CHAPTER 34

GENERALIZED ADDITIVE MODELS

Prior to Chapter 33, continuous explanatory variables were added to models as linear functions, linearized parametric transformations, or through various link functions. In all cases, an explicit or implicit assumption was made about the parametric form of the function to be fitted to the data (whether quadratic, logarithmic, exponential, reciprocal or whatever). As we have seen, however, non-parametric smoothers can capture the shape of a relationship between *y* and *x* without us prejudging the issue.

Generalized Additive Models extend the range of application of Generalized Linear Models by allowing non-parametric smoothers in addition to parametric forms combined with a range of link functions. Two smoothing functions are available: **s** and **lo**. These can be used on their own, or mixed with parametric functions like this:

```
y ~ s(x) + s(w) + s(z)
```

```
y ~ x + lo(w) + z
```

The first model has smoothed functions for all 3 of its continuous explanatory variables, using the s smoother, while the second fits parametric linear models for *x* and *z* and a non-parametric smooth function for *w* using the lo smoother. lo and s can be mixed in the same formula, but it is more efficient and less memory intensive to use one smoothing method or the other in any one **gam**. The error structure of a **gam** can be specified in the same way as a **glm** (families binomial, poisson and gamma can be used). A **gam** has many of the attributes of both **glm** and **lm**, and the output can be modified using **update**.

The model is fit by iteratively smoothing partial residuals in a process known as backfitting. The algorithm separates the parametric and non-parametric parts of the fit. Since the parametric part is estimated using weighted linear least squares within the backfitting algorithm, the object returned has all the components of a **glm**. Because of this, you can use all of the familiar methods like print, plot, summary, anova, predict, and fitted after a **gam** has been fit to data.

Degrees of freedom are approximated as penalties for the complexity of the non-parametetric curve fitted; the more complex the curve, the higher the penalty and the more d.f. lost. Note that in a **gam**, d.f. are real numbers rather than integers.

Fitting a non-parametric curve to data

This example concerns the dynamics of a free-ranging population of feral sheep on a remote Scottish island. The question of what factors cause the ups and downs in population size can only be investigated once a suitable form has been found to describe the dependence of population change (delta) on population size (the 'density dependence').

```
attach(Soaytest)
names(Soaytest)
```

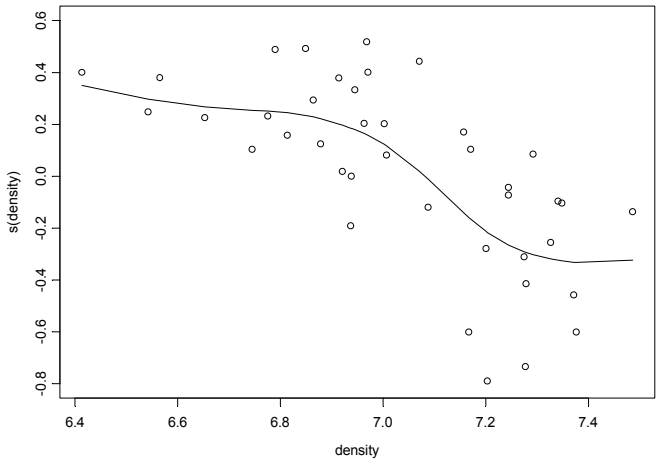
```
[1] "density" "jan"     "feb"     "mar"     "fm"      "all"    "delta"
```

We start by fitting a generalised additive model using the **s** smoother:

```
model<-gam(delta~s(density))
```

Now we can plot the smooth curve and add the data using points:

```
plot(model,ylim=c(-0.8,0.6),rugplot=F)
points(density,delta)
```



The rate of population change is obviously a non-linear function of population density, but there is no obvious parametric form for this relationship based on ecological theory, so we fit a non-parametric, smooth function as s(density)

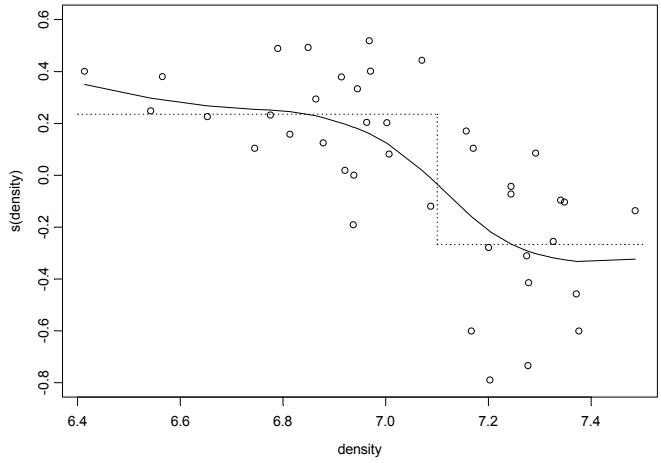
Note that the default **rugplot** has been switched off and the scale extended to allow space for all the data points. The output from a gam looks like this:

```
summary(model)
```

```
(Dispersion Parameter for Gaussian family taken to be
0.0624321 )
```

```
    Null Deviance: 4.689892 on 39 degrees of freedom

Residual Deviance: 2.18517 on 35.00073 degrees of freedom

Number of Local Scoring Iterations: 1

DF for Terms and F-values for Nonparametric Effects

             Df Npar Df   Npar F     Pr(F)
(Intercept)  1
 s(density)  1       3 2.242653 0.1005389
```

The column headed `Npar Df` contains the nonparametric degrees of freedom used up by the fit. Here is the non-parametric smoother with its estimated degrees of freedom = 4 compared with a 3-parameter step function (a threshold and two means):



Here are the statistics for the step function:

```
model2<-gam(delta~factor(density<7.1))
summary(model2)

 (Dispersion Parameter for Gaussian family taken to be
0.0584909 )

    Null Deviance: 4.689892 on 39 degrees of freedom

Residual Deviance: 2.222655 on 38 degrees of freedom
Df
          (Intercept)  1
factor(density < 7.1)  1
```

The step function has almost identical explanatory power (residual deviance 2.222 compared with 2.185) but saves on degrees of freedom. The step function indicates a highly significant effect of density on delta (t = 6.49), but the smoothed **gam** is not significant (p = 0.1).

The ozone data

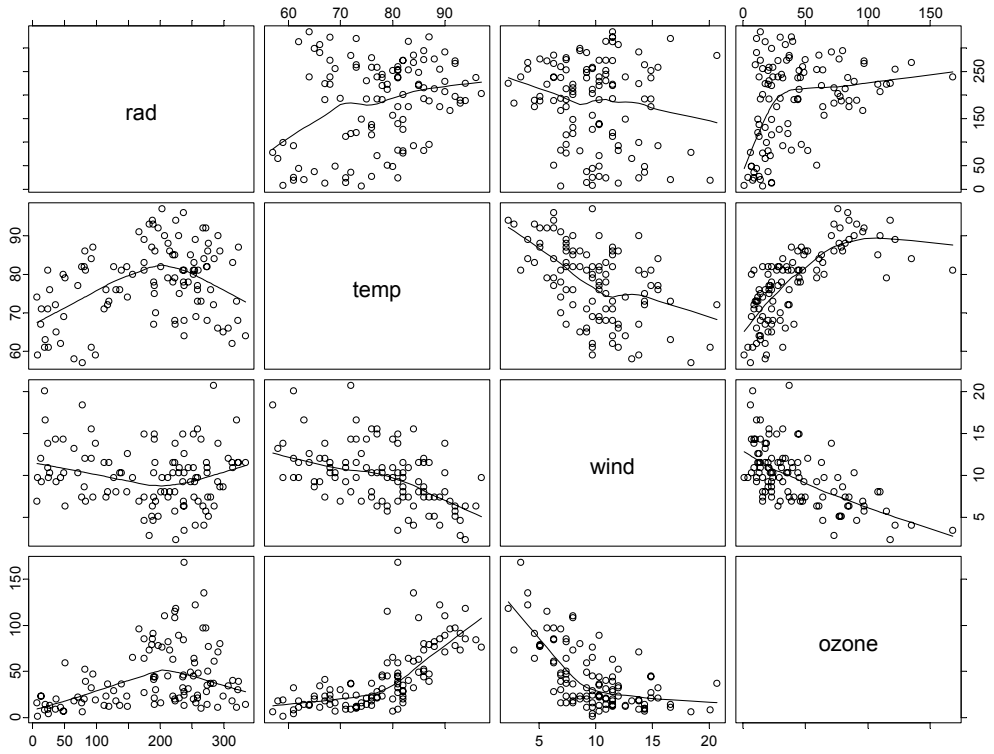
This data frame contains measurements of radiation, temperature, wind speed and ozone concentration. We want to model ozone concentration as a function of the three continuous explanatory variables using non-parametric smoothers rather than specified nonlinear functions.

```
attach(ozone.data)
names(ozone.data)
```

```
[1] "rad"   "temp"  "wind"  "ozone"
```

For data inspection we use **pairs** with a non-parametric smoother, **lowess**:

```
pairs(ozone.data, panel=function(x,y) { points(x,y); lines(lowess(x,y))} )
```



Now fit all 3 explanatory variables using the **s** smoother:

```
model<-gam(ozone~s(rad)+s(temp)+s(wind))
summary(model)
```

```
Call: gam(formula = ozone ~ s(rad) + s(temp) + s(wind))
Deviance Residuals:
       Min       1Q    Median       3Q      Max
 -45.70703 -10.0524 -3.357995 7.959326 73.13056

(Dispersion Parameter for Gaussian family taken to be
303.0611 )

    Null Deviance: 121801.9 on 110 degrees of freedom

Residual Deviance: 29699.72 on 97.99912 degrees of freedom

Number of Local Scoring Iterations: 1

DF for Terms and F-values for Nonparametric Effects

            Df Npar Df   Npar F     Pr(F)
(Intercept)  1
     s(rad)  1       3 2.050060 0.1117955
    s(temp)  1       3 6.677994 0.0003792
    s(wind)  1       3 9.246513 0.0000192
```

Radiation appears to be non significant, so we should delete it (model2) and compare the two models

```
model2<-gam(ozone~s(temp)+s(wind))
anova(model,model2,test="F")
```

```
Analysis of Deviance Table

Response: ozone

                     Terms Resid. Df Resid. Dev    Test        Df  Deviance  F Value
Pr(F)
1 s(rad) + s(temp) + s(wind)   97.9991   29699.72
2          s(temp) + s(wind)  102.0018   34496.71 -s(rad) -4.002692 -4796.984 3.954448
0.005100486
```

The deletion of radiation caused a highly significant increase in deviance ($p = 0.005$), emphasising the fact that deletion is a better test than inspection of parameters (above).

We should investigate the possibility that there is an interaction between wind and temperature. We can not fit interaction terms directly in **gam** so we calculate the product

```
wt<-wind*temp
```

then add this to the model:

```
model3<-gam(ozone~s(temp)+s(wind)+s(rad)+s(wt))
summary(model3)
```

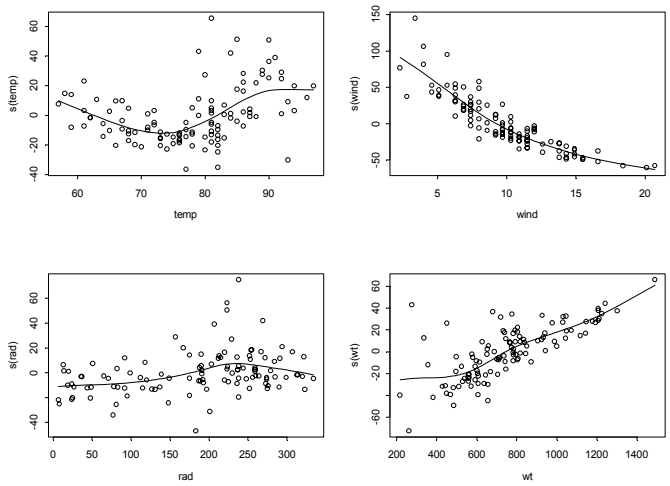
```
Call: gam(formula = ozone ~ s(temp) + s(wind) + s(rad) +
s(wt))
Deviance Residuals:
       Min        1Q    Median      3Q      Max
 -47.74943 -9.30572 -2.497074 7.06667 67.54792

(Dispersion Parameter for Gaussian family taken to be
295.5693 )

    Null Deviance: 121801.9 on 110 degrees of freedom

Residual Deviance: 27782.91 on 93.99798 degrees of freedom

Number of Local Scoring Iterations: 1

DF for Terms and F-values for Nonparametric Effects

            Df Npar Df   Npar F     Pr(F)
(Intercept)  1
    s(temp)  1       3 7.170581 0.0002192
    s(wind)  1       3 5.571937 0.0014584
     s(rad)  1       3 2.025007 0.1156243
      s(wt)  1       3 2.500044 0.0642082
```

The interaction appears to be of borderline significance (p = 0.064), so we try **anova,** comparing model (without the interaction) with model3 (including it):

```
anova(model,model3,test="F")
```

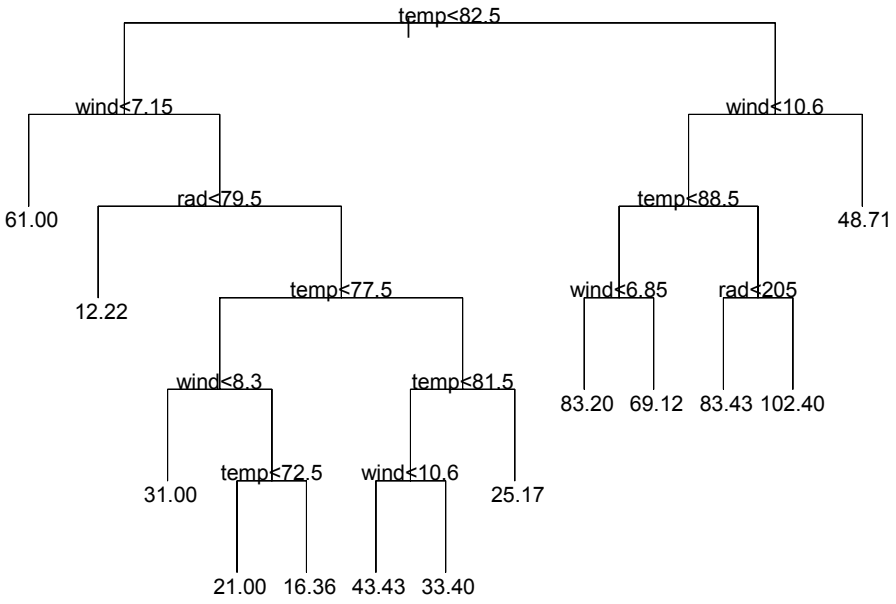
This gives p = 0.1754421, nowhere close to significance, reinforcing the view that model simplification is preferable. We can inspect the fit of model3 like this:

```
par(mfrow=c(2,2))
plot.gam(model3,resid=T,rugplot=F)
```



For comparison, here is the regression tree for these ozone data:

```
model<-tree(ozone~.,ozone.data)
plot(model,type="u")
text(model)
```



This shows that radiation is important, but only during periods of high wind at relatively low temperatures: then low levels of radiation are associated with very low levels of ozone (mean = 12.22). Likewise, at high temperatures on relatively still days, the level of

ozone is positively correlated with radiation (up to a mean of 102.4 at the highest levels of radiation (> 205)). This comparison of **gam** and **tree** reinforces the view that regression trees are excellent tools for exposing high order interaction effects in complex data sets.

3D graphics output with gam

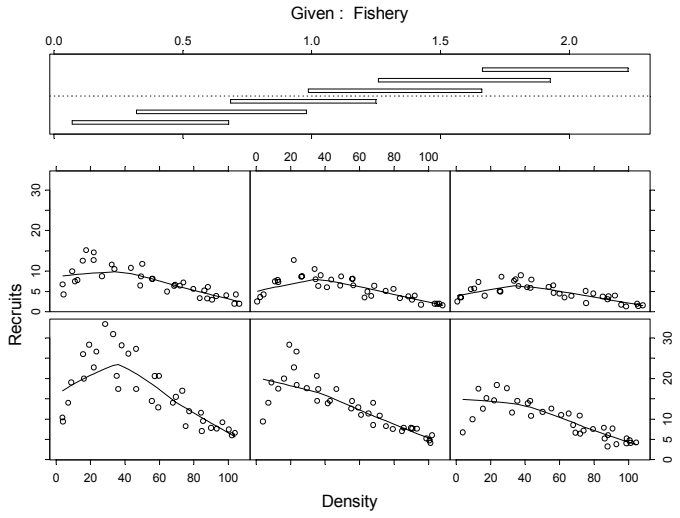
Finally, we look at the production of 3-D surfaces from **gams**. The trick here is to use the built-in function **expand.grid** to produce the values for the two explanatory variables. If you intend to produce a **wireframe** (or **persp**ective) plot, then a grid interpolated between 10 values along each of the axes produces a good-looking surface (neither too course not too fine). The function **expand.grid** then calculates a matrix containing 100 points (10 by 10) at which **predict** can calculate the fitted values.

Our example concerns recruitment of mature fish to a range of stocks differing in population *Density* and in the number of boats comprising the *Fisher*y. The response variable is the number of *Recruits*. Preliminary plots suggest that the regression surface is certainly curved, and may be quite complex. We produce a trellis of plots of *Recruits* against *Density* for different levels of *Fishery* using **coplot**, like this:

```
attach(fishing)
names(fishing)
```

```
[1] "Density"  "Fishery"  "Recruits"
```

```
coplot(Recruits~Density | Fishery,
                panel=function(x,y) { points(x,y); lines(lowess(x,y))})
```



The shape of the relationship changes considerably as the size of the fishery increases (from bottom left to top right as indicated in the uppermost box). Rather than speculate

about particular parametric forms for these relationships, this is a good candidate for beginning with a **gam** , where non-parametric smoothers are used to work out the ideal shape of the fitted curves

```
model<-gam(Recruits~s(Density)+s(Fishery))
```

To inspect the 3-D surface produced by this model we begin by creating a suitable grid of values for the two explanatory variables. A good rule of thumb is to create about 10 different values on each axis (it is a trade-off between speed of computation and roughness of the fitted surface). We need to know the maximum and minimum values of both explanatory variables. We can read these off the **coplot** produced earlier. *Density* should go from 0 to 100 and *Fishery* should go from 0 to 2. Sensible increments would therefore be 10 for the *Density* axis and 0.2 for the *Fishery* axis. The data frame for producing the predicted values is created by the **expand.grid** function like this:

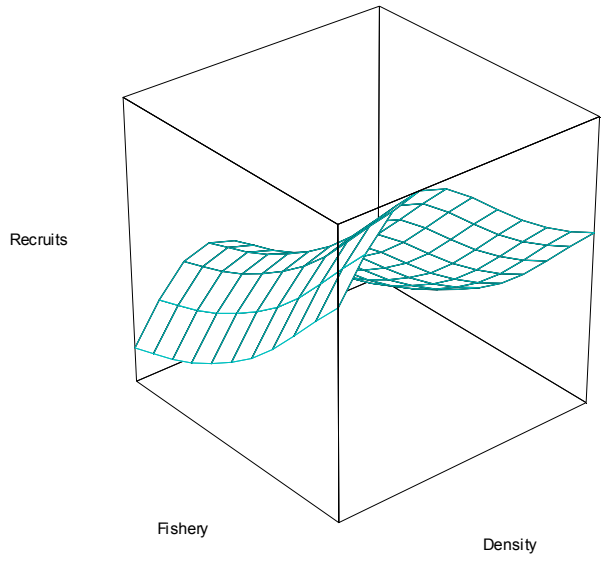
```
xyz<-expand.grid(Density=seq(0,100,10),Fishery=seq(0,2,.2))
```

The object *xyz* created by **expand.grid** is automatically defined as being a data frame. Now we can calculate the fitted values needed to plot the response surface. Use **predict** with the name of the current model, followed by the name of the data frame containing the values of the two explanatory variables that we want to use to generate the surface (*xyz*):

```
xyz$Recruits<-predict(model,xyz)
```

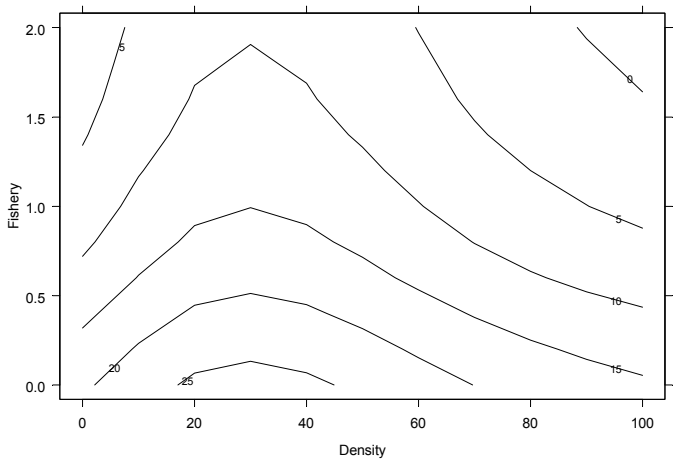
There in an important feature to note here. The predicted values are generated by applying the model formula to the values of the 2 explanatory variables in the data frame called *xyz* to create a new vector called *Recruits*. The generic operator **$** is then used on the left of the assignment statement to add this new variable to the data frame. The hard work is now done. We produce the 3-D plot using **wireframe** like this:

```
wireframe(Recruits~Density*Fishery,xyz)
```



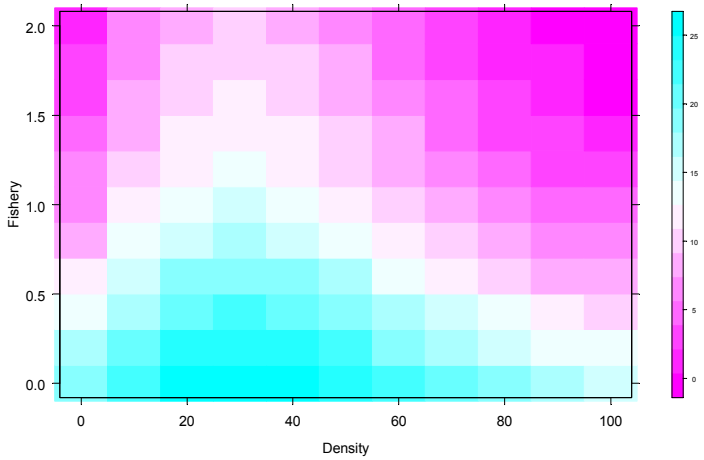
The same data frame could be used to produce other kinds of graphic output. For a contour plot, for instance, with the labels showing recruitment, we would type:

```
contourplot(Recruits~Density*Fishery,xyz)
```



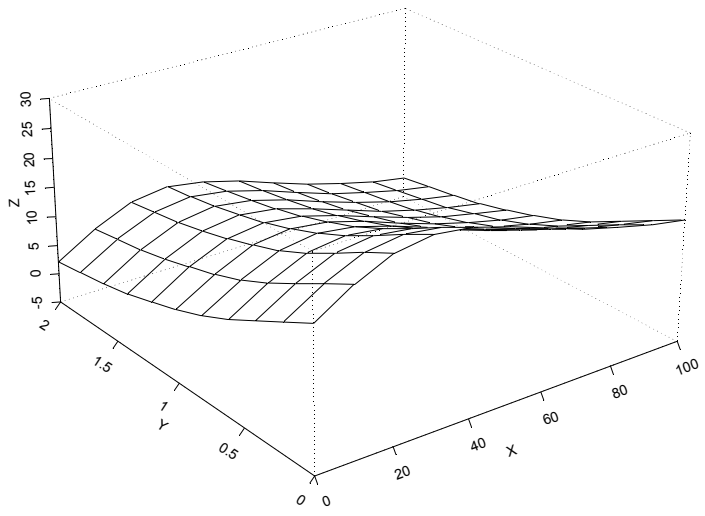
Or to produce an intensity plot (although our chosen grid of 10 by 10 is rather course for this to be particularly effective):

```
levelplot(Recruits~Density*Fishery,xyz)
```



The old-fashioned **persp**ective plot can be used, but this automatically does the interpolation we have already carried out with **expand grid**. Its arguments are the discrete levels for each axis (which we can provide as **seq**uences), then the longer vector containing all of the values of the response variable:

```
persp(seq(0,100,10),seq(0,2,0.2),xyz$Recruits)
```



Further reading

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models*. Chapman and Hall, London.