Shape Constrained Regression for Excel

William Chiu

2022-10-20

Summary

Generalized Additive Models (gam) estimate non-linear but additive relationships between a response and a set of predictors. However, when the training data has a high noise to signal ratio, gam could estimate wiggly (non-monotonic) relationships that are difficult to explain. Shape Constrained Additive Models (scam) impose user-defined monotonic relationships that can be explained by the user.

The `mgcv` and `scam` packages are highly effective tools for developing gam and scam models. Both packages use the p-spline basis to construct non-linear features. Although p-splines have desirable properties, they are difficult to implement in other software tools like Excel. There is no closed-form equation that I could export from gam/scam and into Excel for an end-user.

Multivariate Adaptive Regression Splines (mars), from the `earth` package, allow us to approximate the scam model with linear basis functions – which are Excel-friendly.

Through a data simulation, I demonstrate how to fit gam and scam models using the `mgcv` and `scam` packages. Then I approximate the scam model and provide an Excel-friendly regression equation. Reader should be aware that MARS does *not guarantee* monotonicity. Therefore, after fitting an approximation of the scam model, reader should test the predictions by feeding extreme values of the predictors into the final model.

Data Generating Process (DGP)

The chunk generates 3 data frames: full data, training data, and test data. To demonstrate the wiggliness associated with gam, the sample size of the training data is limited to only 2000 observations.

Each data frame contains a binary response and two predictors. The first predictor has a monotonically increasing relationship with the response, while the second predictor has a monotonically decreasing relationship with the response. Both relationships are sigmoidal and hence highly non-linear.

In addition, the event rate of the response is rare.

```
library(mgcv)
library(scam)
library(tidyverse)
library(caTools)
library(MLmetrics)
library(earth)
library(plotmo)
library(furrr)

plan(multisession, workers = 6) # set to 2 for most PCs
```

```r
set.seed(2001)

nobs <- 100000

x1 <- rnorm(nobs)
x2 <- rnorm(nobs)

# Population log odds is a function of sigmoid features

z <- -35 + 5 * SSfpl(x1, -2, 2, 0, 0.5) - 5 * SSfpl(x2, -1, 1, 0, 0.1) +
  rnorm(nobs,0,20)

y <- rbinom(nobs, 1, prob=boot::inv.logit(z))

full_data <- data.frame(y=y, x1=x1, x2=x2)

summary(full_data)
```

```
##        y                 x1                  x2
##  Min.   :0.00000   Min.   :-4.469540   Min.   :-4.349958
##  1st Qu.:0.00000   1st Qu.:-0.672327   1st Qu.:-0.667040
##  Median :0.00000   Median : 0.006574   Median : 0.002566
##  Mean   :0.05212   Mean   : 0.004819   Mean   : 0.000159
##  3rd Qu.:0.00000   3rd Qu.: 0.681828   3rd Qu.: 0.676167
##  Max.   :1.00000   Max.   : 4.071722   Max.   : 4.590295
```

```r
train_index <- sample.split(full_data$y, SplitRatio = 2/100)

train_data <- full_data[train_index,] # small
test_data <- full_data[!train_index,] # large
```

GAM

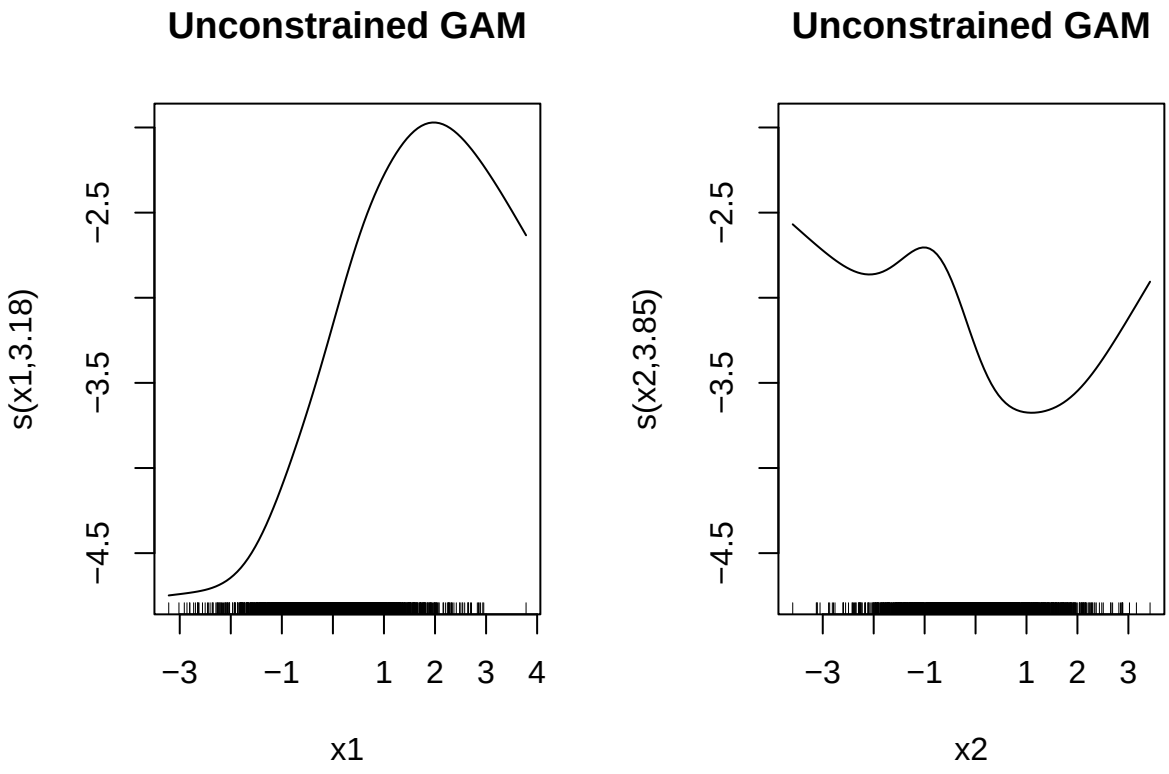
Due to the high noise to signal ratio in the training data, gam estimates very wiggly relationships between the binary response and the predictors. The y-axis is on the log-odds (or logit) scale.

```r
## fit a gam

## when population error is high, gam returns non-monotonic relationships
## even when the population relationships are monotonic

mod_gam <- gam(y ~ s(x1) + s(x2), data = train_data, family = binomial)

summary(mod_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
```

```
## y ~ s(x1) + s(x2)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -3.2119     0.1329  -24.17   <2e-16 ***
## ---
## 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(x1) 3.180  4.051  41.96  <2e-16 ***
## s(x2) 3.846  4.845  13.73   0.015 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0351   Deviance explained = 8.26%
## UBRE = -0.61702  Scale est. = 1         n = 2000
```

```
plot(mod_gam, pages=1, se=FALSE, main="Unconstrained GAM",
     shift=coef(mod_gam)[1])
```



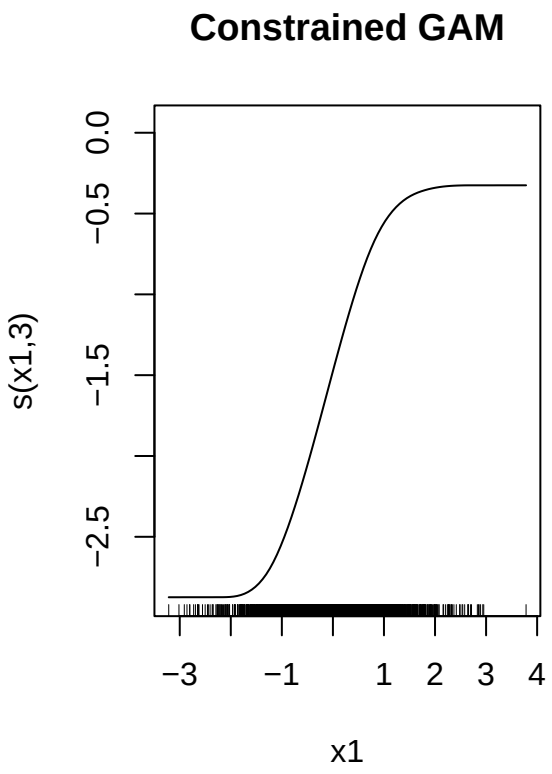
SCAM

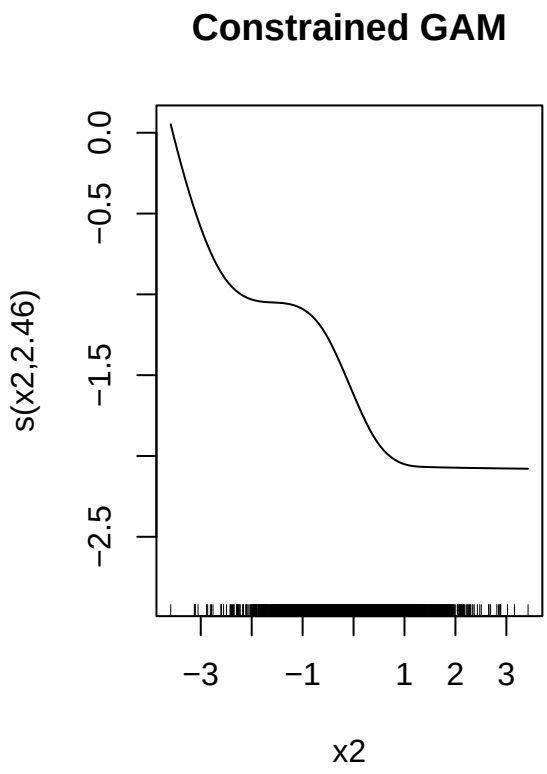
Shape constraints on each predictor reduce variance (wiggliness). Out-of-sample performance is similar between `gam` and `scam`.

```
## gam with monotonic constraints

## scam imposes monotonic relationships between each
## feature and the binary response
## mpi = monotonic p-spline increasing
## mpd = monotonic p-spline decreasing

mod_mono_gam <- scam(y ~ s(x1, bs="mpi") + s(x2, bs="mpd"),
                     data = train_data, family = binomial)

summary(mod_mono_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## y ~ s(x1, bs = "mpi") + s(x2, bs = "mpd")
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.467     19.323  -0.076    0.939
##
## Approximate significance of smooth terms:
##         edf Ref.df Chi.sq  p-value
## s(x1) 3.003  3.005  42.10 3.89e-09 ***
## s(x2) 2.463  2.632  16.07  0.00088 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0328   Deviance explained = 7.93%
## UBRE score = -0.61724  Scale est. = 1         n = 2000
##
## BFGS termination condition:
## 0.002513042
```

```
plot(mod_mono_gam, pages=1, se=FALSE, main="Constrained GAM",
     shift=coef(mod_mono_gam)[1])
```






```
map_dfr(list(mod_gam=mod_gam, mod_mono_gam=mod_mono_gam), AIC) %>%
  pivot_longer(everything(), names_to="model", values_to="AIC") %>%
  knitr::kable()
```

model	AIC
mod_gam	765.9699
mod_mono_gam	765.5259

```
## test set performance

preds_gam <- predict(mod_gam, newdata=test_data, type="response")
preds_mono_gam <- predict(mod_mono_gam, newdata=test_data,
                          type="response")

map_dfr(list(gam=preds_gam, mono_gam=preds_mono_gam), function(x){
  caTools::colAUC(x, test_data$y)
}) %>%
  pivot_longer(everything(), names_to="model", values_to="ROC-AUC") %>%
  knitr::kable()
```

model	ROC-AUC
gam	0.7097935

model	ROC-AUC
mono_gam	0.7098818

```
map_dfr(list(gam=preds_gam, mono_gam=preds_mono_gam), function(x){
  MLmetrics::LogLoss(x, test_data$y) }) %>%
  pivot_longer(everything(), names_to="model", values_to="Log Loss") %>%
  knitr::kable()
```

model	Log Loss
gam	0.1908626
mono_gam	0.1909576

Generate Pseudo-Training Data

The next step is controversial. I pretend that the scam model is the data generating process. In the actual training data, we observe binary outcomes rather than the log odds. In the pseudo data, I ignore the binary outcomes and "observe" the log odds from the scam model.

The reader should be aware that in the actual training data, `y` is a binary response. In the pseudo data, `y` is log odds from the scam model, which is analogous to `z` in the true data generating process.

```
## generate pseudo-data using Mono GAM

## suppose the scam model is the data generating process (dgp).
## append the log odds for each observation in the training
## data

pseudo_train_data <- train_data

pseudo_train_data$y <- as.numeric(predict(mod_mono_gam, newdata=train_data))

summary(pseudo_train_data)
```

```
##        y                 x1                  x2
##  Min.   :-5.0697   Min.   :-3.215527   Min.   :-3.58923
##  1st Qu.:-3.9582   1st Qu.:-0.712003   1st Qu.:-0.73931
##  Median :-3.1844   Median : 0.003207   Median :-0.03104
##  Mean   :-3.2400   Mean   :-0.023366   Mean   :-0.04149
##  3rd Qu.:-2.5328   3rd Qu.: 0.639760   3rd Qu.: 0.65565
##  Max.   :-0.9548   Max.   : 3.783016   Max.   : 3.42584
```

Approximate SCAM with MARS

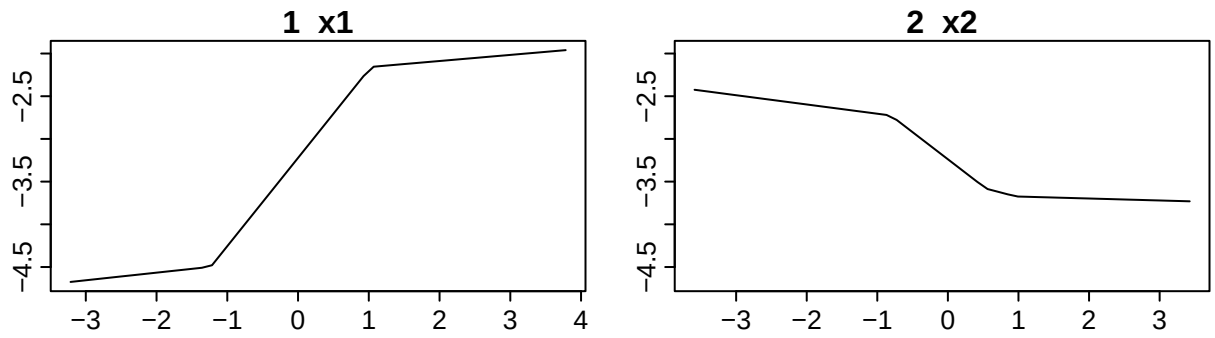
```
## approximate the Mono GAM model with MARS

## unfortunately, both gam and scam models are very difficult to
```

```r
## implement outside of R due to how the basis functions are defined.
## an alternative is to approximate the scam relationships
## with simpler basis functions like the linear basis (which can be
## easily implemented in Excel) -- the linear basis is also called the
## reLU (rectified linear unit)


mod_logit_pseudo <- earth(y ~ x1 + x2, data=pseudo_train_data)

summary(mod_logit_pseudo)
```

```
## Call: earth(formula=y~x1+x2, data=pseudo_train_data)
##
##                   coefficients
## (Intercept)         -3.9947320
## h(x1- -1.23371)      0.9448377
## h(1.0299-x1)        -0.0892467
## h(x1-1.0299)        -0.8733752
## h(x2- -0.807467)    -0.5263907
## h(x2-0.54224)        0.4177184
## h(0.963698-x2)       0.1083663
## h(x2-0.963698)       0.0856736
##
## Selected 8 of 8 terms, and 2 of 2 predictors
## Termination condition: RSq changed by less than 0.001 at 8 terms
## Importance: x1, x2
## Number of terms at each degree of interaction: 1 7 (additive model)
## GCV 0.002690246    RSS 5.300087    GRSq 0.9967042    RSq 0.9967502
```

```r
plotmo(mod_logit_pseudo, caption="MARS with Pseudo Data")
```

```
##  plotmo grid:    x1          x2
##         0.003206692 -0.03104272
```



Since the pseudo model is in the log-odds scale, I need to convert the predictions into probabilities to compare against the probability predictions from gam and scam.

```
## test set performance

preds_pseudo <- boot::inv.logit(predict(mod_logit_pseudo, newdata=test_data))

map_dfr(list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo=preds_pseudo), function(x){
  caTools::colAUC(x, test_data$y)
}) %>%
  pivot_longer(everything(), names_to="model", values_to="ROC-AUC") %>%
  knitr::kable()
```

model	ROC-AUC
gam	0.7097935
mono_gam	0.7098818
pseudo	0.7091226

```
map_dfr(list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo=preds_pseudo), function(x){
  MLmetrics::LogLoss(x, test_data$y)
}) %>%
  pivot_longer(everything(), names_to="model", values_to="Log Loss") %>%
  knitr::kable()
```

model	Log Loss
gam	0.1908626
mono_gam	0.1909576
pseudo	0.1910508

Excel-friendly Equation

The equation below predicts the log odds or logit.

```
cat(format(mod_logit_pseudo, style="pmax", use.names=TRUE))
```

```
##   -3.994732
##   +  0.9448377 * pmax(0,         x1 -  -1.233707)
##   - 0.08924666 * pmax(0,   1.029901 -         x1)
##   -  0.8733752 * pmax(0,         x1 -   1.029901)
##   -  0.5263907 * pmax(0,         x2 - -0.8074673)
##   +  0.4177184 * pmax(0,         x2 -  0.5422403)
##   +  0.1083663 * pmax(0,  0.9636977 -         x2)
##   + 0.08567365 * pmax(0,         x2 -  0.9636977)
```

```
## Compare predict() against score_function

as.func <- function(object, digits = 20, use.names = TRUE, ...)
  eval(parse(text=paste(
    "function(x){\n",
    "if(is.vector(x))\n",
    "  x <- matrix(x, nrow = 1, ncol = length(x))\n",
    "with(as.data.frame(x),\n",
    format(object, digits = digits, use.names = use.names, style = "pmax", ...),
    ")\n",
    "}\n", sep = "")))

score_function <- as.func(mod_logit_pseudo)

compare_df <- expand.grid(x1 = seq(-10, 10, 0.1),
           x2 = seq(-10, 10, 0.1))

earth_preds <- predict(mod_logit_pseudo, newdata=compare_df)

score_preds <- score_function(compare_df)

max(abs(earth_preds - score_preds))
```

```
## [1] 1.776357e-15
```

Testing monotonicity

MARS does not guarantee monotonic relationships. Reader should always test the model for violations of monotonicity by feeding extreme predictor values into the model.

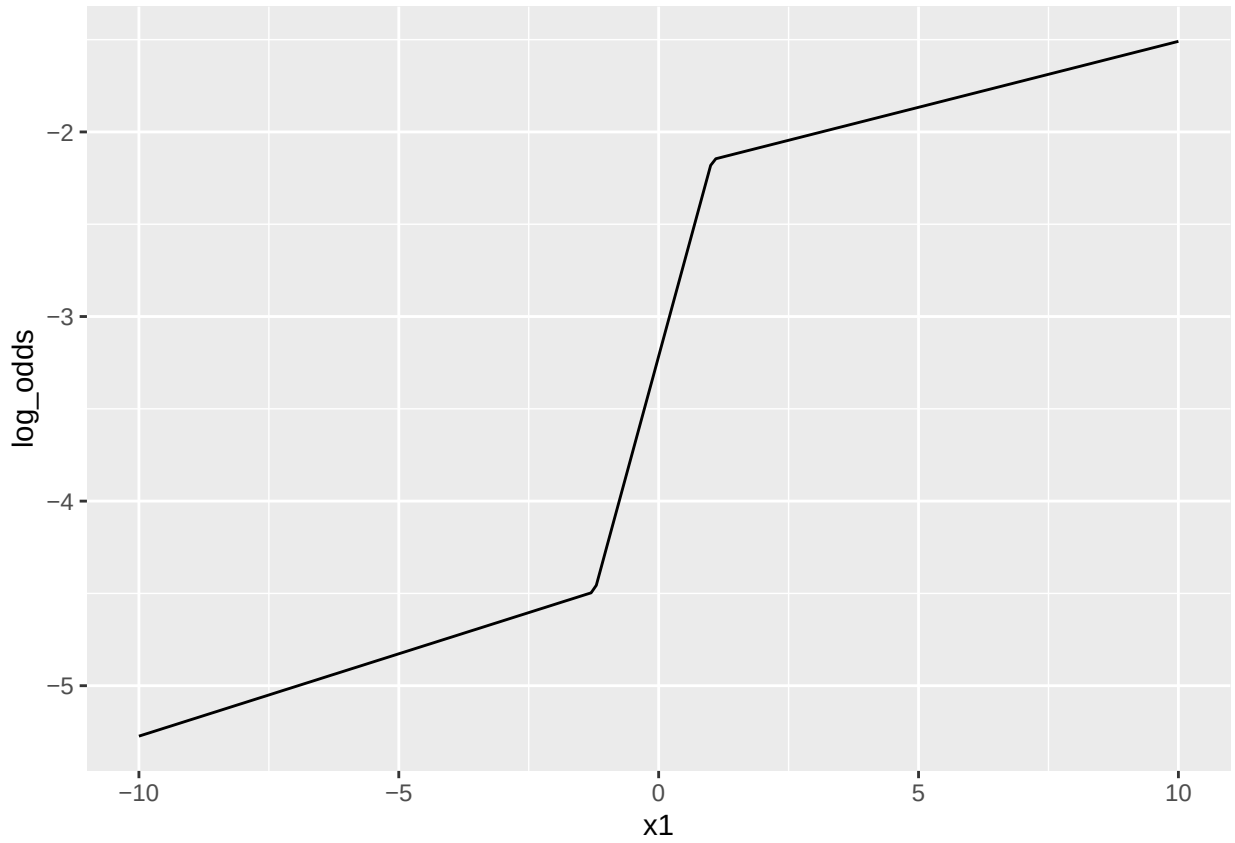
```r
mono_check <- function(x, increasing=TRUE){
  if(increasing==TRUE){
    out <- all(x == cummax(x))
  } else {
    out <- all(x==cummin(x))
  }

  return(out)
}

df_x1 <- data.frame(x1 = seq(-10,10, 0.1),
                    x2 = mean(train_data$x2))

df_x2 <- data.frame(x1 = mean(train_data$x1),
                    x2 = seq(-10,10, 0.1))

df_x1$log_odds <- score_function(df_x1)

df_x2$log_odds <- score_function(df_x2)

mono_check(df_x1$log_odds)
```

```
## [1] TRUE
```

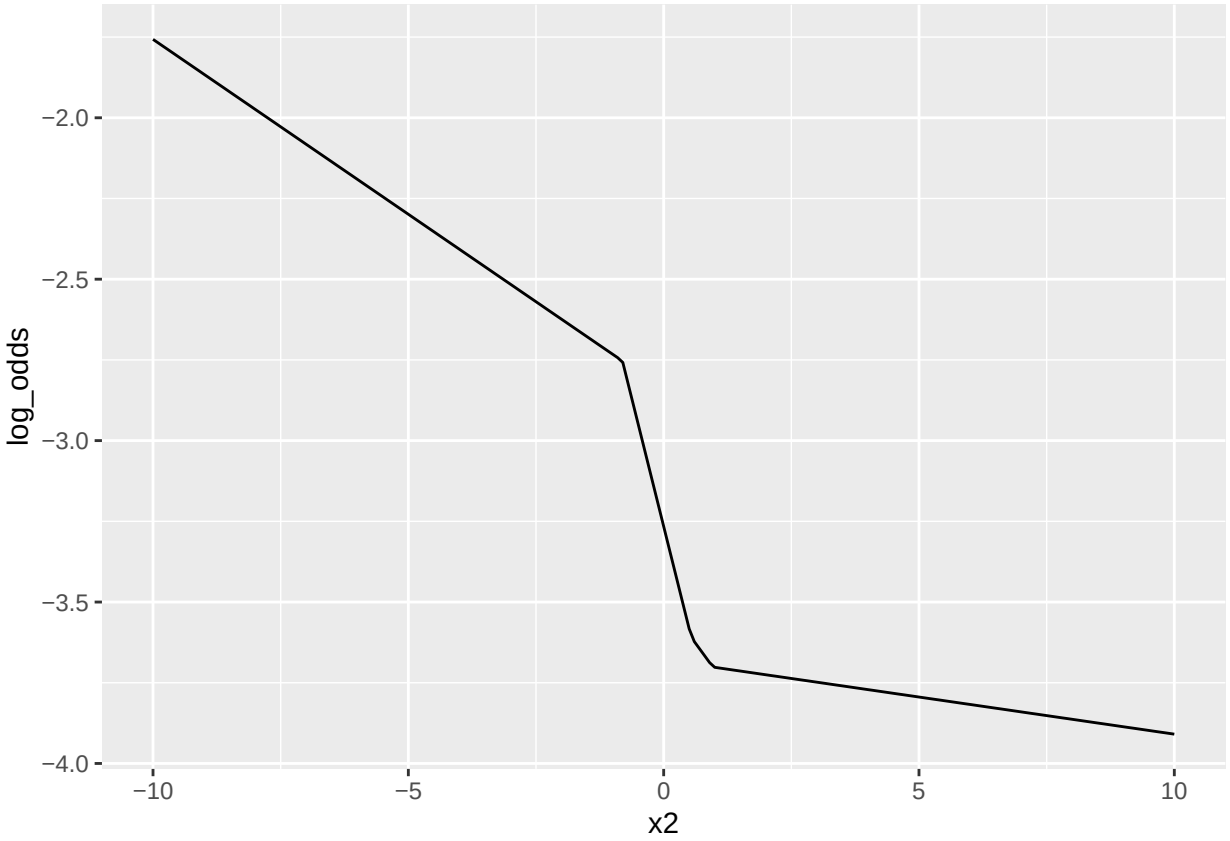
```r
mono_check(df_x2$log_odds, FALSE)
```

```
## [1] TRUE
```

```r
ggplot(df_x1, aes(x=x1, y=log_odds)) + geom_line()
```




```
ggplot(df_x2, aes(x=x2, y=log_odds)) + geom_line()
```



How often does monotonicity fail?

Since MARS does not guarantee monotonicity. I repeated, 60 times, the following steps:

1. Set a new seed value
2. Generate full, train, and test sets
3. Fit a scam model to the training data
4. Generate pseudo data
5. Fit a MARS model to the pseudo data
6. Check shape constraints (x1 should increase with y and x2 should decrease with y)

```
simulate_one <- function(myseed){

  # Set a new seed value

  set.seed(myseed)

  nobs <- 100000

  x1 <- rnorm(nobs)
  x2 <- rnorm(nobs)

  z <- -35 + 5 * SSfpl(x1, -2, 2, 0, 0.5) - 5 * SSfpl(x2, -1, 1, 0, 0.1) +
    rnorm(nobs,0,20)
```

```
  y <- rbinom(nobs, 1, prob=boot::inv.logit(z))

  # Generate full, train, and test sets
  full_data <- data.frame(y=y, x1=x1, x2=x2)

  train_index <- sample.split(full_data$y, SplitRatio = 2/100)
  train_data <- full_data[train_index,] # small
  test_data <- full_data[!train_index,] # large

  # SCAM

  my_mod_mono_gam <- scam(y ~ s(x1, bs="mpi") + s(x2, bs="mpd"),
                          data = train_data, family = binomial)

  # Pseudo data

  pseudo_train_data <- train_data

  pseudo_train_data$y <- as.numeric(predict(my_mod_mono_gam, newdata=train_data))

  summary(pseudo_train_data)

  # MARS

  my_mod_pseudo_mars <- earth(y ~ x1 + x2, data=pseudo_train_data)


  # Check shape constraints

  df_x1 <- data.frame(x1 = seq(-10,10, 0.1),
                      x2 = mean(train_data$x2))

  df_x2 <- data.frame(x1 = mean(train_data$x1),
                      x2 = seq(-10,10, 0.1))

  df_x1$log_odds <- predict(my_mod_pseudo_mars, newdata=df_x1)

  df_x2$log_odds <- predict(my_mod_pseudo_mars, newdata=df_x2)

  success <- mono_check(df_x1$log_odds) & mono_check(df_x2$log_odds, FALSE)

  return(success)
}

mono_success <- future_map_int(1:60, simulate_one,
                           .options=furrr_options(seed=NULL,
                                                  packages=c('earth', 'scam')
                                                  )
                           )

future:::ClusterRegistry("stop")
```

```
mean(mono_success)
```

```
## [1] 0.7
```

MARS successfully approximated the scam model about 70% of the time.

Why Pseudo Data?

The pseudo data filters out the noise in the latent response (`z`), which drives the binary response (`y`). Although we could directly pass the actual training data into MARS, the noise in the data generating process could cause `earth` to return non-monotonic relationships.

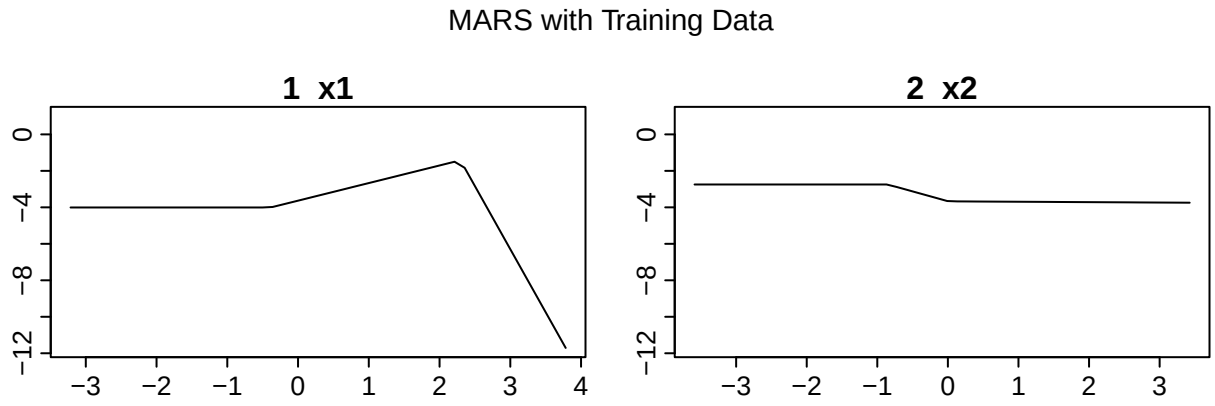
Reader should notice that `y` in the actual training data is a binary outcome.

```
mod_earth <- earth(y ~ x1 + x2, data=train_data, glm=list(family=binomial))

summary(mod_earth)
```

```
## Call: earth(formula=y~x1+x2, data=train_data, glm=list(family=binomial))
##
## GLM coefficients
##                             y
## (Intercept)      -3.1240890
## h(x1- -0.387815)  0.9645083
## h(x1-2.2952)     -7.8781852
## h(x2- -0.852946) -1.0756331
## h(x2-0.00710431)  1.0539147
##
## GLM (family binomial, link logit):
##  nulldev   df       dev   df   devratio     AIC iters converged
##   817.45 1999   754.646 1995     0.0768   764.6     6         1
##
## Earth selected 5 of 8 terms, and 2 of 2 predictors
## Termination condition: RSq changed by less than 0.001 at 8 terms
## Importance: x1, x2
## Number of terms at each degree of interaction: 1 4 (additive model)
## Earth GCV 0.04804009    RSS 95.2174    GRSq 0.0264512    RSq 0.0342279
```

```
plotmo(mod_earth, type="link", caption="MARS with Training Data")
```

```
##  plotmo grid:    x1          x2
##         0.003206692 -0.03104272
```



Conclusion

Imposing shape constraints on data with high noise to signal ratios could greatly reduce variance (wiggliness). The `scam` package is highly effective in developing models with user-defined monotonicity constraints for each predictor. MARS can approximate the scam model with linear basis functions, which are easy to write down on paper and implement in an Excel formula.