Chapter 7: GAMs in Practice

7.1 Specifying Smooths

- `s()` is for univariate smooths, isotropic smooths of several variables, and random effects
- `te()` is for tensor product smooths
- `ti()` is for tensor product smooths with the main effects (and lower order interactions) removed
- `t2()` alternate parameterization of tensor product smooths, useful for mixed modeling

Some important arguments are:

- `bs` type of basis
- `k` basis dimension
- `m` order of basis and penalty
- `id` labels the smooth; smooths sharing a label will have the same smoothing parameter
- `by` variable is either multiplied by this value or if it is a factor variable then a seperate curve is fit for each level of the factor

When `by` is a factor we should also include the factor variable in the model since the different curves will be subject to sum to zero constraints to make them identifiable. We still need to add an `id` argument to force each curve to share a smoothing parameter. For example `te(z, x, by = g, id = "a")` causes the smooths for each level of `g` to share the same smoothing parameter.

How Smooth Specification Works

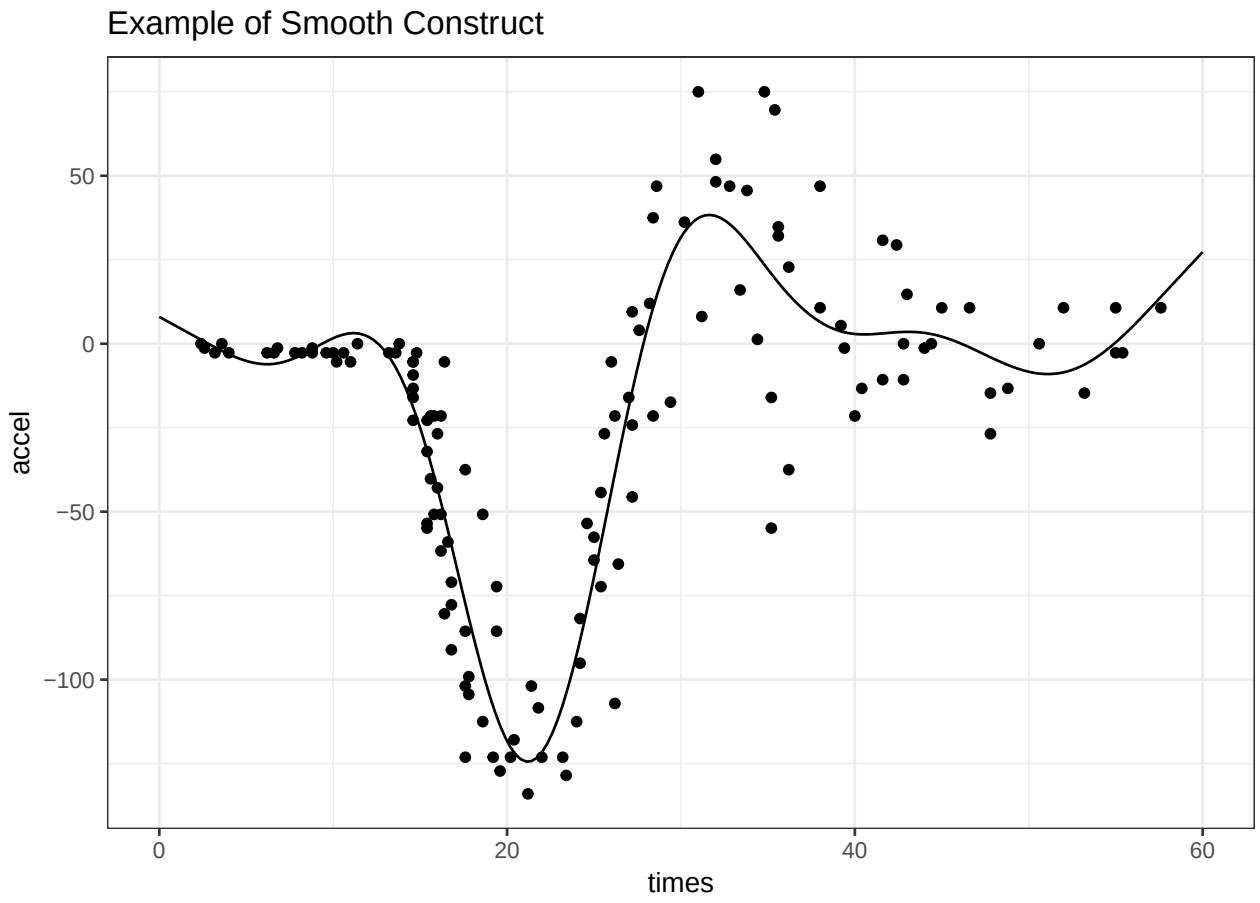
Each smooth is just a set of model matrix columns and corresponding penalty. Each smooth can be constructed with `smoothCon` and used to predict with `PredictMat` even outside of `mgcv`.

```
suppressPackageStartupMessages(library(mgcv))
suppressPackageStartupMessages(library(MASS))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(dplyr))
## Set up a smoother
sm <- smoothCon(s(times, k = 10), data = mcycle, knots = NULL)[[1]]

## Use it to fit a regression spline model
beta <- coef(lm(mcycle$accel ~ sm$X - 1))

## Get matrix mapping beta to spline prediction at `times`
pred_df <- data_frame(times = seq(0, 60, length.out = 200))
Xp <- PredictMat(sm, data = pred_df)
pred_df$preds <- Xp %*% beta

ggplot() +
  geom_point(aes(x = times, y = accel), data = mcycle) +
  geom_line(aes(x = times, y = preds), data = pred_df) +
  theme_bw() +
  xlab("times") +
  ylab("accel") +
  ggtitle("Example of Smooth Construct")
```

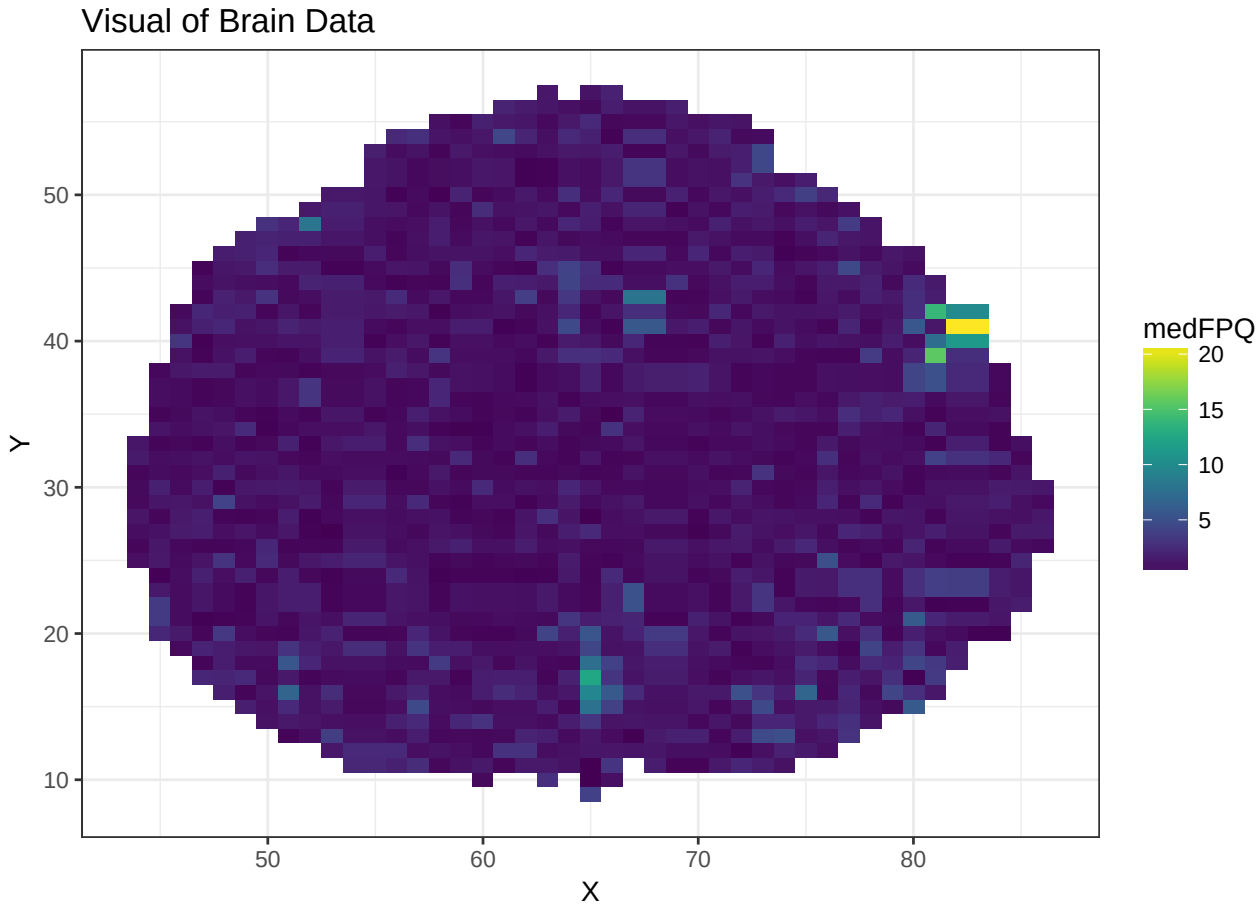


7.2 Brain Imaging Example

```
suppressPackageStartupMessages(library(gamair))
data(brain)
knitr::kable(head(brain))
```

X	Y	medFPQ	region	meanTheta
65	9	3.923048	NA	2.838555
60	10	0.492985	0	1.218145
63	10	2.759576	NA	1.983947
65	10	0.000003	0	1.306030
66	10	0.854175	0	-2.496284
54	11	2.235795	NA	1.648068

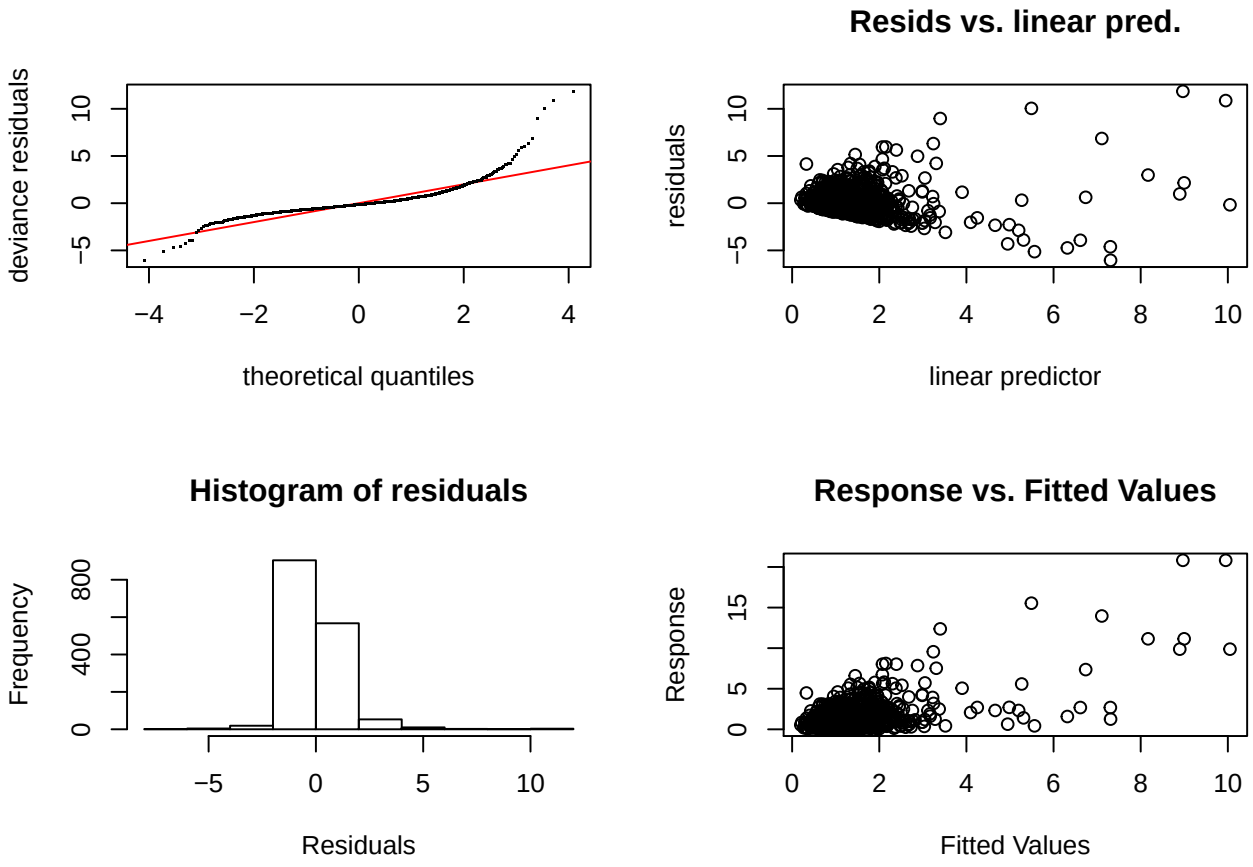
```
ggplot(aes(x = X, y = Y, fill = medFPQ), data = brain) +
  geom_raster() +
  theme_bw() +
  viridis::scale_fill_viridis() +
  ggtitle("Visual of Brain Data")
```




```
brain <- brain[brain$medFPQ > 5e-3, ]
```

We are going to model the `medFPQ` as a function of `X` and `Y`. The response data are strictly positive and slightly skewed so some transformation is necessary, but we can check this with `gam.check`.

```
m0 <- gam(medFPQ ~ s(Y, X, k = 100), data = brain)
gam.check(m0)
```




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 6 iterations.
## The RMS GCV score gradient at convergence was 6.236018e-05 .
## The Hessian was positive definite.
## Model rank =  100 / 100
##
## 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(Y,X) 99.000 86.782   0.863       0
```

`gam.check` gives you the maximum possible degrees of freedom (`k'`), the actual effective degrees of freedom, the ratio of the model estimated scale parameter to an estimate based on differencing neighbouring residuals (`k-index`), and the p-value associated with this.

Also `gam.check` produces some residual plots. These show that the variance seems to increase with the fitted value, impying that our model is misspecified.

If we assume that $var(y_i) \propto \mu_i^{\beta}$ where $\mu_i = E(y_i)$ then we can estimate β using a simple regression:

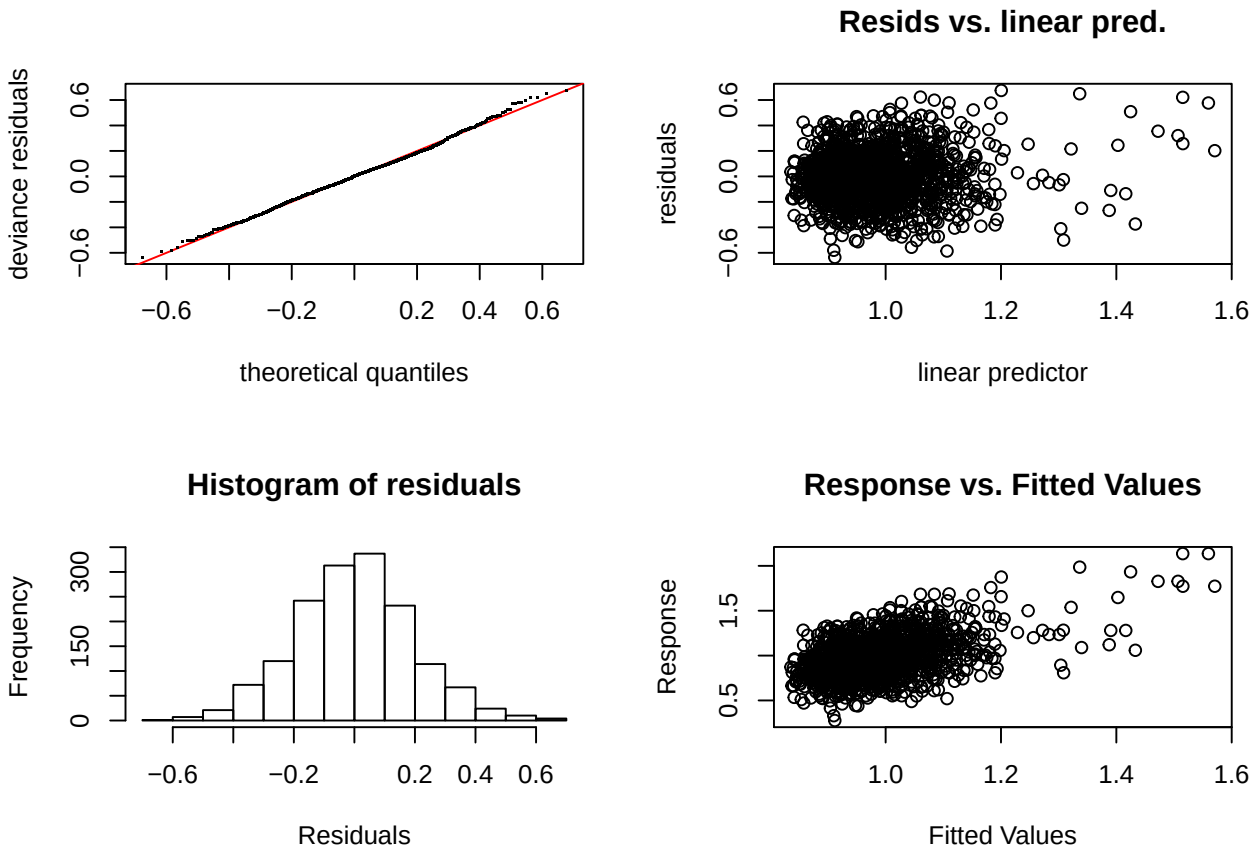
```
e <- residuals(m0)
fitted_values <- fitted(m0)
summary(lm(log(e^2) ~ log(fitted_values)))
```

```
##
## Call:
## lm(formula = log(e^2) ~ log(fitted_values))
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.401  -1.029   0.491   1.373   6.939
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -1.96105    0.05764  -34.02   <2e-16 ***
## log(fitted_values)  1.91248    0.12435   15.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.222 on 1562 degrees of freedom
## Multiple R-squared:  0.1315, Adjusted R-squared:  0.131
## F-statistic: 236.5 on 1 and 1562 DF,  p-value: < 2.2e-16
```

β is estimated to be around 2, so the variance increases with the square of the mean so the *gamma* distribution seems to fit.

```
m1 <- gam(medFPQ ^ .25 ~ s(Y, X, k = 100), data = brain)
gam.check(m1)
```




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 4 iterations.
## The RMS GCV score gradient at convergence was 4.811308e-06 .
## The Hessian was positive definite.
## Model rank =  100 / 100
```

```
##
## 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(Y,X) 99.00 64.50    0.92       0
```

```
m2 <- gam(medFPQ ~ s(Y, X, k = 100), data = brain, family = Gamma(link = log))
```

This 4th root transformation has biased the data on our response scale

```
print(c(actual_mean = mean(brain$medFPQ),
        fourth_mean = mean(fitted(m1)^4),
        gamma_mean = mean(fitted(m2))))
```

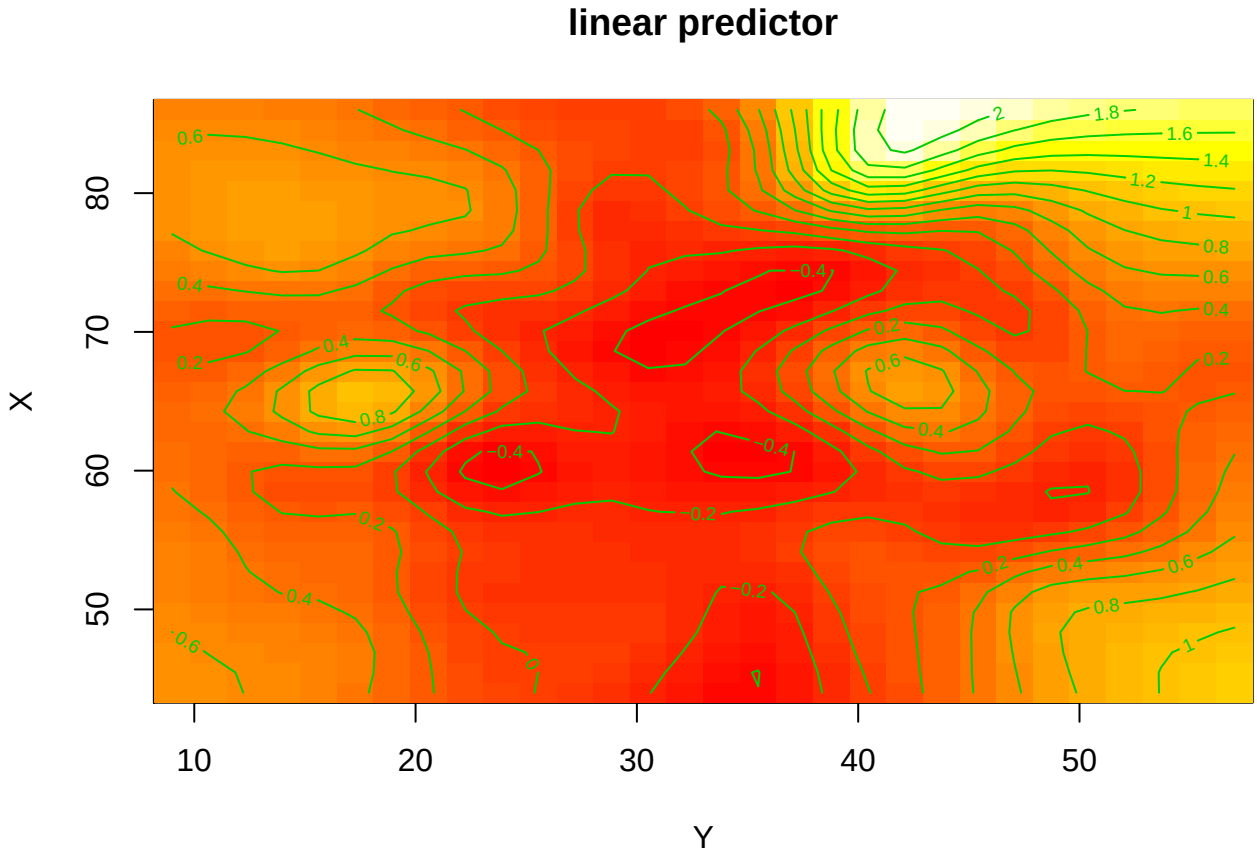
```
## actual_mean fourth_mean  gamma_mean
##   1.2503024   0.9855539   1.2114832
```

The `vis.gam` function can display predictions from a gam fit:

```
summary(m2)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## medFPQ ~ s(Y, X, k = 100)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.12031    0.01922   6.258 5.07e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F p-value
## s(Y,X) 60.61  77.82 4.794  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.307   Deviance explained = 26.4%
## GCV = 0.62169  Scale est. = 0.57802   n = 1564
```

```
vis.gam(m2, plot.type = "contour")
```



Perhaps a simpler additive model like

$$log(E(medFPQ_i)) = f_1(Y_i) + f_2(X_i), medFPQ_i \sim gamma$$

would be better.

```
m3 <- gam(medFPQ ~ s(Y, k = 30) + s(X, k = 30), data = brain, family = Gamma(link = log))
summary(m3)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## medFPQ ~ s(Y, k = 30) + s(X, k = 30)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.14358    0.02065   6.953 5.27e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df      F p-value
## s(Y)  9.577  11.95 11.948  <2e-16 ***
## s(X) 20.198  23.91  7.312  <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.161   Deviance explained = 20.5%
## GCV = 0.64535  Scale est. = 0.66688   n = 1564
```

The GCV score is higher (higher error), the percentage deviance explained is lower, and the AIC is higher for the additive model:

```
AIC(m2, m3)
```

```
##          df      AIC
## m2 62.61062 3321.681
## m3 31.77467 3393.738
```

The additive model also produces various horizontal and vertical stripes instead of interacting smoothly:

```
vis.gam(m3, plot.type = "contour")
```