What's up with `geom_smooth` and additive models?

immediate

May 1, 2023

1 Introduction

If you have ever wondered what `geom_smooth`[1] actually does or been baffled by *generalised additive models* (GAMs) then this blog post is for you. It is not a comprehensive introduction but intended as an intuition-building stepping-stone into the more advanced literature. Michael Clark has written an elegant and more in-depth introductory blog[2] while Simon Wood has written the bible (Wood, 2017). Be warned, like the new testament the book of Wood is mostly in Greek...

The blog first introduces and compares GAMs to its main competitor, polynomial regression. From there we proceed with some concrete examples where we manually replicate the functionality of `geom_smooth` and **mgcv**. Lastly, we cover the central topic of penalised estimation.

2 What are generalised additive models?

Generalised additive models (GAMs) are statistical models specialized in detecting highly non-linear trends in data. In this regard, they are similar to polynomial regression which can be regarded as their main "competitor".

Both polynomials and GAMs perform a *basis expansion* to detect non-linear patterns. The basis is our measured variable x which we expand with additional predictors to enable a non-linear fit. Formally, polynomial regression and additive models share this basic setup.

$$
\begin{aligned}
y &\sim \mathcal{N}(\mu(x), \sigma), \\
\mu(x) &= \sum_{i=1}^{k} \beta_i f_i(x).
\end{aligned}
\tag{1}
$$

[1]A plotting function from `ggplot2` in R (?R Core Team, 2020).

[2]https://m-clark.github.io/generalized-additive-models/

y is our outcome variable which we assume to be drawn from a normal distribution with a mean μ and standard deviation σ. We model μ as a sum over functions $f(x)$. This is the *basis expansion.* The parameter k controls the extend of the basis expansion and β_i is a vector of associated coefficients. By setting $f(x) = x^i$ we have polynomial regression. In this cases k controls which polynomial order we are estimating. We later see how $f_i(x)$ works differently for GAMs.

A central difference between GAMs and polynomials is that polynomials do *global* estimation. Essentially this means that, for instance, the data points at $x < 2$ will affect the fit at $x > 10$. This global property is easy to see for linear models. Like a seesaw, we cannot change the slope at $x < 2$ without changing it at $x > 10$. The same holds for polynomial regression. GAMs work locally and are more flexible. In the context of GAMs k controls how many bins we want to carve the predictor variable up into. In GAM vocabulary the points separating each bin are called *knots* and k is then the number of knots. GAMs fit a model to each bin - hence the local fit. These local models are then added together to obtain a single statistical model. Let us illustrate what this means.

All code for this example can be found at XXX. For our simulation we first need some data. In this simple example we consider an outcome y and predictor x.

```
n <- 200
x <- runif(n)
true <- 4*x + 3*x^2 - 3*x^3 - 6*x^4
y <- rnorm(n, true, 1.5)
d <- data.frame(x,y,true)
```

Figure 1 shows the true non-linear relation (yellow) and a sample of 200 data points. The red line visualises the output of `geom_smooth` which recovers the true function well. Computationally `geom_smooth` is estimated via `mgcv::gam`[3] so our task is to understand `mgcv::gam` which we will do by manually re-creating its output. The default setting of `geom_smooth` is to use a cubic regression spline as its basis function and $k = 10$ knots. In other words, `geom_smooth` spreads 10 cubic functions, $f(x) = x + x^2 + x^3$, across our predictor variable. In Figure 2 we visualise this basis expansion. The knots are the points where one basis function equals 1 while the others equal zero. This setup makes GAMs localised as the fit of the orange colored

[3]With less than 1000 observations `geom_smooth` uses *loess* regression by default. I've manually over-written this so we use `mgcv::gam(x s̃(x, bs = "cs", k = 10)`

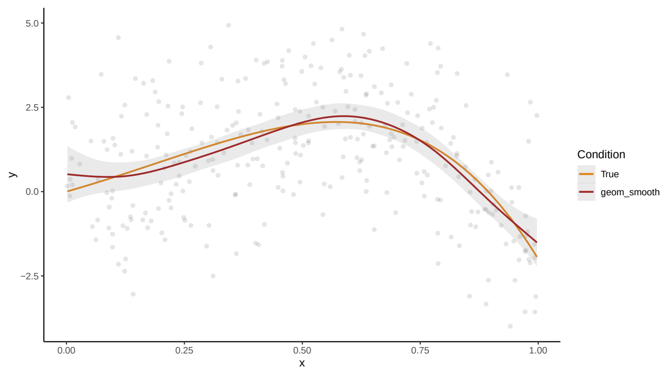

Figure 1: 200 data points (dots) simulated from a non-linear function (yellow line). The red line is the default output of `geom_smooth`. We see it recovers the relationship well.

basis functions will not depend on the data in area associated with purple basis functions.

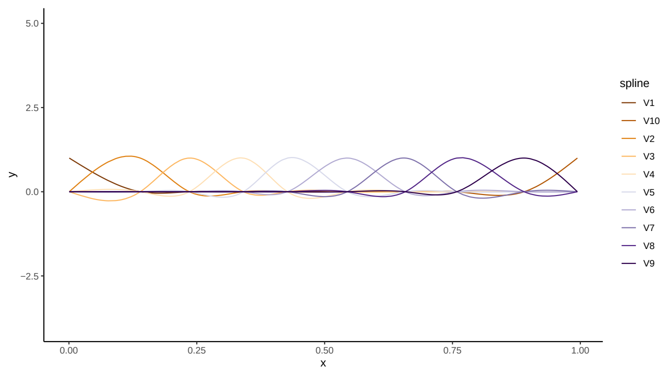

Figure 2: A 10 knot cubic regression spline basis expansion.

The next step is to evaluate each participant on each basis function $f_i(x)$. We can do this in R with `smoothCon` and obtain a n by k matrix (named `splines` in the code below).

```
splines <- smoothCon(s(x , #predictor
                  bs="cr", #cubic regression spline
                  k = 10), #10 knots
                data=d)[[1]]$X
```

The object `splines` can then be used with `lm` to estimate β.[4] Once we know β we can multiply each basis function with its coefficient and sum everything to achieve our prediction.

```
lmMod = lm(d$y ~ .-1, data= as.data.frame(splines))
#".-1" means use all variables in the data and omit adding intercept

bscoefs = coef(lmMod) #extracting coefficients

#multiplying basis functions with coefficients
bsScaled = sweep(splines, 2, bscoefs,'*') %>% as.data.frame()

#adding everything -> prediction
bsScaled$sum_spline  <- colSums(t(bsScaled))
```

Figure 3 shows how the coefficients modulate the basis functions and the resulting overall model in red. In Figure 4 I compare our manually fitted GAM to `geom_smooth`. As expected, their general pattern is similar, however, our model is much more "wiggly". This difference leads *penalised estimation* of GAMs.

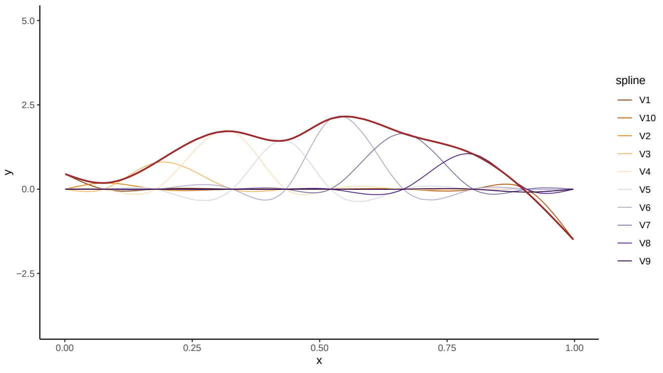

Figure 3: The 10 basis function multiplied by their coefficients obtained by fitting them to an outcome variable. The red line is the sum of the basis functions and represent the prediction of our model.

2.1 *Penalised* additive models

Since GAMs are highly flexible are susceptible to over-fitting, especially in cases with many knots and few data points. Consequently, selecting the right

[4]The code is from the technical appendix of Michael Clark's blog: `https://m-clark.github.io/generalized-additive-models/technical.html`

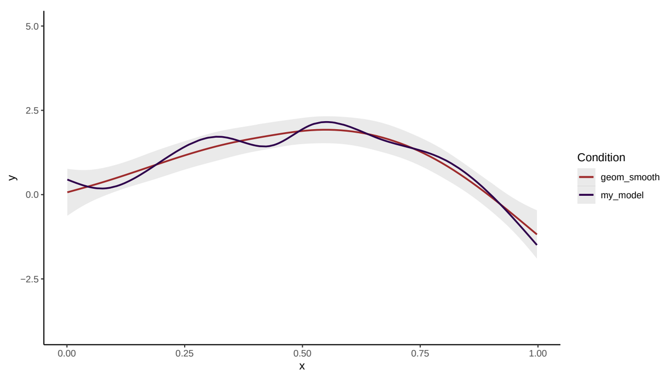

Figure 4: To evaluate our custom spline based model (red), I compare it to the result of the a `mgcv::gam()`.

number of knots and spacing them correctly might be a central challenge. Luckily it is not and GAMs are generally easy to use. The solution is to use plenty of knots and avoid over-fitting through *penalised estimation*. Below I illustrate how the penalisation works.

When we use `lm` to estimate our model it tries to find the model that has the highest likelihood. This objective pushes the model towards a very wiggly shape. With penalised estimation we add an opposing objective that favors a smoother/simpler model. We can express the two opposing objectives mathematically:

$$\underbrace{(y - g(x, \beta_i))^2}_{\text{Wiggle}} + \lambda \underbrace{\int g''(x)^2}_{\text{Don't wiggle}} \tag{2}$$

The first term expresses the goal of minimising the difference between our outcome y and our model $g(x)$. The second term is a measure of wiggliness. If $g(x)$ is a straight line then $g''(x) = 0$. Thus, if $g''(x) \neq 0$ then $g(x)$ has some curvature. By computing the area under the curvature we can quantify its amount. The square makes sure we get similar results for convex and concave curvature. By adding this constraint to the minimization problem we penalise complicated models and favor simpler models. The extend to which we penalise our model is determined by the smoothing parameter λ which controls the trade-off between the goals. High values of λ favors a more smooth/straight curve so less over-fitting but possible under-fitting.

It turns out there is a function that minimises Equation 2 and is continuous in its first derivative so we can compute $g''(x)$. This function is the cubic spline we and `geom_smooth` use.

The question is then how we can implement this insight to make our wiggly function operate more smoothly? Wood (2017) shows we can compute a penalty matrix S that allow us to penalise our model without having to manually compute second derivatives[5].

$$\int g''(x)^2 = \boldsymbol{\beta}^T \boldsymbol{S} \boldsymbol{\beta} \tag{3}$$

Conveniently, the function we used to compute our splines, `smoothCon`, returns $\boldsymbol{S}$. We use the penalty matrix $\boldsymbol{S}$ to manually penalise our wiggly model. We do this by appending rows to our data: zeros as outcomes and $\boldsymbol{S}\boldsymbol{\beta}$ as predictor values[6]. For an explanation of this trick see the appendix. Akin to adjusting λ we can control the penalty by varying the number of times we append our penalty rows to our data. In the code below we add penalty rows 200 times.

```
#Retrieving the penalty matrix S
S = smoothCon(s(x,
                bs="cr",
                k = 10),
              data=d)[[1]]$S[[1]]

lambda <- 200 #smoothing parameter
penalty_mx <- as.data.frame(splines) #new data frame
penalty_mx$y <- d$y #adding outcome variable

zeros<- rep(0, nrow(S)) #vector of zeros
penalty_mx <- as.matrix(penalty_mx)

#We add our penalty matrix S and zeros to our data frame
#This is done lambda times
for (i in 1:lambda){
    penalty_mx <- rbind(penalty_mx,cbind(S,zeros))
    #outcome var is last column in pen matrix!
}
#We re-estimate our GAM model with the penalty rows added
lmMod_pen = lm(y ~ -1 + V1+V2+V3+V4+V5+V6+V7+V8+V9+V10,
data = as.data.frame(penalty_mx))  # regression
bscoefs_pen = coef(lmMod_pen) #extracting coefficients
```

[5]Here I jump to matrix notation. Boldface Greek letters denote (eg. $\boldsymbol{\beta}$) vectors, while boldface roman letter denote matrices (eg.$\mathbf{S}$).

[6]I do not recall where I saw this trick, but it's not mine.

```
bsScaled_pen = sweep(splines, 2, bscoefs_pen,'*') %>%
               as.data.frame() #multiplying basis functions with coefficients
bsScaled_pen$sum_spline_pen <- colSums(t(bsScaled_pen))
```

We can now evaluate our penalised model which should give a similar results as `geom_smooth`. In Figure 5 we see our penalised models together with `geom_smooth` and the unpenalised version of our model. As expected, the green model (our penalised model) and the red model (`geom_smooth`) are almost indistinguishable.

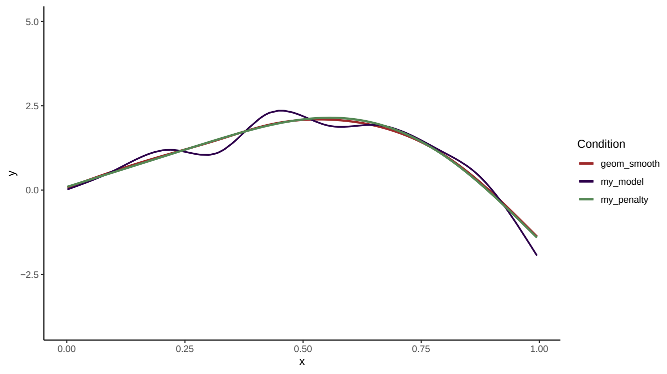

Figure 5: The 10 basis function multiplied by their coefficients obtained by fitting them to an outcome variable. The red line is the sum of the basis functions and represent the main prediction of our model

The final issue, which I won't go into details with, is how to choose the smoothing parameter λ. **mgcv** optimises a smart and fast cross-validation-criterion to identify the optimal λ. This means the user do not have to specify any hyper-parameters to estimate a GAM model. In my model, I tinkered around until I found the value (`lambda=200`) which yielded agreement with `geom_smooth`.

3 Conclusion and practical considerations

GAMs are extended linear models where we carve the predictor up into regions and fit separate models to each region before piecing it all together. This results in a flexible model that is effective at recovering non-linear trends. The penalties and cross-validation-criterion decrease the chances of over-fitting and make the model easy to use. In comparison with polynomial regression, I find GAMs simpler as you do not need to fit several models and do model comparisons. In terms of fit, I've generally found that GAMs and polynomials yield highly similar results. However, I've only studied simple cases with a few predictors and fairly simple non-linear relations.

4 Appendix

We can understand the trick of appending $\boldsymbol{S\beta}$ to our predictors in order to penalise our model as follows. First we consider the standard linear model. In the case of GAMs, x and z are two basis expansions.

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \beta_1 + \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_n \end{bmatrix} \beta_2 + \epsilon. \tag{4}$$

Equation 3 shows we can compute the wiggliness score by multiplying our coefficients with our penalty matrix. To illustrate we assume $\boldsymbol{S} = \begin{bmatrix} s_1 & s_2 \\ s_3 & s_4 \end{bmatrix}$ and $\boldsymbol{\beta} = \begin{bmatrix} \beta_1 & \beta_2 \end{bmatrix}$. We then compute the wiggliness score as follows:

$$\int g''(x)^2 = \boldsymbol{S\beta} = \begin{bmatrix} s_1\beta_1 & + & s_2\beta_2 \\ s_3\beta_1 & + & s_4\beta_2 \end{bmatrix}.$$

Remember, if this expression is zero the overall function is be a straight line. Thus, we can shrink or penalise our GAM by appending rows of $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} s_3 \\ s_4 \end{bmatrix}$ To our model from Equation 4 as follows:

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ s_1 \\ s_2 \end{bmatrix} \beta_1 + \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_n \\ s_3 \\ s_4 \end{bmatrix} \beta_2 + \epsilon.$$

The more times we append rows the more we favor a smooth model over a wiggly one.

References

R Core Team (2020). R: A language and environment for statistical computing. manual, Vienna, Austria. tex.organization: R Foundation for Statistical Computing.

Wood, S. (2017). *Generalized additive models: An introduction with R*. Chapman and Hall/CRC, 2 edition.