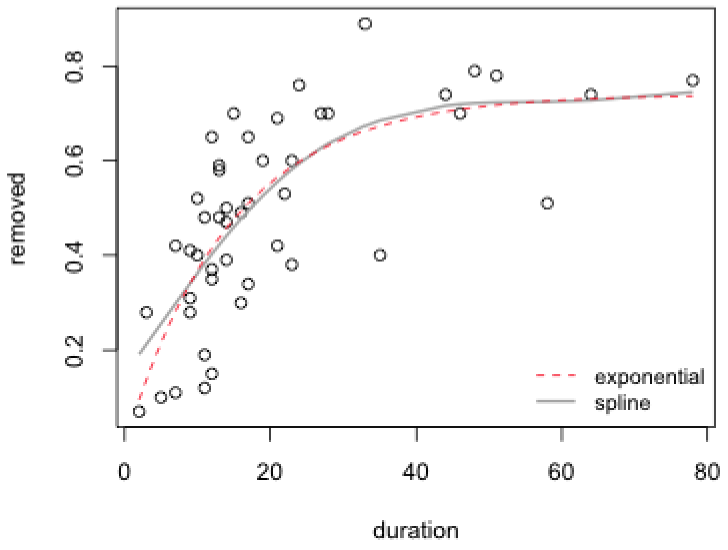


Fig. 5 Smoothing spline and nonlinear parametric model

GAMs with interactions

If we use different symbols and colors to denote the kinds of bees (bumblebee or honeybee) in the scatter plot we see that the two kinds of bees are not exhibiting the same pattern.

```
plot(removed~duration, data=bees)
points(bees$duration, bees$removed, col=bees$code, pch=c(1,16)[bees$code])
legend('bottomright', c('bumblebees', 'honeybees'), pch=c(1,16), col=1:2, cex=.9, bty='n')
```

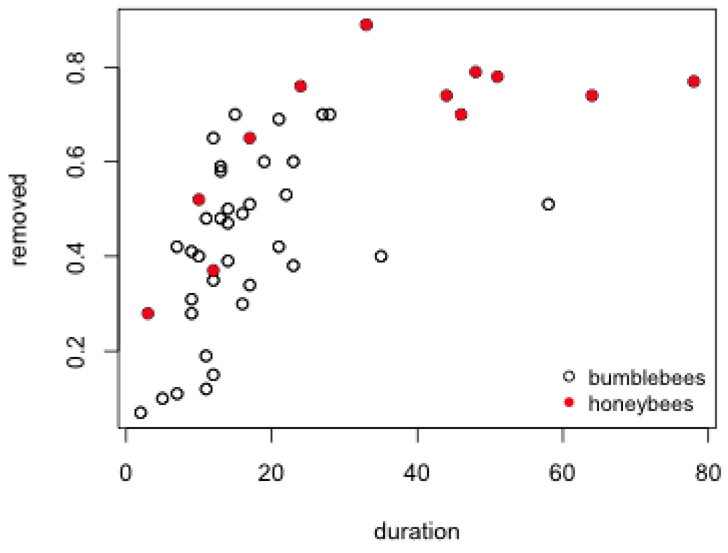


Fig. 6 Using the code variable to identify the bees

Honeybees appear to be largely responsible for driving the saturation model we've been fitting. Bumblebees may show no saturation at all (if we ignore two outlying values) or at best saturate at a much lower level. To examine these patterns we could fit separate lowess curves to subsets of the data in an obvious way. Alternatively we can interact the smoother with a categorical predictor and fit a GAM. I illustrate the latter approach.

To fit a model in which both bee models have the same basic functional form and only the asymptote is shifted we can include a single smooth of duration as before but include the code variable as a factor. The factor adds to the intercept and acts to shift the curve and asymptote up and down.

```
model1 <- gam(removed~s(duration) + factor(code), data=bees)
```

To plot the estimated curves I use the **predict** function and supply it with new data in the **newdata** argument. The value of **newdata** must be a data frame with columns that have the same names as the predictors that were used to fit the model. I first determine the range of the *x*-variable duration for the two bee types and then predict the mean response separately for the two kinds of bees.

```
tapply(bees$duration, bees$code, range)
$`1`
[1]  2 58

$`2`
[1]  3 78

#obtain the predictions
out.p1a <- predict(model1, newdata=data.frame(duration=1:58, code=rep(1,58)))
out.p1b <- predict(model1, newdata=data.frame(duration=1:78, code=rep(2,78)))
```

```
#plot the results
plot(removed~duration, data=bees)
points(bees$duration, bees$removed, col=bees$code, pch=c(1,16)[bees$code])
lines(1:58, out.p1a, lty=2)
lines(1:78, out.p1b, lty=2, col=2)
legend('bottomright', c('bumblebees','honeybees'), pch=c(1,16), col=1:2, cex=.9, bty='n')
```

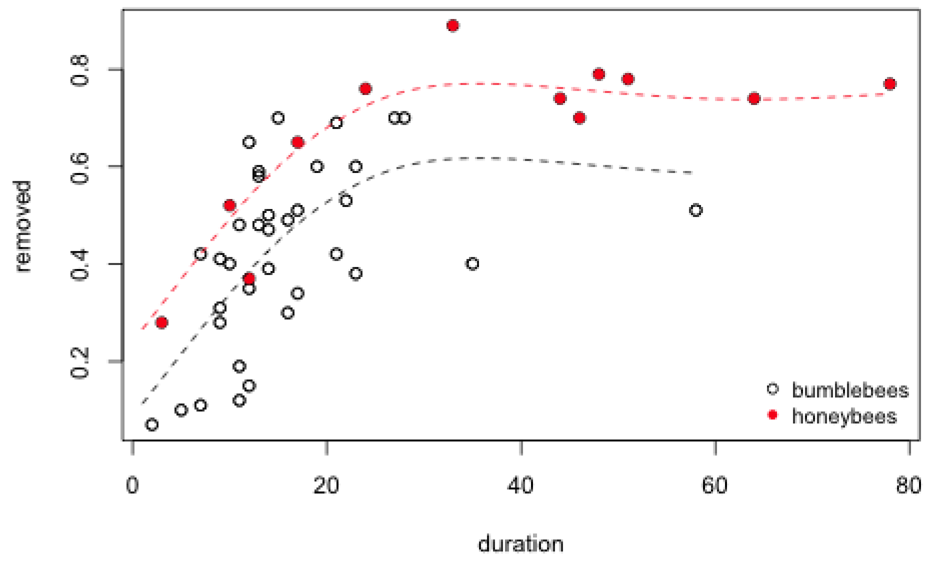


Fig. 7 Additive model with a factor variable and a smooth

We can also let the entire functional form of the smoother vary by bee type by including factor(code) as a predictor and in the **by** argument of the smoother.

```
model2 <- gam(removed~factor(code)+s(duration, by=factor(code)), data=bees)
```

A second way to fit this same model is to use the smoother twice each time selecting a different subset of the data using the **by** argument.

```
model2a <- gam(removed~s(duration, by=as.numeric(code==1))+s(duration, by=as.numeric(code==2)),
   data=bees)
AIC(model2, model2a)
              df       AIC
model2  8.161188 -49.07263
model2a 8.161188 -49.07263
```

I graph this model and add it to the smooths of Fig. 7.

```
out.p2a <- predict(model2, newdata=data.frame(duration=1:58, code=rep(1,58)))
out.p2b <- predict(model2, newdata=data.frame(duration=1:78, code=rep(2,78)))
plot(removed~duration, data=bees)
points(bees$duration, bees$removed, col=bees$code, pch=c(1,16)[bees$code])
lines(1:58, out.p1a, lty=2)
lines(1:78, out.p1b, lty=2, col=2)
legend('bottomright', c('bumblebees', 'honeybees', 'additive GAM', 'interaction GAM'),
   pch=c(1,16,NA,NA), col=c(1,2,1,1), lty=c(NA,NA,2,1), lwd=c(1,1,1,2), cex=.9, bty='n')
lines(1:78, out.p2b, lwd=2, col='pink')
lines(1:58, out.p2a, lwd=2, col='grey70')
```

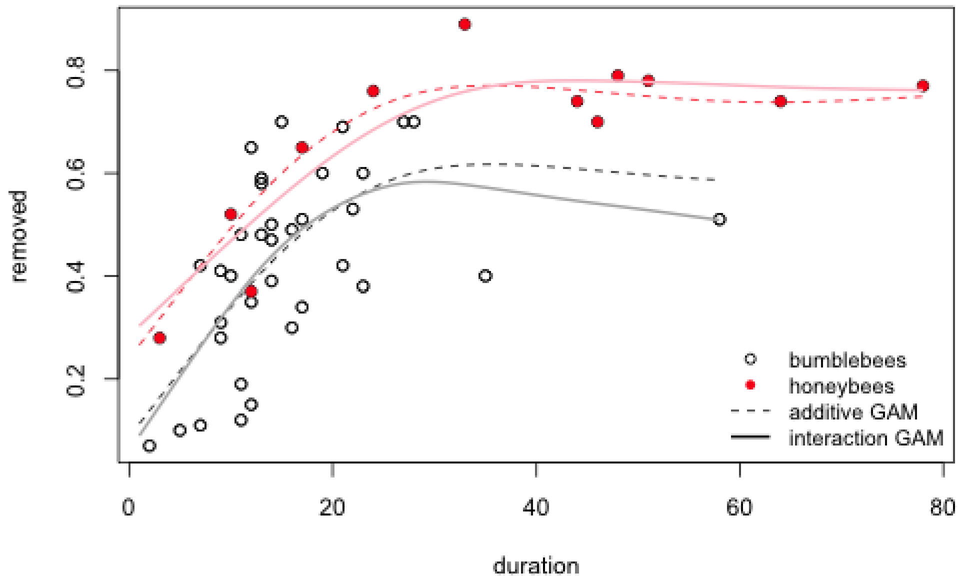


Fig. 8 Interaction model between a factor variable and a smooth

We can compare these models using AIC. I include the common smooth model fit in the previous section for a baseline comparison.

```
model2 <- gam(removed~factor(code)+s(duration, by=factor(code)), data=bees)
AIC(model0, model1, model2)
             df       AIC
model0 4.669237 -44.23431
model1 6.168346 -51.59651
model2 8.161188 -49.07263
```

Because these models are fit using the GCV criterion, I'm not sure how seriously to take the AIC values reported here. To play it safe I refit the three models using maximum likelihood, **method='ML'**.

```
model0.mls <- gam(removed~s(duration), data=bees, method='ML')
model1.mls <- gam(removed~s(duration)+factor(code), data=bees, method='ML')
model2.mls <- gam(removed~factor(code)+s(duration, by=factor(code)), data=bees, method='ML')
AIC(model0.mls, model1.mls, model2.mls)
                 df       AIC
model0.mls 4.766599 -44.23703
model1.mls 6.317117 -51.60267
model2.mls 7.533869 -48.52325
```

So based on the GAMs we should prefer a model in which the smooth for honeybees is the same as that of bumblebees except the smooth of honeybees is shifted upward. We can compare these results to the comparable parametric models. I write two new functions that allow the asymptote and/or the scale parameter of the exponential function to vary depending on the value of the `code` variable.

```
myfunc2 <- function(x,z,a,b,b1) (b+b1*z)*(1-exp((a)*x))
myfunc3 <- function(x,z,a,b,a1,b1) (b+b1*z)*(1-exp((a+a1*z)*x))
#separate asymptotes
out.nls2 <- nls(removed~myfunc2(duration, I(code-1), a, b, b1), data=bees, start=list(a=-.1, b=.7,
   b1=0))
#separate rates and asymptotes
out.nls3 <- nls(removed~myfunc3(duration, I(code-1), a, b, a1, b1), data=bees, start=list(a=-.1,
   b=.7, a1=0, b1=0))
AIC(out.nls, out.nls2, out.nls3)
         df       AIC
out.nls   3 -48.27628
out.nls2  4 -54.92918
out.nls3  5 -52.95404
#compare fit of the two models
anova(out.nls2, out.nls3)
Analysis of Variance Table

Model 1: removed ~ myfunc2(duration, I(code - 1), a, b, b1)
Model 2: removed ~ myfunc3(duration, I(code - 1), a, b, a1, b1)
  Res.Df Res.Sum Sq Df     Sum Sq F value Pr(>F)
1     44    0.72134
2     43    0.72096  1 0.00038147  0.0228 0.8808

#summary table of the simpler model
summary(out.nls2)
Formula: removed ~ myfunc2(duration, I(code - 1), a, b, b1)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
a  -0.09472    0.01956  -4.842 1.62e-05 ***
b   0.59962    0.05696  10.527 1.34e-13 ***
b1  0.18386    0.05725   3.211  0.00247 **
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.128 on 44 degrees of freedom

Number of iterations to convergence: 6
Achieved convergence tolerance: 2.331e-06
```

Both AIC and statistical testing agree that the two kind of bees should have the same rate parameter *a*, but different asymptotes *b*. We can compare the parametric models to the GAMs using AIC. The parametric model ranks best.

```
AIC(out.nls, out.nls2, out.nls3, model0.mls, model1.mls, model2.mls)
                 df       AIC
out.nls    3.000000 -48.27628
out.nls2   4.000000 -54.92918
out.nls3   5.000000 -52.95404
model0.mls 4.766599 -44.23703
model1.mls 6.317117 -51.60267
model2.mls 7.533869 -48.52325
```

I plot the best GAM and the best parametric model superimposed on a scatter plot of the data.

```
plot(removed~duration, data=bees)
points(bees$duration, bees$removed, col=bees$code, pch=c(1,16)[bees$code])
lines(1:58, out.p1a, lty=2)
lines(1:78, out.p1b, lty=2, col=2)
legend('bottomright', c('bumblebees', 'honeybees', 'GAM', 'parametric'), pch=c(1,16,NA,NA),
   col=c(1,2,1,1), lty=c(NA,NA,2,1), lwd=c(1,1,1,2), cex=.9, bty='n')
curve(myfunc2(x,0, coef(out.nls2)[1], coef(out.nls2)[2], coef(out.nls2)[3]), from=0, to=58, add=T,
   col='grey70', lwd=2)
curve(myfunc2(x, 1, coef(out.nls2)[1], coef(out.nls2)[2], coef(out.nls2)[3]), add=T, col='pink',
   lwd=2)
```

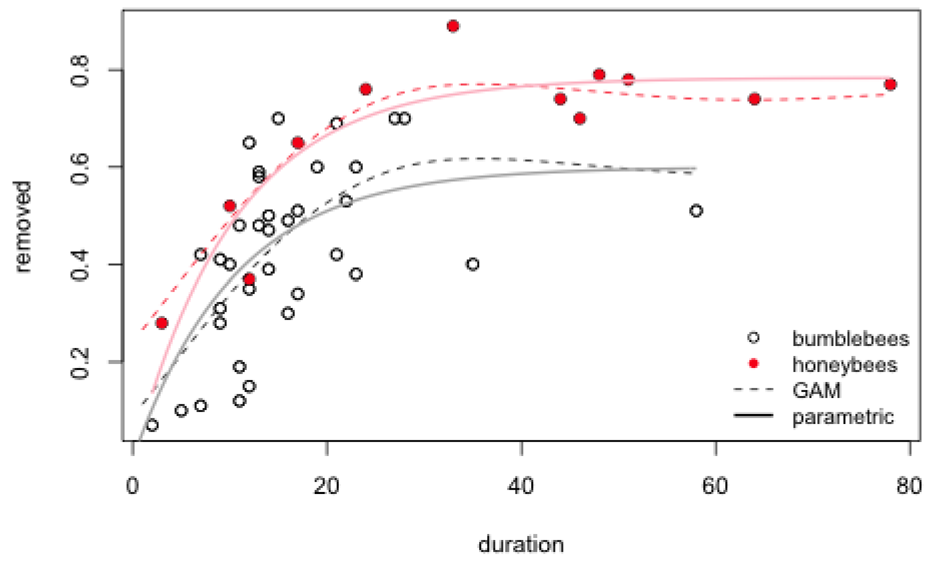


Fig. 9 Best parametric and GAM models for the bee data

Using a GAM to choose a transformation of a predictor

As an example of using a GAM to determine transformations of predictors for a parametric model we examine habitat suitability data for the rare Corroboree frog of Australia. The data are from a student's Master's thesis at the University of Canberra, Hunter (2000), and can be found in the **DAAG** library as the data set `frogs`. Additional analysis of these data can be found in Chapter 8 of Maindonald and Braun (2003). The variables included in frogs.txt are the following:

1. pres.abs: 0 = frogs were absent, 1 = frogs were present
2. northing: reference point
3. easting: reference point
4. altitude: altitude, in meters
5. distance: distance in meters to nearest extant population
6. NoOfPools: number of potential breeding pools
7. NoOfSites: number of potential breeding sites within a 2 km radius
8. avrain: mean rainfall for Spring period
9. meanmin: mean minimum Spring temperature
10. meanmax: mean maximum Spring temperature

The response variable of interest is pres.abs and to simplify things we'll look at just four of the nine predictors: distance, NoOfPools, meanmin, and meanmax.

```
frogs <- read.csv("ecol 562/frogs.txt")
```

```
names(frogs)
 [1] "pres.abs"  "northing"  "easting"   "altitude"  "distance"  "NoOfPools" "NoOfSites" "avrain"
 [9] "meanmin"   "meanmax"
```

I verify that there are enough unique values of each predictor to justify fitting a smooth.

```
apply(frogs[, c("distance", "NoOfPools", "meanmin", "meanmax")], 2, function(x) length(unique(x)))
 distance NoOfPools   meanmin   meanmax
       32        60        61        83
```

Because the response is binary, I carry out logistic regression by specifying the **family=binomial** argument of **gam**.

```
library(mgcv)
out.gam <- gam(pres.abs~s(distance) + s(NoOfPools) + s(meanmin) + s(meanmax), data=frogs,
    family=binomial)
```

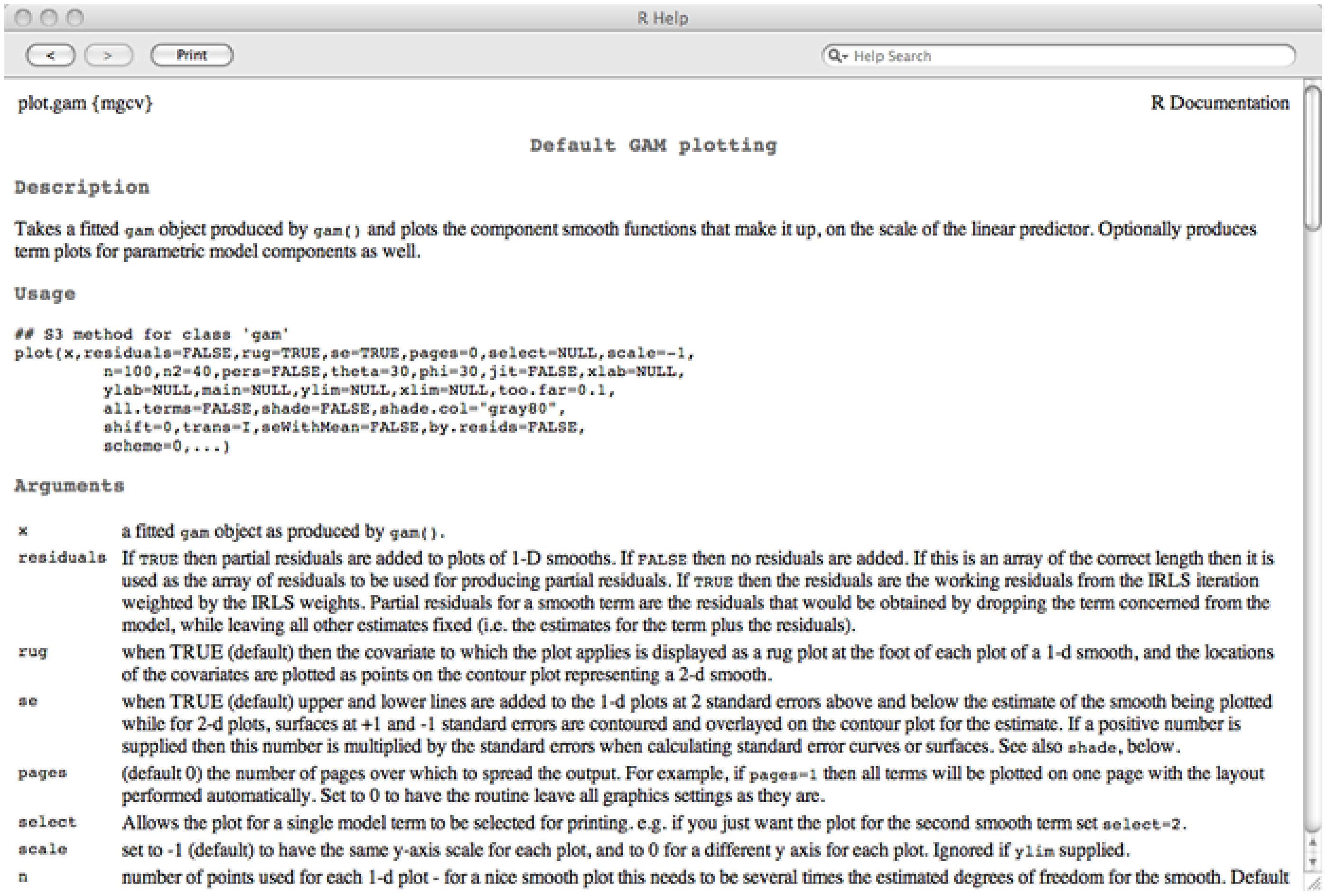
The **plot** function has a method for **gam** objects. To see a list of available methods for **plot** use the **methods** function on **plot**.

```
methods(plot)
 [1] plot.acf*              plot.ACF*              plot.augPred*
 [4] plot.compareFits*      plot.data.frame*       plot.decomposed.ts*
 [7] plot.default           plot.dendrogram*       plot.density
[10] plot.ecdf              plot.factor*           plot.formula*
[13] plot.function          plot.gam               plot.gls*
[16] plot.hclust*           plot.histogram*        plot.HoltWinters*
[19] plot.intervals.lmList* plot.isoreg*           plot.lm
[22] plot.lme*              plot.lmList*           plot.medpolish*
[25] plot.mlm               plot.nffGroupedData*   plot.nfnGroupedData*
[28] plot.nls*              plot.nmGroupedData*    plot.pdMat*
[31] plot.ppr*              plot.prcomp*           plot.princomp*
[34] plot.profile.nls*      plot.ranef.lme*        plot.ranef.lmList*
[37] plot.shingle*          plot.simulate.lme*     plot.spec
[40] plot.stepfun           plot.stl*              plot.table*
[43] plot.trellis*          plot.ts                plot.tskernel*
[46] plot.TukeyHSD          plot.Variogram*

   Non-visible functions are asterisked
```

Notice that **plot.gam** is listed. This is a plot method for **gam** objects and has a number of options specific for displaying GAMs. To see these options bring up the help screen for the **plot.gam** method.

```
?plot.gam
```



plot.gam {mgcv} R Documentation

Default GAM plotting

Description

Takes a fitted gam object produced by gam() and plots the component smooth functions that make it up, on the scale of the linear predictor. Optionally produces term plots for parametric model components as well.

Usage

```
## S3 method for class 'gam'
plot(x,residuals=FALSE,rug=TRUE,se=TRUE,pages=0,select=NULL,scale=-1,
         n=100,n2=40,pers=FALSE,theta=30,phi=30,jit=FALSE,xlab=NULL,
         ylab=NULL,main=NULL,ylim=NULL,xlim=NULL,too.far=0.1,
         all.terms=FALSE,shade=FALSE,shade.col="gray80",
         shift=0,trans=I,seWithMean=FALSE,by.resids=FALSE,
         scheme=0,...)
```

Arguments

`x`	a fitted `gam` object as produced by `gam()`.
`residuals`	If `TRUE` then partial residuals are added to plots of 1-D smooths. If `FALSE` then no residuals are added. If this is an array of the correct length then it is used as the array of residuals to be used for producing partial residuals. If `TRUE` then the residuals are the working residuals from the IRLS iteration weighted by the IRLS weights. Partial residuals for a smooth term are the residuals that would be obtained by dropping the term concerned from the model, while leaving all other estimates fixed (i.e. the estimates for the term plus the residuals).
`rug`	when TRUE (default) then the covariate to which the plot applies is displayed as a rug plot at the foot of each plot of a 1-d smooth, and the locations of the covariates are plotted as points on the contour plot representing a 2-d smooth.
`se`	when TRUE (default) upper and lower lines are added to the 1-d plots at 2 standard errors above and below the estimate of the smooth being plotted while for 2-d plots, surfaces at +1 and -1 standard errors are contoured and overlayed on the contour plot for the estimate. If a positive number is supplied then this number is multiplied by the standard errors when calculating standard error curves or surfaces. See also `shade`, below.
`pages`	(default 0) the number of pages over which to spread the output. For example, if `pages=1` then all terms will be plotted on one page with the layout performed automatically. Set to 0 to have the routine leave all graphics settings as they are.
`select`	Allows the plot for a single model term to be selected for printing. e.g. if you just want the plot for the second smooth term set `select=2`.
`scale`	set to -1 (default) to have the same y-axis scale for each plot, and to 0 for a different y axis for each plot. Ignored if `ylim` supplied.
`n`	number of points used for each 1-d plot - for a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth. Default

Fig. 10 Help screen for the plot.gam method

Two especially useful options are **pages** and **select**. To get all of the smooths to display on a single page we need to add the argument **pages=1** to the **plot.gam** call.

```
plot(out.gam, pages=1)
```

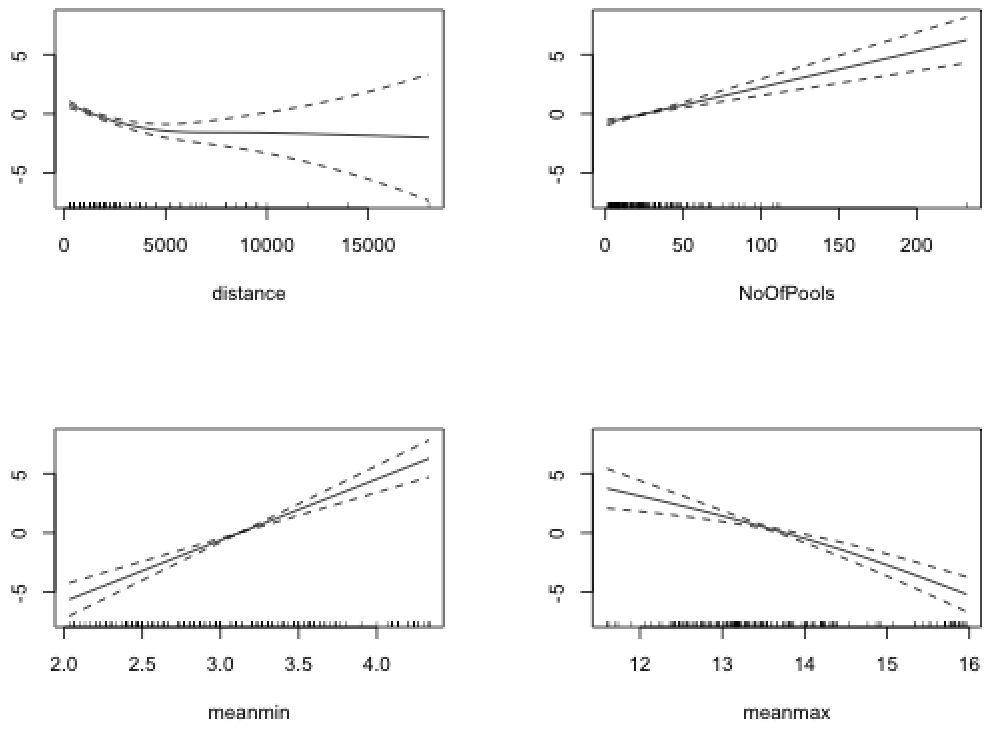


Fig. 11 Estimated partial response functions from the GAM

Each displayed object is the partial response function as estimated by **gam**. The graphs are analogous to plotting individual regression terms such as $\beta_j x_j$ in an ordinary regression model. Fig. 11 suggests that the relationship of the logit with NoOfPools and meanmin is probably linear. The summary table provides additional information.

```
summary(out.gam)
Family: binomial
Link function: logit

Formula:
pres.abs ~ s(distance) + s(NoOfPools) + s(meanmin) + s(meanmax)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.9335     0.2341  -3.987 6.68e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
               edf Ref.df Chi.sq  p-value
```

```
s(distance)   1.794  2.247  8.455 0.018951 *
s(NoOfPools) 1.000  1.000 10.429 0.001241 **
s(meanmin)   1.000  1.000 16.145 5.87e-05 ***
s(meanmax)   1.772  2.224 17.418 0.000225 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.363   Deviance explained =   31%
UBRE score = -0.026688  Scale est. = 1         n = 212
```

The estimated degrees of freedom of the meanmax and distance smoothers exceed 1, while the edf of the smooths for NoOfPools and meanmin is 1, confirming linearity. The graph of the smooth of distance in Fig. 11 resembles an inverted logarithm suggesting log(distance) might be a good choice for a parametric term. (The large range of distance also suggests a logarithmic transformation might be appropriate.) I replace the nonparametric smooth of distance with log(distance) in the GAM.

```
out.gam1 <- gam(pres.abs~log(distance) + s(NoOfPools) + s(meanmin) + s(meanmax), data=frogs,
    family=binomial)
```

I compare the two models using AIC.

```
AIC(out.gam, out.gam1)
               df      AIC
out.gam  6.565388 206.3422
out.gam1 5.789663 204.1376
```

The model with log(distance) has a lower AIC. We can perhaps obtain a better estimate of AIC by refitting the two models using maximum likelihood estimation, **method="ML"**.

```
out.gam.ml <- gam(pres.abs~s(distance) + s(NoOfPools) + s(meanmin) + s(meanmax), data=frogs,
    family=binomial, method="ML")
out.gam1.ml <- gam(pres.abs~log(distance) + s(NoOfPools) + s(meanmin) + s(meanmax), data=frogs,
    family=binomial, method="ML")
AIC(out.gam.ml, out.gam1.ml)
                  df      AIC
out.gam.ml  5.000011 209.6275
out.gam1.ml 5.000013 205.1332
```

AIC still prefers the semi-parametric model with log(distance) instead of a smoothed distance. The coefficient of the log term is the second entry in the coefficient vector from the model.

```
coef(out.gam1)[1:6]
   (Intercept)  log(distance)  s(NoOfPools).1  s(NoOfPools).2  s(NoOfPools).3  s(NoOfPools).4
  4.717276e+00  -8.066707e-01  -1.704321e-05    6.134517e-06  -4.454965e-06    5.878728e-06
```

We can compare the two fits by superimposing the log term on the graph of the smooth. Because these are marginal response functions the vertical scales are not be the same for these two terms. I choose a value of the intercept for the parametric term so that the two curves overlap. By inspecting the graph and plotting points by trial and error I discover that the first plotted point of the smooth is approximately (250, .8).

```
plot(out.gam, select=1)
min(frogs$distance)
[1] 250
points(250, .8, pch=16, cex=.5, col=2)
#calculate the intercept that yields 0.8 when distance = 250
.8-coef(out.gam1)[2]*log(250)
log(distance)
    5.254001
curve(5.254+coef(out.gam1)[2]*log(x), add=T, col=2)
```

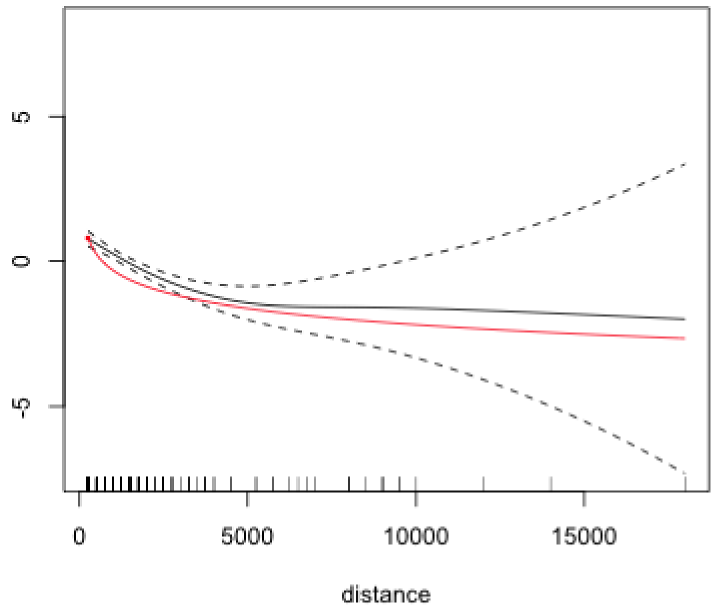


Fig. 12 Superimposing a parametric and a nonparametric smooth

A simpler way to overlay the two graphs is to use the **shift** argument of **plot.gam** (the plot method for gam objects) and to specify for this argument the intercept of the gam model.

```
plot(out.gam, select=1, shift=coef(out.gam)[1])
```

For the breakpoint function we do the same thing.

```
curve(coef(out.gam1)[2]*log(x) + coef(out.gam1)[1], add=T, col=2)
```

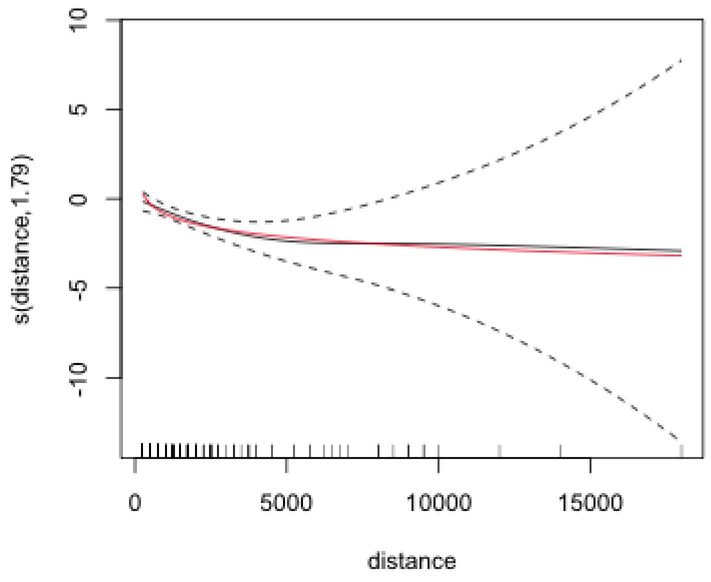


Fig. 13 Superimposing a parametric and a nonparametric smooth, method 2

For meanmax I try a quadratic function.

```
out.gam2 <- gam(pres.abs~s(distance) + s(NoOfPools) + s(meanmin) + meanmax + I(meanmax^2),
   data=frogs, family=binomial)
AIC(out.gam,out.gam2)
               df      AIC
out.gam  6.565388 206.3422
out.gam2 7.576463 205.8462
```

So there is some evidence that we can replace the smooth with a quadratic. To obtain a more reliable result we should compare models fit with maximum likelihood.

```
out.gam2.ml <- gam(pres.abs~s(distance) + s(NoOfPools) + s(meanmin) + meanmax + I(meanmax^2),
   data=frogs, family=binomial, method="ML")
AIC(out.gam.ml, out.gam2.ml)
                  df      AIC
out.gam.ml  5.000011 209.6275
out.gam2.ml 6.000007 208.1088
```

Using the GAM we have obtained possible structural forms for the predictors for use in a parametric model: the logarithm of distance and a quadratic model for meanmax.

Generalized additive mixed effects models

The data set carib.csv is taken from Selig and Bruno (2010) who compiled a comprehensive global database to compare long-term changes (1969 to 2006) in coral cover from 5170 independent surveys inside 310 MPAs (marine protected areas) around the world and 3364 surveys of unprotected reefs. Surveys were from 4456 reefs across 83 different countries, although a few well-surveyed countries were better represented in the data set. 993 of the surveys were repeated at least twice with 306 in the Caribbean and 687 in the Indo-Pacific. The surveys in the database were conducted across more than 40 years in the Caribbean and more than 30 years in the Indo-Pacific. In today's class we restrict ourselves to the data obtained from reefs in the Caribbean.

```
carib <- read.csv("ecol 562/carib.csv")
names(carib)
 [1] "REEF_ID_GR"     "UNIQUE_POO"     "SURVEY_ID"      "REEF_NAME"      "OCEAN"          "LOCATION"
 [7] "COUNTRY"        "LONGITUDE"      "LATITUDE"       "YEAR"           "DEPTH_AVE"      "DEPTH_MIN"
[13] "DEPTH_MAX"      "HARD_COR_P"     "SUBREGION"      "ALGAE_P"        "NOTES"          "D_SOURCE"
[19] "SUBREGION_"     "REGION"         "REGION_COD"     "OLDDIS"         "AREANAME"       "GISNAME"
[25] "DESIGNATE"      "IUCNCAT"        "IUCNCAT_PO"     "SITE_CODE"      "TERRE"          "MPA_POLY"
[31] "MPA"            "LON_CEN"        "LAT_CEN"        "YEAR_MPA2"      "YEAR_MPA"       "AREA_M"
[37] "PERIMETER_"     "GRPMPA_ID"      "MPA_UID_ORIG"   "DISTANCE_ORIG"  "ORIG_MPA_UID"   "MPA_UID"
[43] "MPA_UID_SE"     "DISTANCE_T"     "logit"          "MPA_CODE"       "YEAR.PROT"      "FIRST.SURVEY"
[49] "z"
```

Only a few of the variables are of interest to us today. I extract those below and display the first four observations of each.

```
carib[1:4, c(1,4,7,14,45,42,35,10,49)]
  REEF_ID_GR        REEF_NAME COUNTRY HARD_COR_P     logit MPA_UID YEAR_MPA YEAR z
1      10006 East Snake Caye  Belize     15.625 -1.686399    3314     2000 2004 4
2      10007     Pelican Reef  Belize      6.250 -2.708050    2162     1996 2004 8
3      10008 Bungy Back Side  Belize     20.625 -1.347680    2162     1996 2004 8
4      10009 Seal Caye Point  Belize      5.625 -2.820055    2162     1996 2004 8
```

The data consist of annual surveys of coral reefs carried over various years in the Caribbean. At each survey an estimate was made of the percent of the reef occupied by living coral (HARD_COR_P). This was recorded as a number between 0 and 100. A bounded response can pose a challenge in regression models. To get around this coral cover was converted to an unbounded number by taking its logit.

$$\operatorname{logit}(\text{Hard\_Cor\_P}) = \log\left(\frac{\text{Hard\_Cor\_P}}{100 - \text{Hard\_Cor\_P}}\right)$$

Coral cover has been declining in the Caribbean over time. To see if the protection of coral reefs in marine sanctuaries is having a positive effect, the trend in coral cover was modeled as a function of the amount of time since the marine sanctuary was created (YEAR_MPA). This is the variable z in the data set.

$$z = \text{YEAR} - \text{YEAR\_MPA}$$

When fitting regression models to these data we need to account for the structure of the data set. At its simplest this is a repeated measures design with multiple annual measurements coming from the same reef (REEF_ID_GR). The data are extremely unbalanced with most of the reefs providing only a single year of data. Still, 95 reefs provided 11 years of data and one reef provided 17 years of data.

```
table(table(carib$REEF_ID_GR))
  1    2    3    4    5    6    7    9   10   11   12   15   17 
508   36   17   10   40    3    6    6   95    1    1    1    1 
```

Many of the reefs are located in the same marine sanctuary (MPA_UID). We might expect that because of their relative proximity and similar management strategies that reefs from the same sanctuary might have a similar response to protection.

The repeated measures design coupled with spatial clustering of reefs leads to observational heterogeneity at three levels. To account for the repeated measures aspect we can assign observations coming from the same reef the same random effect: u_{0ij}, where j denotes the reef and i denotes the marine sanctuary containing that reef. To link reefs coming from the same marine sanctuary we can assign them a common random effect, u_{0i}. In multilevel model notation we can write the three-level random intercepts model as follows.

$$\begin{aligned}\text{Level 1:}\quad & \operatorname{logit} p_{ijk} = \beta_{0ij} + \beta_1 z_{ijk} + \varepsilon_{ijk}\\ \text{Level 2:}\quad & \beta_{0ij} = \beta_{0i} + u_{0ij}\\ \text{Level 3:}\quad & \beta_{0i} = \beta_0 + u_{0i}\end{aligned}$$

where i denotes the sanctuary, j the reef in that sanctuary, and k the annual observation on that reef. Each of the error terms is assumed to be drawn from a normal distribution.

$$\varepsilon_{ijk} \sim \text{Normal}(0, \sigma^2),\, u_{0ij} \sim \text{Normal}(0, \tau^2),\, u_{0i} \sim \text{Normal}(0, \psi^2)$$

Written as a single composite model we have the following.

$$\operatorname{logit} p_{ijk} = \beta_0 + \beta_1 z_{ijk} + u_{0i} + u_{0ij} + \varepsilon_{ijk}$$

The **gamm** function of the **mgcv** package can be used to fit generalized additive mixed effects models. The **gamm** function calls the **lme** function of the **nlme** package to fit the model. In the **nlme** package there are two different ways to represent a 3-level model with random intercepts at each level. One version is the following.

```
random = ~1|MPA_UID/REEF_ID_GR
```

Notice that the larger grouping variable, in this case the marine sanctuary MPA_UID, appears before the smaller grouping variable, the reef REEF_ID_GR and the two of them are separated by a forward slash, /. This notation forces the same coefficient, in this case the intercept, to be random at each level. A more flexible notation also supported by **lme** that can be used to specify different random effects at each level is the following.

```
random = list(MPA_UID=~1, REEF_ID_GR=~1)
```

The second version of the notation is the only version supported by the **gamm** function. To fit a generalized additive mixed effects model for the logit in which we include a smooth of the predictor z we would enter the following.

```
out.carib <- gamm(logit~s(z), random=list(MPA_UID=~1, REEF_ID_GR=~1), data=carib, method="ML")
names(out.carib)
[1] "lme" "gam"
```

Two components are returned: an **lme** component and a **gam** component. The **lme** component contains mostly details from the **lme** fit that are of little interest to us. It does contain the AIC and log-likelihood of the model though. The **gam** component contains information about the smooth and it is the component that we need to plot to visualize the smooth.

```
plot(out.carib$gam)
```

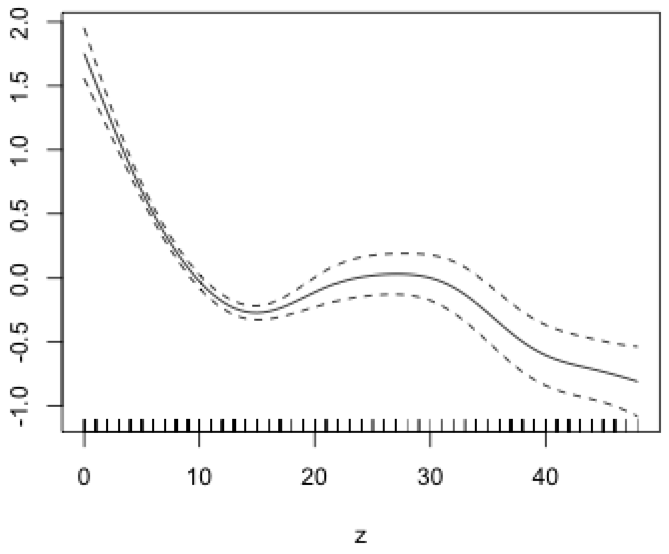


Fig. 14 Estimated partial response function for the predictor z

The plot suggests that initially coral cover continued its decline but that at approximately 15 years of protection things began to turn around. Then inexplicably at around 30 years of protection the decline began anew. The fitted smooth looks remarkably like a cubic although the reported degrees of freedom of the smooth exceeds six.

```
summary(out.carib$gam)
```

```
Family: gaussian
Link function: identity

Formula:
logit ~ s(z)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.94830    0.09755  -19.97   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
       edf Ref.df     F p-value
s(z) 6.427  6.427 59.65  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.0157  Scale est. = 0.22176   n = 1990
```

The column labeled edf is the estimated degrees of freedom. This can be thought of as analogous to the degree of a polynomial. When edf = 1 the relationship is linear. If the reported estimated degrees of freedom approaches the default dimension for the spline basis, *k* = 10, as it does here one should refit the model with a larger value for *k* using the **k=** argument of the smooth.

```
out.carib2 <- gamm(logit~s(z,k=15), random=list(MPA_UID=~1, REEF_ID_GR=~1), data=carib, method="ML")
summary(out.carib2$gam)
Family: gaussian
Link function: identity

Formula:
logit ~ s(z, k = 15)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.95104    0.09785  -19.94   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
       edf Ref.df     F p-value
s(z) 9.472  9.472 37.27  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.0205  Scale est. = 0.22003   n = 1990
```

The estimated degrees of freedom has increased substantially, but when we examine the smooth we see it has the same basic form as before but with more wiggles.

```
plot(out.carib2$gam)
```

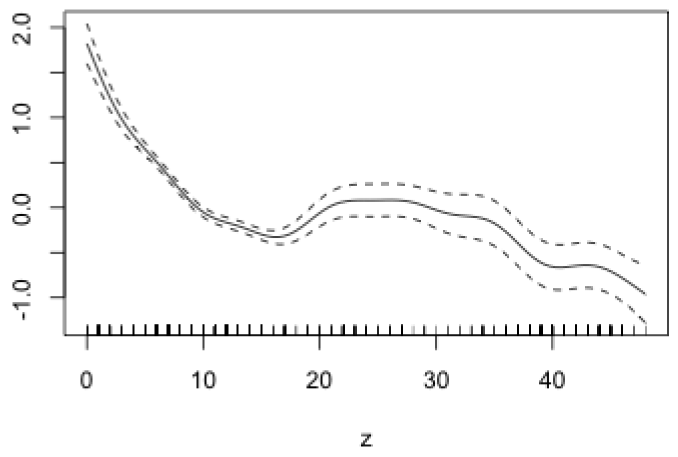


Fig. 15 Estimated partial response function for the predictor z using k = 15 knots

If we increase the dimension further, say $k = 20$, the edf continues to increase (edf = 13.64) and the curve wiggles even more. Part of this may be due to the fact that z is discrete.

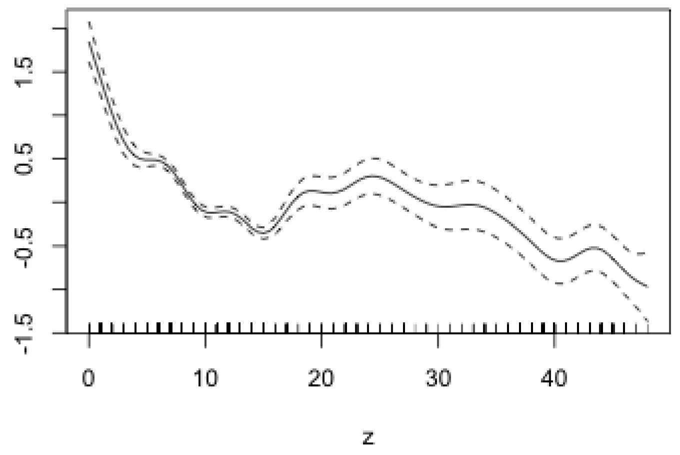


Fig. 16 Estimated partial response function for the predictor z using k = 20 knots

I next try replacing the smooth with a cubic.

```
out.carib.cubic <- lme(logit~z+I(z^2)+I(z^3), random=list(MPA_UID=~1, REEF_ID_GR=~1), data=carib,
   method="ML")
AIC(out.carib.cubic, out.carib$lme, out.carib2$lme)
                df      AIC
out.carib.cubic  7 4447.960
out.carib$lme    6 4449.802
out.carib2$lme   6 4445.780
```

The cubic model has a lower AIC than the first GAM with a smooth with a basis dimension of 10 but a larger AIC than the second GAM with a smooth with a basis dimension of 15, suggesting that a cubic might be a more parsimonious model. Because there is a single predictor in the model we can compare the smooth with the cubic by plotting the predictions from each model on the same graph. I plot the population-level model for each, **level = 0**. In order for the plot to display properly we need to sort the predicted values in order of the *x*-axis variable z. This needs to be done for the predictions from both models.

```
gam.p <- predict(out.carib$gam, level=0)
out10 <- cbind(gam.p, carib$z)
out10 <- out10[order(out10[,2]),]
colnames(out10) <- c('y', 'x')
lme.p <- predict(out.carib.cubic, level=0)
out11 <- cbind(lme.p, carib$z)
out11 <- out11[order(out11[,2]),]
colnames(out11) <- c('y', 'x')
```

Finally we can plot things.

```
plot(y~x, data=out10, type='l', ylab='Logit', xlab='Years protected')
lines(out11[,'x'], out11[,'y'], col=2, lty=2)
legend('topright', c('GAM','cubic'), col=c(1,2), lty=c(1,2), cex=.9, bty='n')
```

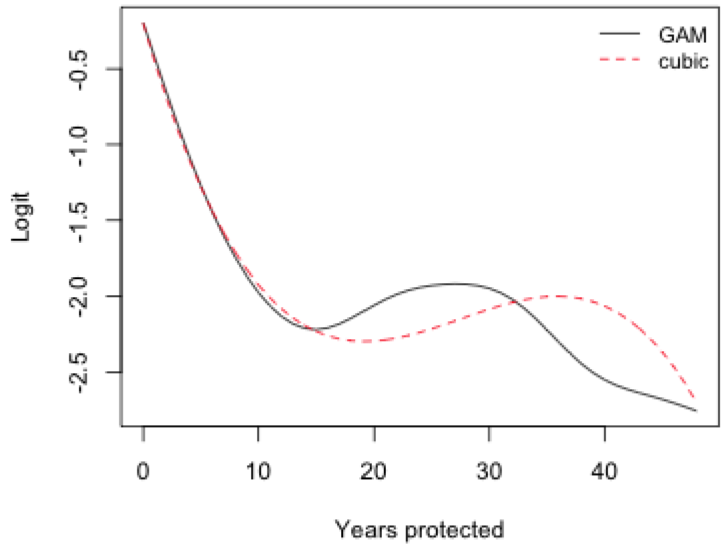


Fig. 17 GAM smoother (10 knots) and cubic function

But there's a problem here that becomes apparent if we add to the graph a bar chart that indicates the distribution of the data. Because the bar chart is on a different scale from the logit graph, I add it by overlaying a second plot on the top of the logit plot. This is accomplished with the global graphics parameter setting **new=T** to **par** which prevents a subsequent call to **plot** from erasing the current graph. To make sure the graphs line up I explicitly specify the same **xlim** argument for both. I use the **ylim** argument in the second graph to scale the height of the bars in the bar plot.

```
plot(y~x, data=out10, type='l', ylab='Logit', xlab='Years protected', xlim=c(0,50))
lines(out11[,'x'], out11[,'y'], col=2, lty=2)
legend('topright', c('GAM','cubic'), col=c(1,2), lty=c(1,2), cex=.9, bty='n')
#add bar plot
par(new=T)
par(lend=1)
plot(as.numeric(names(table(carib$z))), table(carib$z), type='h', axes=F, ylim=c(0,500), lwd=4,
   col='grey70', xlab='', ylab='', xlim=c(0,50))
par(new=F)
```

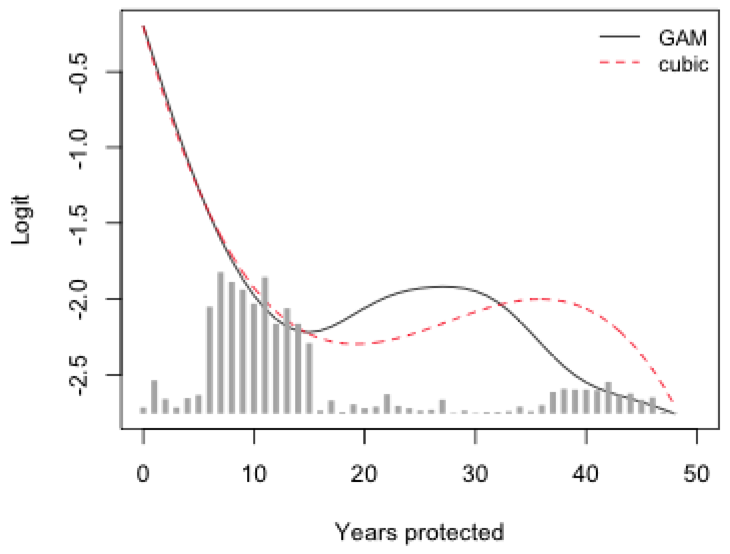


Fig. 18 Plot of the GAM along with distribution of the data

The graph shows that the point where the smooth begins its rebound in coral cover corresponds to a portion of the graph where we have very little data. So, it is quite possible the upturn is not real, or at least not reliable. This is one of the dangers of fitting generalized additive models when there are gaps in the data. An alternative to the GAM picture is one continuous decline or perhaps two separate declines but with different slopes. The latter is called a breakpoint regression model. To fit it we need to determine the location of the breakpoint.

A breakpoint regression term in the variable *z* with breakpoint at *z* = *c* has the following form: `z + (z-c)*(z>c)`. From the graph the breakpoint appears to occur somewhere between *z* = 5 and *z* = 25. I fit a separate breakpoint model at each integer value in this interval. Because of a peculiarity in the way **lme** fits models, I need to add the putative breakpoint as a column in the data frame before I fit the model. This should not be necessary and is not required when fitting models using, e.g., the **lm** function.

```
my.aic2 <- NULL
for(i in 5:25){
   #add breakpoint to data frame--should not be necessary
   carib$c <- i
   trial <- lme(logit~z+I((z-c)*(z>c)), random=list(MPA_UID=~1, REEF_ID_GR=~1), data=carib,
      method='ML')
   my.aic2 <- rbind(my.aic2, c(i, AIC(trial)))
}
which.min(my.aic2[,2])
[1] 5
my.aic2[5,]
[1] 9.000 4436.615
```

Observe that the AIC of the breakpoint model is much lower than that of the cubic model and either GAM. (Note: the correct AIC of the breakpoint model is 4436.615 + 2 = 4438.615, because we should count the estimated breakpoint as a parameter.) I add the breakpoint model to Fig. 18.

```
bp.model <- lme(logit~z+I((z-9)*(z>9)), random=list(MPA_UID=~1, REEF_ID_GR=~1), data=carib,
   method='ML')
plot(y~x, data=out10, type='l', ylab='Logit', xlab='Years protected', xlim=c(0,48), ylim=c(-3.5,0))
#cubic model
lines(out11[,'x'], out11[,'y'], col=2, lty=2)
#breakpoint model
curve(fixef(bp.model)[1] + fixef(bp.model)[2]*x + fixef(bp.model)[3]*(x-9)*(x>9), add=T, col=3,
   lwd=2)
legend('topright', c('GAM', 'cubic', 'breakpoint'), col=c(1,2,3), lty=c(1,2,1), lwd=c(1,1,2),
   cex=.9, bty='n')
par(new=T)
par(lend=1)
plot(as.numeric(names(table(carib$z))), table(carib$z), type='h', axes=F, ylim=c(0,500), lwd=4,
   col='grey70', xlab='', ylab='', xlim=c(0,48))
par(new=F)
```

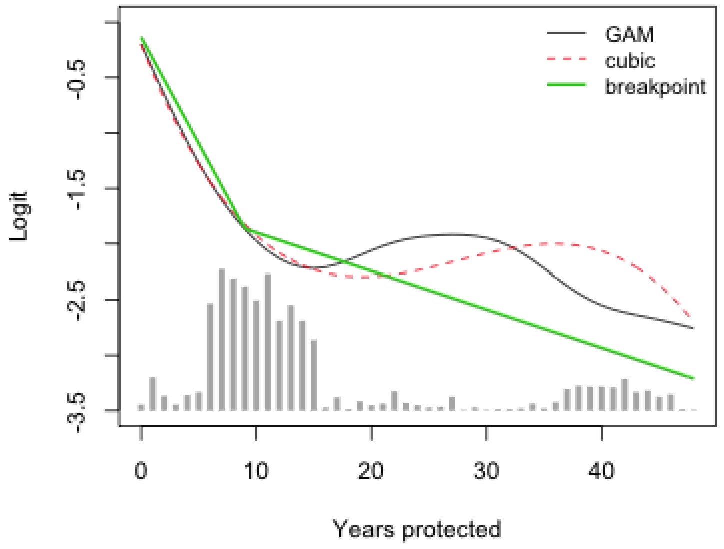


Fig. 19 Plot of the GAM along with breakpoint regression model

Cited References

- Harder, L. D. and J. D. Thompson. 1989. Evolutionary options for maximizing pollen dispersal of animal-pollinated plants. *American Naturalist* **133**: 323–344.
- Hunter, D. (2000) The conservation and demography of the southern corroboree frog (*Pseudophryne corroboree*). M.Sc. thesis, University of Canberra, Canberra.
- Maindonald, J., and J. Braun. 2003. *Data Analysis and Graphics Using R: An Example-based Approach.* Cambridge University Press.
- Selig E. R. and J. F. Bruno. 2010. A global analysis of the effectiveness of marine protected areas in preventing coral loss. *PLoS ONE* **5**(2): e9278.
- Wood, S. N. 2006. *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC Press, Boca Raton, FL.

Course Home Page

Jack Weiss
Phone: (919) 962-5930
E-Mail: jack_weiss@unc.edu
Address: Curriculum for the Environment and Ecology, Box 3275, University of North Carolina, Chapel Hill, 27599