Lecture 17. Generalized additive models

Now that we've spent some time thinking about model selection, we'll get back to learning new kinds of modeling techniques. Next up is generalized additive models (GAMs). GAMs are useful for modeling nonlinear relationships. We've already considered how to model nonlinear relationships using nls(), so how are additive models different? The difference is that with nonlinear least squares (or an equivalent method for non-normal distributions), we have a particular nonlinear curve that we fit to the data. With additive models we can model nonlinear relationships without specifying a formula for that relationship. Instead we say "let's assume that *Y* is some smooth function of *X*; based on the data, what does that function look like?". Because we don't specify a nonlinear function in advance, these methods are called *non-parametric*.

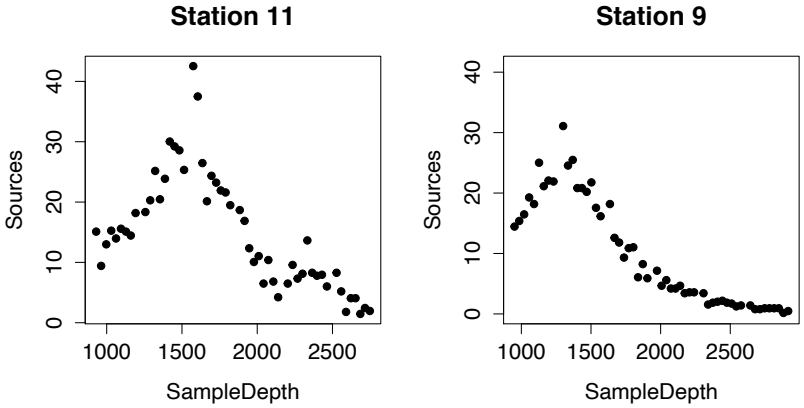
There are pros and cons when using GAMs. The pros are that we can ask questions like:

- How does *Y* depend on *X*, without assuming any particular curve?
- Does the relationship between *Y* and *X* differ between groups of data (different sites, different years, etc)?
- Even if I don't really care about the effect of *X*, I need to control for it in this analysis, so what is the best relationship to do that?

The cons are:

- We don't get an interpretable equation for the relationship between *Y* and *X*, which means we have to interpret it by just plotting it.
- Making predictions is harder, though not impossible.
- Combining additive models with other kinds of complexity (mixed models, correlated residuals) is tricky, though not impossible.

Let's look at an example to make this more concrete. I will use some data from this study (http://www.int-res.com /abstracts/meps/v341/p37-44/), where the prevalence of bioluminescence was measured as a function of depth at a number of stations in the northeast Atlantic. The data for two of the stations look like this:



The y-axis 'Sources' is the estimated number of sources of bioluminescence per cubic meter. Clearly bioluminescence has a nonlinear relationship with depth at these sites. The relationships are pretty smooth, perhaps smooth enough that there isn't really a need to fit a statistical model. However we might want to ask whether the nonlinear relationship differs between these stations, and that is not clear from a visual comparison, but we can answer it with a GAM.

I've been writing the mathematical form for a GLM like this:

$$\text{link}(\mu_i) = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \ldots$$
$$Y_i \sim \text{Probability\_Distribution}(\mu_i)$$

Where 'link' is the link function that relates the linear predictors to the scale of the data, which is drawn from some probability distribution. We can write the form of a GAM in a similar way:

$$\text{link}(\mu_i) = \beta_0 + f(X_i) + \ldots$$
$$Y_i \sim \text{Probability\_Distribution}(\mu_i)$$

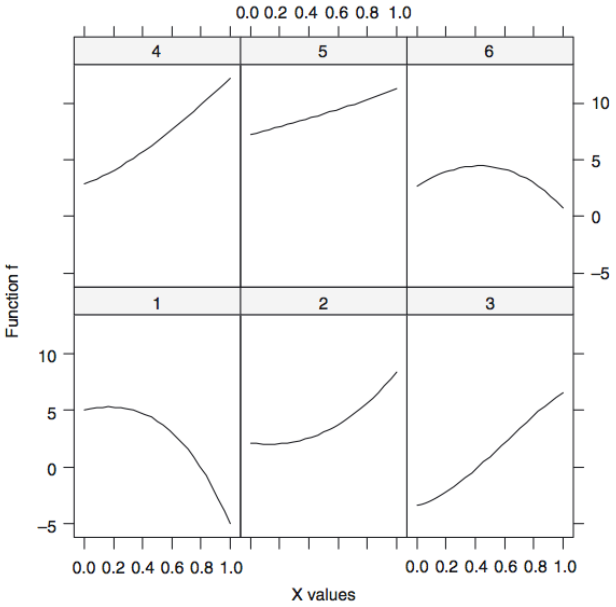
Here $f(X_i)$ is some smooth function of X. The model can contain more than one smooth function, other linear predictors like a standard GLM, as well as interactions that allow the smooths to vary by other predictors, as we will see. If more than one smooth function (for different predictors) is included, the effect of those functions is just added together, hence the name "additive models". Although GAMs model nonlinear relationships, the smoothers are actually comprised of linear components (as we will see), which allows these models to be fit using similar machinery as GLMs.

Cubic regression splines

The challenge for a GAM is to find a smooth function that approximates the pattern in the data, and we want it to be the 'best' approximation in some sense. In

practice, there are two choices to make: 1) which kind of smoother to use, 2) how 'wiggly' the smoother should be. There are many different kinds of smoother, too many to cover here, but fortunately the results are usually very similar regardless of the smoother used. I'll cover two common kinds of smoother.

The first smoother I'll describe is called cubic regression splines. The basic idea is to paste together a bunch of cubic polynomials. A cubic polynomial is a curve with this kind of formula: $Y = a + b*X + c*X^2 + d*X^3$. Cubic polynomials can take on a variety of shapes, here are some examples:



To use this kind of curve to fit an arbitrary nonlinear relationship, we can take several polynomials and paste them together. If we just chopped the data into a number of pieces, e.g. 4 pieces, and fit a cubic polynomial to each piece, it would look something like this:

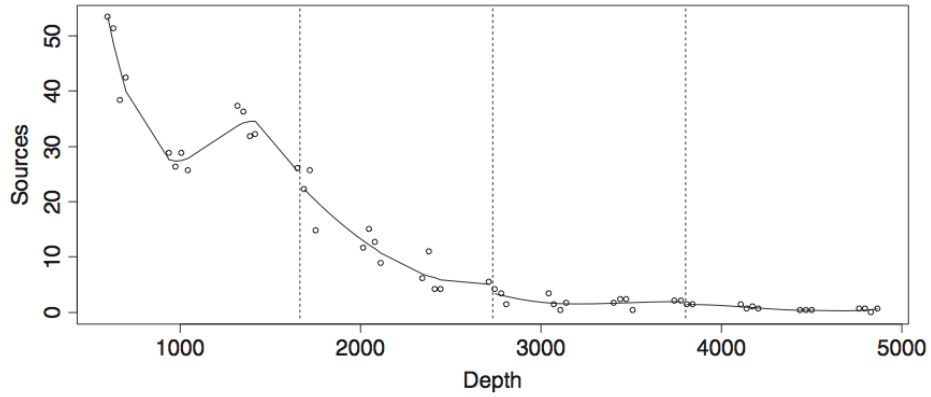


Fig. 3.8 Illustration of fitting a cubic polynomial on four segments of data using the ISIT data from station 19. We arbitrarily choose four segments along the depth gradient. The dotted lines mark these segments, and the line in each segment is the fit from the cubic polynomial model. R code to create this graph can be found on the book website

This looks like a pretty good approximation of the data, but it is not smooth, due to the breaks between the curves. The idea of cubic regression splines is to find a set of cubic polynomials that are constrained to be smooth at the breaks, which are called 'knots'. Here 'smooth' has a mathematical meaning: the curves that meet at the knots must have the same first and second derivatives. Here is an example of a cubic spline with 7 knots, with extra illustration to show how the curve is smooth:

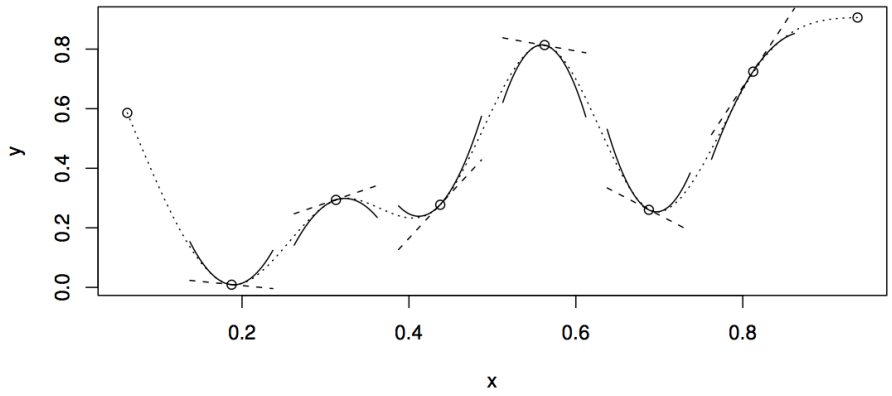
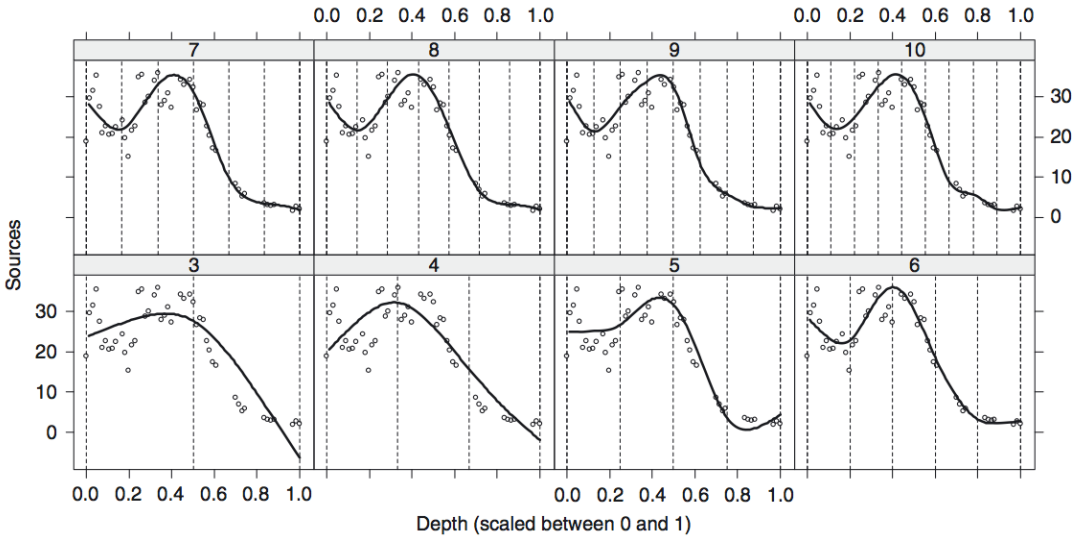


Figure 3.3 *A cubic spline is a curve constructed from sections of cubic polynomial joined together so that the curve is continuous up to second derivative. The spline shown (dotted curve) is made up of 7 sections of cubic. The points at which they are joined (○) (and the two end points) are known as the* knots *of the spline. Each section of cubic has different coefficients, but at the knots it will match its neighbouring sections in value and first two derivatives. Straight dashed lines show the gradients of the spline at the knots and the curved continuous lines are quadratics matching the first and second derivatives at the knots: these illustrate the continuity of first and second derivatives across the knots. This spline has zero second derivatives at the end knots: a* 'natural *spline'.*

The dotted curve is the smooth spline. The circles show the knots where cubic polynomials are pasted together. The dashed tangent lines illustrate that the curves that are pasted together have the same first derivative (recall that the first derivative of a curve is the slope of the tangent line). The solid quadratic curves illustrate that the curves that are pasted together have the same second derivative (the second derivative of a curve quantifies how 'curvy' the curve is at that point).

If we want to use this kind of smoother on some data, the result will depend on how many knots we use. Here is an example of fitting cubic regression splines using different numbers of knots:



The top row uses 7-10 knots, and the bottom row uses 3-6 knots. As the number of knots increases, the curve becomes wigglier, because it is now more flexible in fitting the data. But eventually the shape of the curve converges, at around 5 or 6 knots. The wiggliness of a smoother is important, because it determines *the tradeoff between underfitting and overfitting*, which we've discussed several times already. If the smoother is not wiggly enough, then it will miss real patterns in the data. If the smoother is too wiggly, then it will overfit the data, i.e. it will fit subtle random patterns that will reduce the predictive performance of the curve. Therefore, the key challenge for GAMs is to find a smooth curve that is optimal in terms of the balance between underfitting and overfitting.

One way we could control the wiggliness of a cubic regression spline is vary the number of knots, i.e. the number of polynomials that we paste together, and compare models with different numbers of knots using AIC or null hypothesis tests. This is certainly possible, but theoreticians have shown that there is a better approach. Instead of varying the number of knots, we can use a reasonably large number of knots, and *control the wiggliness of the curve by controlling the parameters of the spline*. These are called penalized splines.
How does this work? I'm not going to get into the math that underlies fitting GAMs, because it is too much detail for this course. But in conceptual terms, it is not too hard to understand:

- A cubic polynomial is a kind of linear model, because the parameters for $Y = a + b*X + c*X^2 + d*X^3$ are just multipliers of X, X^2, X^3
- A cubic regression spline is a bunch of cubic polynomials pasted together, which means it is just a very complicated linear model
- To find the spline that is the best fit to the data, we just need to find the maximum likelihood estimates for the spline parameters, using similar algorithms as for GLMs

This clarifies how we could imagine fitting a spline, if we specify the number/location of the knots. To understand penalized splines, we just need to modify the maximum likelihood function. For a gaussian model, e.g. a standard linear regression, we find the parameter estimates by minimizing this:

$$\sum (Y_i - \mu_i)^2$$

I.e., we minimize the sum of squares, which are the squared differences between the observed values and the predicted values. As we've discussed, minimizing the sums of squares is equivalent to maximizing the likelihood, for a gaussian model. To fit a penalized regression spline, we will minimize this instead:

$$\sum (Y_i - \mu_i)^2 + \lambda \int f''(x)^2 dx$$

Here *f(x)* is the smoother, and *f''(x)* is the second derivative of the smoother. The integral of the second derivative, $\int f''(x)^2 dx$, is quantifying how wiggly the smoother is, on average. The constant λ determines the penalty for wiggliness. If λ = 0, then there is no penalty for wiggliness, and smoother can be as flexible as possible in fitting the data. As λ increases, then a wigglier curve will fit the data better (reducing the sums of squares) but will also receive a higher penality (the second term in the formula). Therefore, by minimizing this criterion, the fitting algorithm will fit the smoother to the data while also determining the appropriate amount of wiggliness for the smoother. Minimizing this criterion can be thought of as *maximizing a penalized likelihood*.

Here's an example of a penalized regression spline fit to some data on engine wear, using different values for λ.

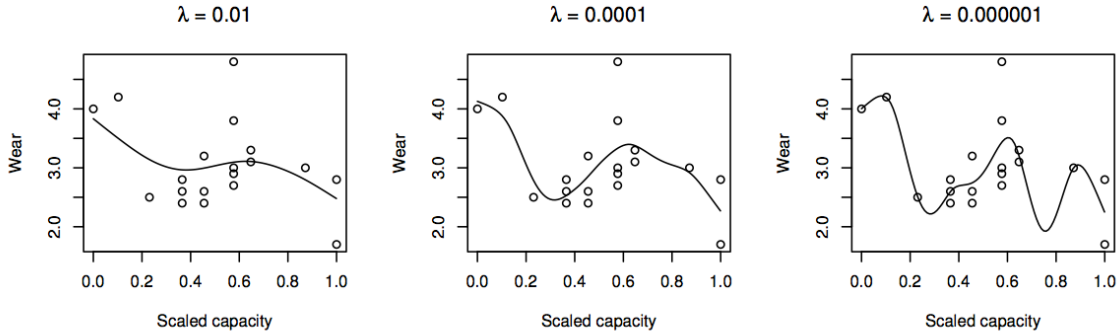


Figure 3.8 *Penalized regression spline fits to the engine wear versus capacity data using three different values for the smoothing parameter.*

The final question is, what is the right value of λ to use? From the above plot you can see that the curve on the right is probably overfit to the data, while the curve on the left may be underfit. Fortunately we already know how to quantify the

predictive performance of a model, which is to use cross-validation. That is essentially how λ is chosen in GAMs. There are various metrics of cross-validation performance that are used by the mgcv package, which I will not explain in depth. For the engine wear data, we can look at how one of these, the *generalized cross-validation* (GCV) changes as we change λ for the smoother:

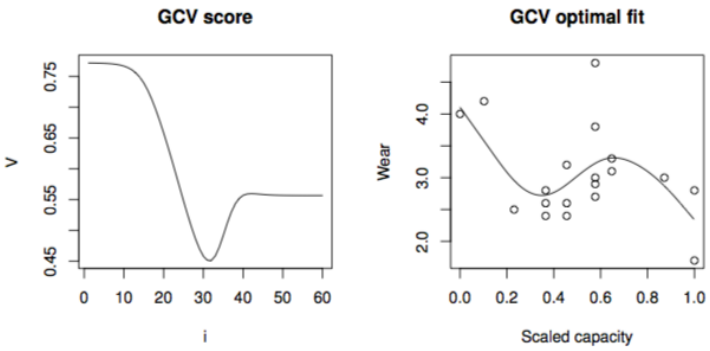


Figure 3.10 *Left panel: the GCV function for the engine wear example. The smoothing parameters are shown on a log scale on the x axis, such that* $\lambda = 1.5^i \times 10^{-8}$. *Right panel: the fitted model which minimizes the GCV score.*

A lower GCV score is better, because it means less predictive error. The plot on the right shows the smoother fit using the values of λ that yields the smallest GCV.

I've introduced GAMs using penalized cubic regression splines as the example smoother, because it is (relatively) easy to visualized how they work. However, you should note that there are many other kinds of smoother, and the default smoother in the software we will used is not cubic regression splines, but is another kind of smoother called *thin plate regression splines*. These are similar in the sense that many component curves are added together to make an arbitrary smooth curve. These are considered a bit better than the alternatives, but they don't used fixed knots, and so it is a bit harder to visualized how they work.

I've skipped nearly all the mathematical detail underlying GAMs, because it is difficult and would take a lot of time to understand. Using them in practice is actually fairly easy, with the caveat that there is a 'black box' feel due to the many automated algorithms that are fitting the smoother. So, caution is in order. We'll do some examples now, but first let's review:

- We want an arbitrary smooth curve to relate *Y* to *X*
- We can construct such a curve by adding together many simpler curves
- To control the underfitting-overfitting tradeoff, we allow the smoother to have a large number of components, but we control the wiggliness of the curve using a penalty term
- The optimal penalty term is chosen as part of the fitting process, via generalized cross-validation

GAM with mgcv

Let's go back to the example of bioluminescence vs. depth, to see how to use GAMs in R. The package we will use is called 'mgcv'. Specification is the same as for GLM, except now we specify a smoother with s():

```
mod = gam(log(Sources) ~ s(SampleDepth), data = lumsub)
```

This syntax tells the function to fit a smoother to the predictor SampleDepth. I've decided to model Sources as a lognormal variable, after an initial look at the data (it's heteroscedastic if we use a normal distributinon). The smoother is chosen by default (thin plate regression spline is the default). I haven't specified a family() argument, because I'm starting off by assuming that a lognormal distribution is appropriate, but specification of non-normal responses is the same as for glm(). We can inspect the model with summary()

```
summary(mod)
```

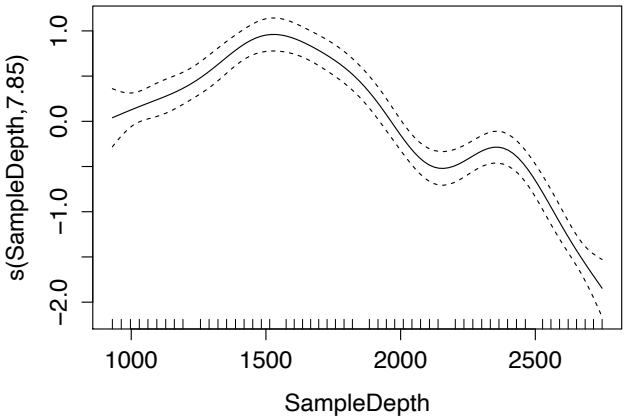
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Sources) ~ s(SampleDepth)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.4570     0.0334    73.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df    F p-value
## s(SampleDepth) 7.85   8.66 56.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.904   Deviance explained = 91.8%
## GCV = 0.070849  Scale est. = 0.059024  n = 53
```

The output will have two sections: any parametric coefficients (i.e. continuous predictors that do not have smoothers, or factors that model the mean level of the response), and the smooth terms. In this case the only parametric coefficient is the intercept. The smooth term reports 'edf', which is the *estimated degrees of freedom*. This number is using the wiggliness of the smooth to say "if this smoother were written as a parametric equation, approximately how many parameters would that equation have". This number is useful for things like AIC and null hypothesis tests. The F-statistic, and associated p-value, come from an approximate F-test for how (marginal) variation is explained by the smoother. Clearly significant variation is explained by the smoother. In general one should be a little wary of the p-values

returned from this package, because the tests do not account for uncertainty in lambda, the smoother penalty. This tends to make the p-values a little smaller than they should be. We also get output for the deviance explained, the generalized cross-validation score, etc.

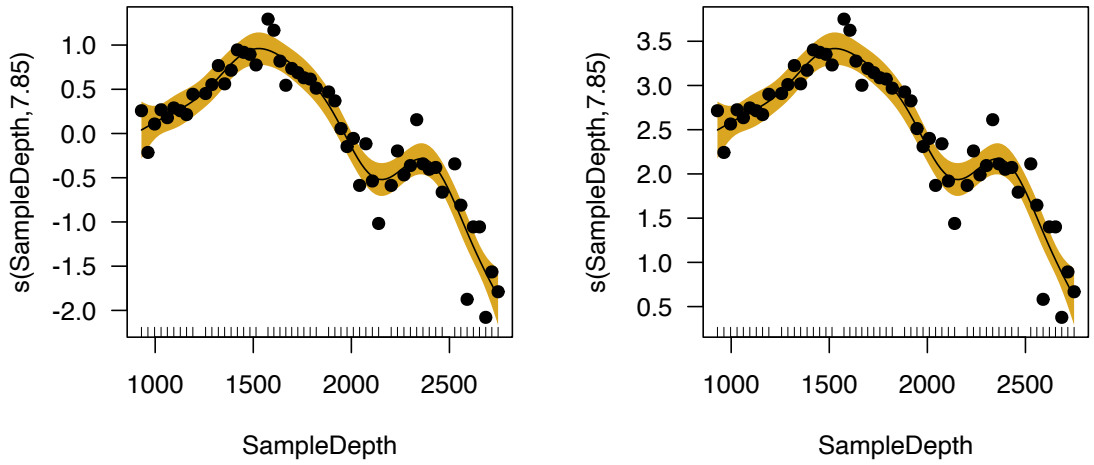
The functions for plotting the fitted response and residuals are a little different from what we've used for lm() and glm(). We can visualize the fitted smoother with the plot() function (look at the help file for plot.gam() for details):

```
plot(mod)
```



The solid line is the fitted smoother, and the dotted lines are +/- 2 standard errors around the smoother, which is an approximate confidence interval. The y-axis is labeled s(SampleDepth,7.85) because it is plotting the fitted smoother, which has estimated degrees of freedom 7.85 (more on that soon). The y-axis is not the same as the raw data, because the smoother does not include the intercept. For this simple case, with an intercept and a single smoother, the intercept will be the mean of the data, and the smoother will be the variation around that mean. We can plot *partial residuals* on top of this smoother plot, and also shade the confidence ribbon:

```
plot(mod, residuals = TRUE, pch = 19, shade = TRUE, shade.col =
'goldenrod')
```



The plot on the left shows the same smoother fit, plus partial residuals. What are partial residuals? They are defined like this:

$$\varepsilon_i^p = f(X_i) + \varepsilon_i$$

In other words, the partial residual ε_i^p for observation *i* is the predicted value from the smoother, $f(X_i)$, plus the residual ε_i. Partial residuals are useful for asking "what is the effect of predictor *X*, while accounting for any variation explained by the other predictors in the model?". They are not only used for GAMs; you can use them for any regression-type model, e.g. for GLMs you can plot them using plot(allEffects(model, partial.residuals = TRUE)).

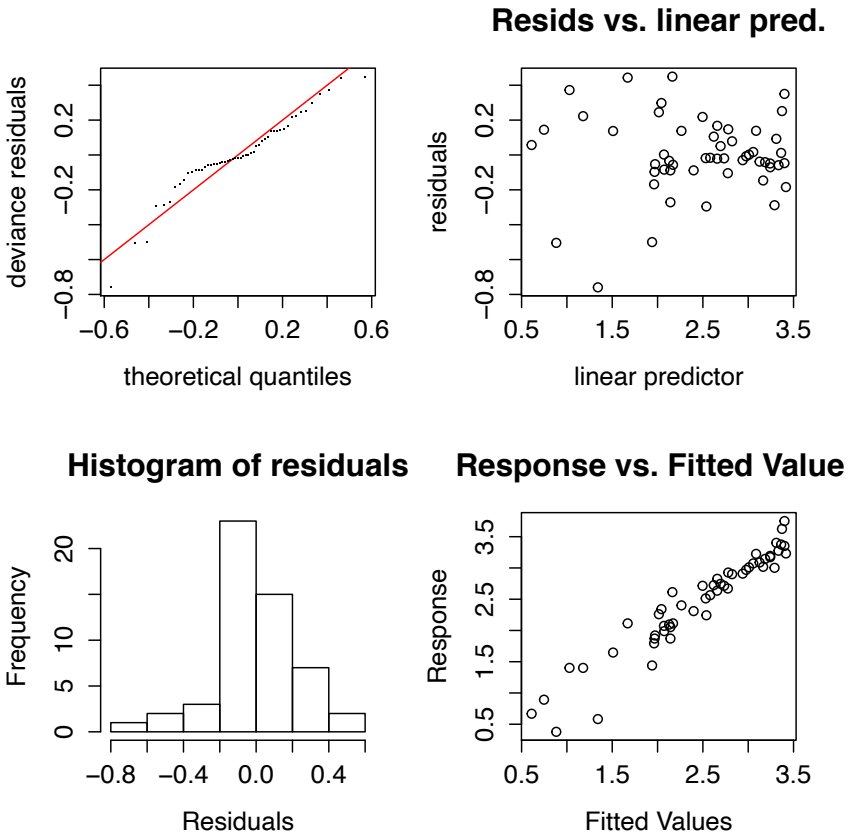
As a final step for plotting, if we want the fitted relationship on the scale of the raw data, we can add the intercept back into the plot, by saying shift = coef(mod)[1]:

```
plot(mod, shift = coef(mod)[1], shade = TRUE, shade.col = 'goldenrod',
residuals = TRUE, pch = 19)
```

That produces the plot on the right above.

If we want to do residual diagnostics, there is a nice function gam.check():

```
gam.check(mod)
```



The plots in the left column give a sense for whether the deviance residuals are normally distributed (of course this might not be that useful for non-normal responses), and the plot on the right column give a sense for whether the variance is homoscedastic. There is a bit of weirdness at low fitted values, but overall it looks OK.

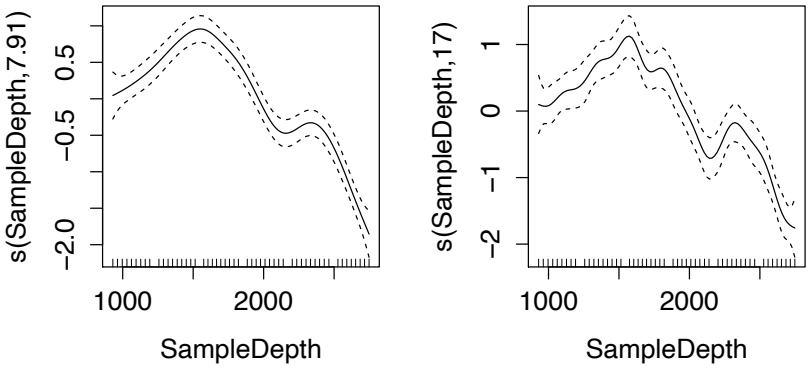
Now we've seen the basics for fitting and interpreting a GAM, let's mess around a bit with the algorithm. We've let the function automate everything involved in fitting the smoother. It automatically decides the dimension of the smoother (e.g., how many knots for a cubic regression spline), and then it choose the optimal penalty for wiggliness using GCV. Usually the default settings work quite well, but let's explore what happens when we change some settings. When creating the smoother, the algorithm decides what the maximum complexity of the smoother should be, and then the actual complexity of the smoother will be reduced to some degree by the penalty term. We get some info on this from gam.check():

```
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 6 iterations.
## The RMS GCV score gradiant at convergence was 1.326e-06 .
## The Hessian was positive definite.
## The estimated model rank was 10 (maximum possible: 10)
```

```
## Model rank =  10 / 10
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                   k'  edf k-index p-value
## s(SampleDepth) 9.00 7.85    1.25    0.96
```

I won't decipher all of this text, but at the bottom it tells us k', which is the *basis dimension*. The meaning of the basis dimension depends on the kind of smoother used, but it effectively determines the upper bound on the complexity/wiggliness of the smoother. In this case the model set k' = 9, and then during the optimization process the estimated degrees of freedom was reduced to 7.85. In some cases we might think that the model did not permit enough complexity to begin with, so we can set k' ourself, like this:

```
mod = gam(log(Sources) ~ s(SampleDepth, k = 18), data = lumsub)
plot(mod)
```

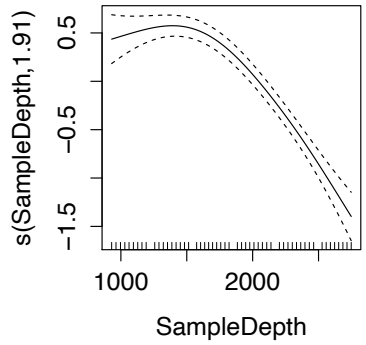


Now I've set the basis dimension to 18, which is twice as large as the default setting, and it produced the smoother on the left. This looks essentially identical to what we found with the default settings, which is good. It means that the default settings allowed for enough complexity according to the cross-validation criterion. We can also see what a basis dimension of 18 would look like *if the smoother was not penalized for wiggliness*. We can do this by setting fx = TRUE:

```
mod = gam(log(Sources) ~ s(SampleDepth, k = 18, fx = T), data = lumsub)
plot(mod)
```

This results on the curve on the right, which has a lot more small scale ups and downs. According to the cross-validation criterion, this curve overfits the data, and so a smoother curve results when we let the model penalize the smoother. We can also increase the penalty on the smoother, with the option 'gamma':

```
mod = gam(log(Sources) ~ s(SampleDepth), data = lumsub, gamma = 8)
plot(mod)
```

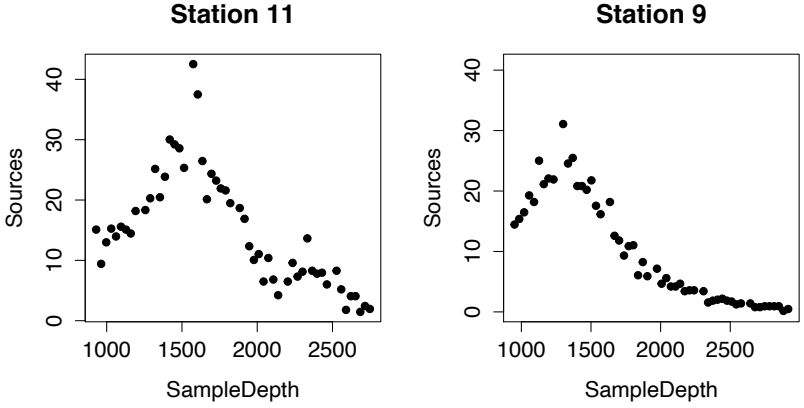


:

The default is gamma = 1; by setting gamma = 8, I've multiplied the wiggliness penalty by 8, which results in a less curvy smoother being fit to the data. It still has the same overall shape, but is less wiggly.

Interaction between a smoother and a factor

The example I've been using so far is fairly dull. The data has a very smooth relationship, and there are no predictors other than depth, so it hardly seems necessary to fit any model. It gets more interesting if we want to compare two smooths, e.g fit to two different stations:



These two stations have a similar bioluminescence profile, but it seems like Station 9 has a peak at a shallower depth than Station 11. One of the nice things about GAMs is that we can ask "does the relationship between Y and X differ between these groups?", without specifying any particular relationship.

To allow a smoother to vary by factor levels, we can use the 'by' argument in the s() function. There are two ways to do this, and both can be useful. To fit a separate smoother for each station, we can do:

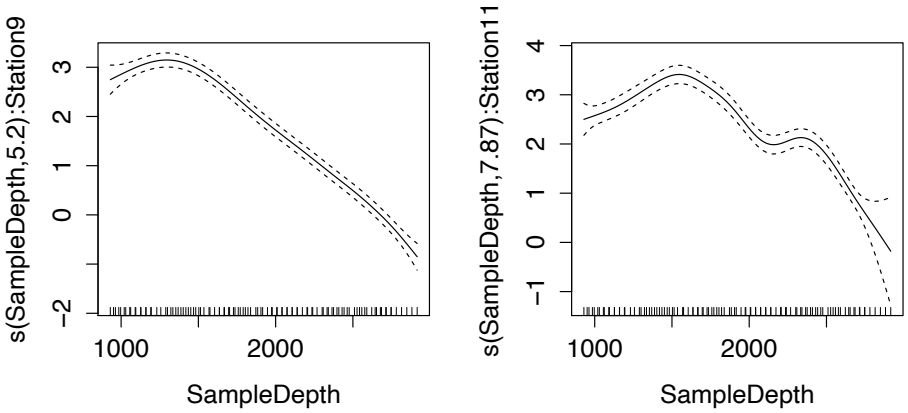
```
mod1 = gam(log(Sources) ~ Station + s(SampleDepth, by = Station, k = 18), data
 = lumsub)
summary(mod1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Sources) ~ Station + s(SampleDepth, by = Station, k = 18)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7930     0.0338    53.1   <2e-16 ***
## Station11     0.5590     0.0509    11.0   <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(SampleDepth):Station9  5.20   6.47 218.2  <2e-16 ***
## s(SampleDepth):Station11 7.87   9.67  49.2  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.953   Deviance explained = 95.9%
## GCV = 0.070368  Scale est. = 0.060456  n = 107
```

Now we have a model with two smoothers, one for each station. Note that I also included a parametric term for Station. This is important because the smoothers are centered around zero, and so the parametric factor quantifies whether the stations differ in their mean bioluminescence. The smoothers will therefore quantify whether the stations differ in the shape of the depth profile, after any difference in mean bioluminescence has been accounted for. The parametric coefficients indicate that the mean bioluminescence at station 11 is greater by a factor of $\exp(0.56) = 1.75$.

The two smoothers are each highly significant, which is not surprising. Let's plot what they look like:

```
par(mfrow = c(1,2))
plot(mod1, select = 1, shift = coef(mod1)[1])
plot(mod1, select = 2, shift = coef(mod1)[1] + coef(mod1)[2])
```

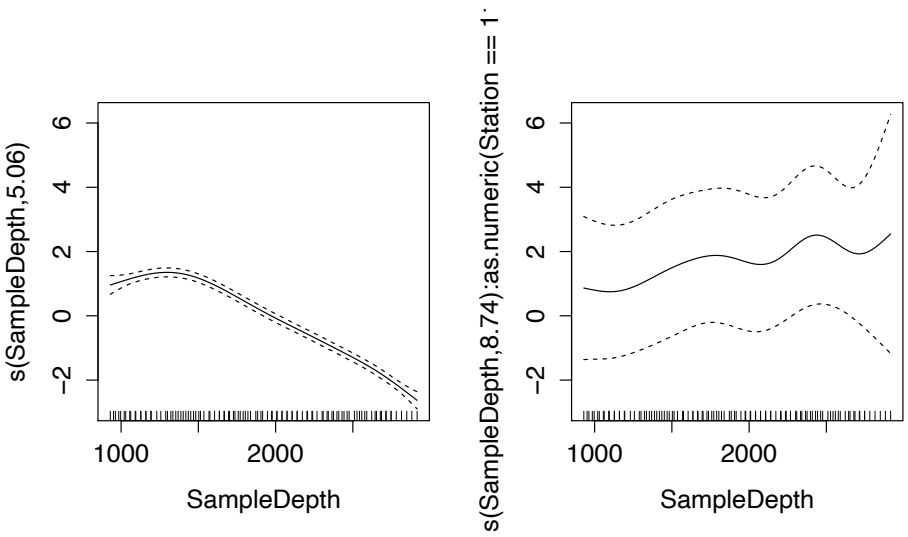


Here I've added in the intercept for each smoother as well, which is not necessary. We can see that station 11 may have a peak that is shift to the left, and may have more bioluminescence at greater depth. However, it is hard to make this comparison by looking at the two smoothers.

An alternative way to fit this interaction is to fit (1) a smoother for the depth profile at station 9, and (2) a smoother that quantifies the difference between station 11 and station 9. This is like the way that interactions are parameterized in the GLMs you've been using. To do this, we can fit this model:

```
mod2 = gam(log(Sources) ~ Station + s(SampleDepth) + s(SampleDepth, by
= as.numeric(Station == 11)), data = lumsub)
```

The first smoother just says 'what is the relationship with depth, regardless of station'. The second smoother uses a different 'by' argument, `by = as.numeric(Station == 11))`. Now we are giving by a vector of 0's and 1's; this vector will be zero for Station 9 and one for Station 11. When the s() function is given a numeric variable, it creates an interaction by multiplying the smoother by that numeric variable. In this case it will multiple the smoother by 1 whenever Station == 1; the result is that this smoother quantifies how station 11 differs from the overall relationship fit by the first smoother. Let's plot the results:



The first smoother is similar to what we found for station 9 before, which is good. The second smoother is quantifying how station 11 differs from station 9. It looks like at shallower depths (around 1000 meters) the two curves are the same, but as the depth increases there are more bioluminescent sources at station 11. The confidence intervals on this effect are large, but the overall difference in shape between the sites is significant:

```
summary(mod2)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Sources) ~ Station + s(SampleDepth) + s(SampleDepth, by = as.numeric(St
ation ==
##     11))
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7930     0.0338   53.01   <2e-16 ***
## Station11    -1.0805     1.0440   -1.03      0.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                           edf Ref.df     F p-value
## s(SampleDepth)                           5.06   6.13 228.9  <2e-16 ***
## s(SampleDepth):as.numeric(Station == 11) 8.74   9.26  16.9  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Rank: 20/21
## R-sq.(adj) =  0.953   Deviance explained = 95.9%
## GCV = 0.070766  Scale est. = 0.060653  n = 107
```

We could also compare this model to a model with no depth*station interaction, using AIC:

```
mod0 = gam(log(Sources) ~ Station + s(SampleDepth), data = lumsub)
```

```
library(MuMIn)
AICc(mod0)

## [1] 98.45

AICc(mod2)

## [1] 28.14
```

But wait a minute, how do we count the number of parameters for a GAM? AIC is logLik(model) + 2*(number of parameters), but for GAMs counting parameters is tricky. The basis dimension of the smoother(s) determines the total number of parameters in the underlying model, but due to the penalty on smoother curviness the fitted model does not truly have as much flexibility as the basis dimension. The solution is to use the effective degrees of freedom (edf) that is estimated for each smoother, as an approximation of how many parameters that smoother represents. This is used for calculating AIC, as you can see if you do ?AIC.gam.

Looking at the AIC results above, clearly the model with the interaction has much more support. From this we can conclude that the shape of the depth profile differs between stations, and we can use the plot on the right above to see that bioluminescence tends to be more common at greater depths at station 11.