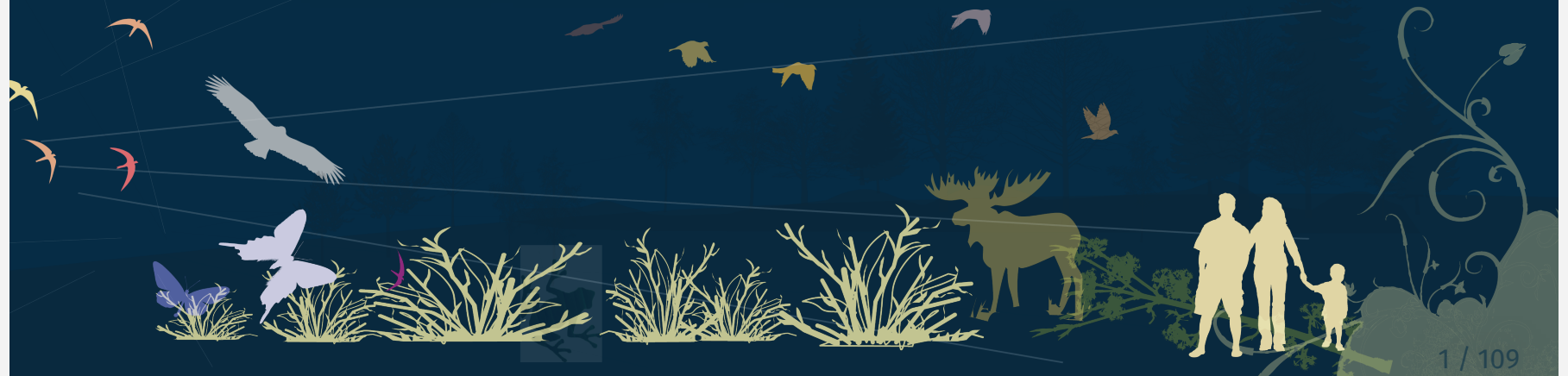
Workshop 8: Generalized additive models

QCBS R Workshop Series

Québec Centre for Biodiversity Science



About this workshop

Required packages

- ggplot2
- itsadug
- mgcv

```
install.packages(c('ggplot2', 'itsadug', 'mgcv'))
```

Workshop overview

1. The linear model... and where if fails
2. Introduction to GAM
3. Multiple smooth terms
4. Interactions
5. Changing basis
6. Other distributions
7. Quick intro to GAMM
8. GAM behind the scene

Learning objectives

1. Use the mgcv package to fit non-linear relationships,
2. Understand the output of a GAM to help you understand your data,
3. Use tests to determine if a non-linear model fits better than a linear one,
4. Include smooth interactions between variables,
5. Understand the idea of a basis function, and why it makes GAMs so powerful,
6. Account for dependence in data (autocorrelation, hierarchical structure) using GAMMs.

Prerequisites

> Some experience in R (enough to be able to run a script and examine data and R objects) a basic knowledge of regression (you should know what we mean by linear regression and ANOVA).

1. The linear model

...and where if fails

Linear regression

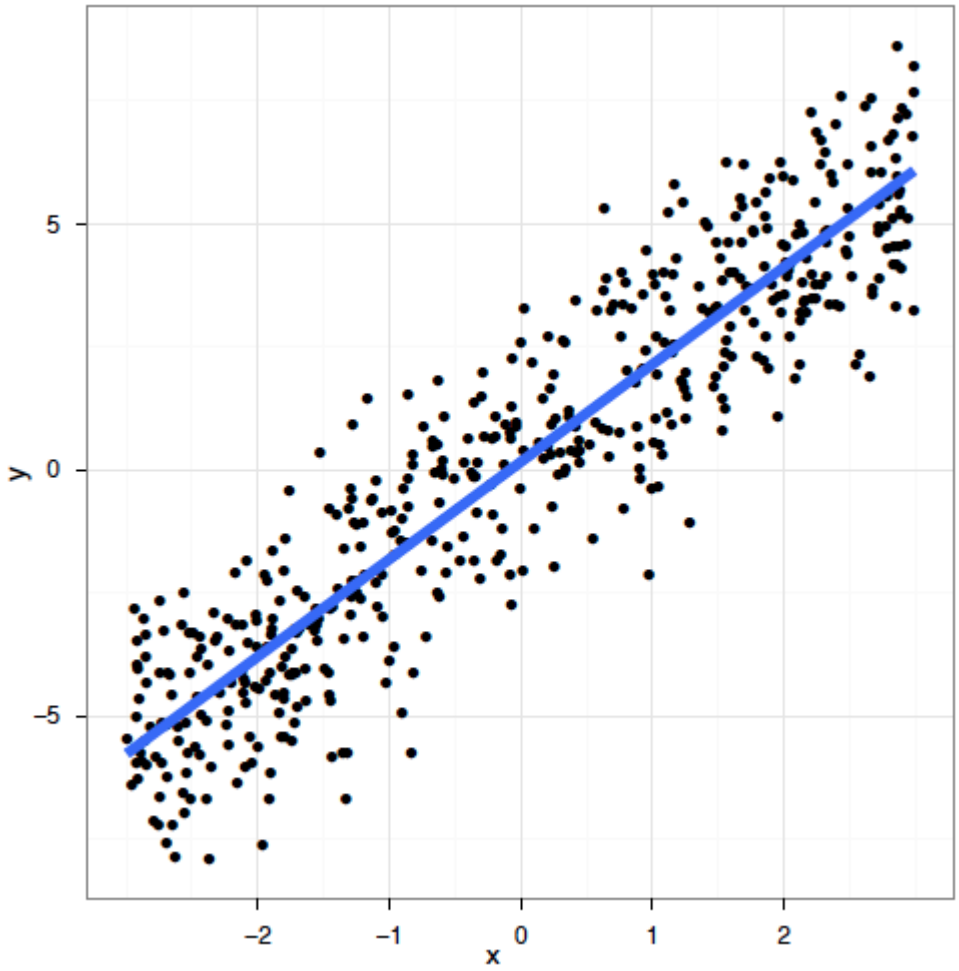
Regression is the workhorse of statistics. It allows us to model a response variable as a function of predictors plus error.

As we saw in the linear models workshop, regression makes 4 major assumptions:

1. Normally distributed error
2. Homogeneity of the variance
3. Indenpendance of the errors
4. The response is linear : $y = \beta_0 + \beta_1 x$

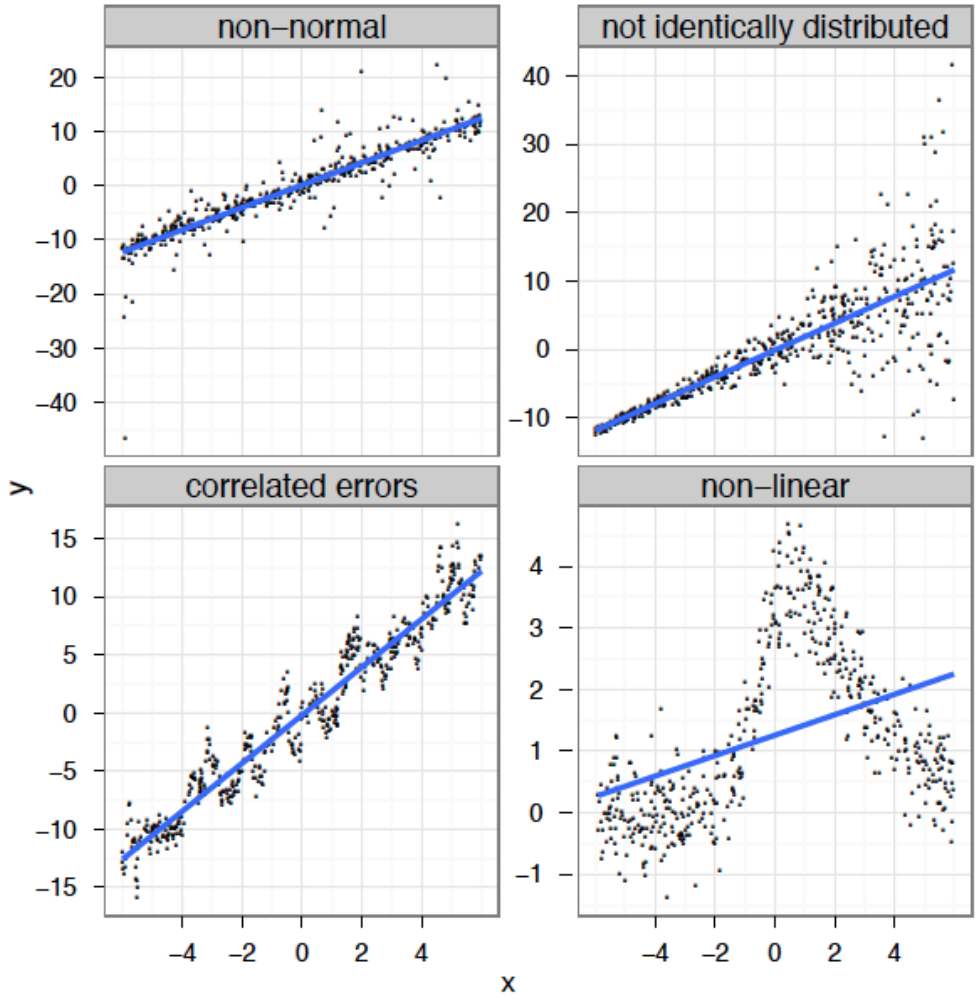
Linear regression

There's only one way for the linear model to be right:



Linear regression

And yet so many ways for it to fail:



Linear regression

What's the problem and do we fix it?

A **linear model** tries to fit the best **straight line** that passes through the data, so it doesn't work well for all datasets.

In contrast, a **GAM** can capture complexe relationships by fitting a **non-linear smooth function** through the data, while controlling how wiggly the smooth can get (more on this later).

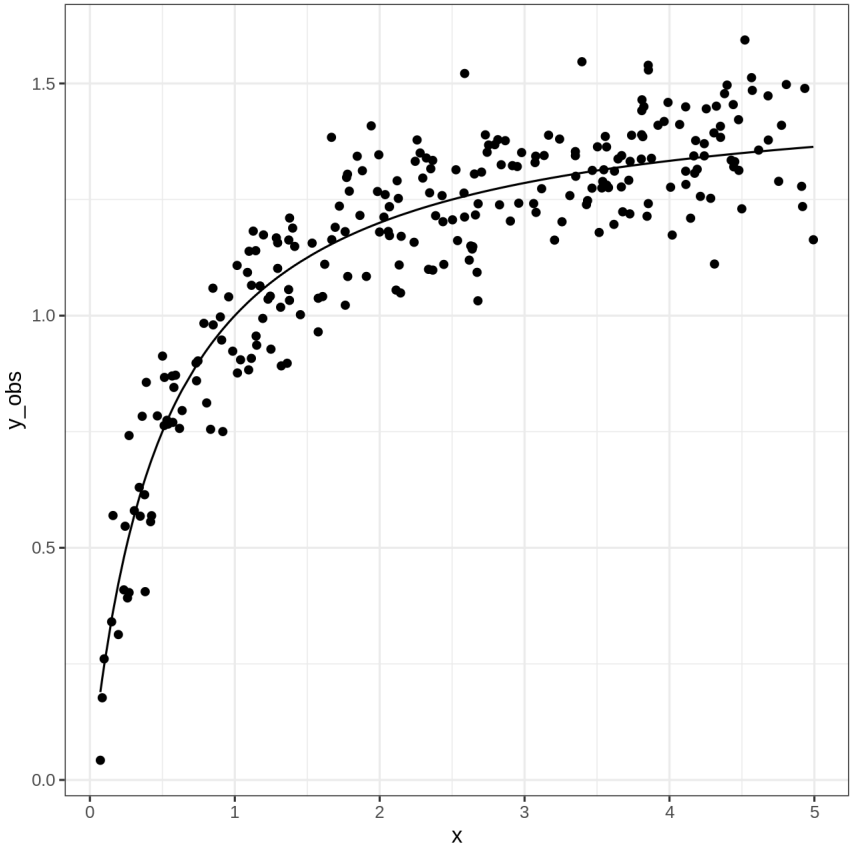
2. Introduction to GAM

Generalized Additive Models (GAM)

Let's look at an example. First, we'll generate some data, and plot it.

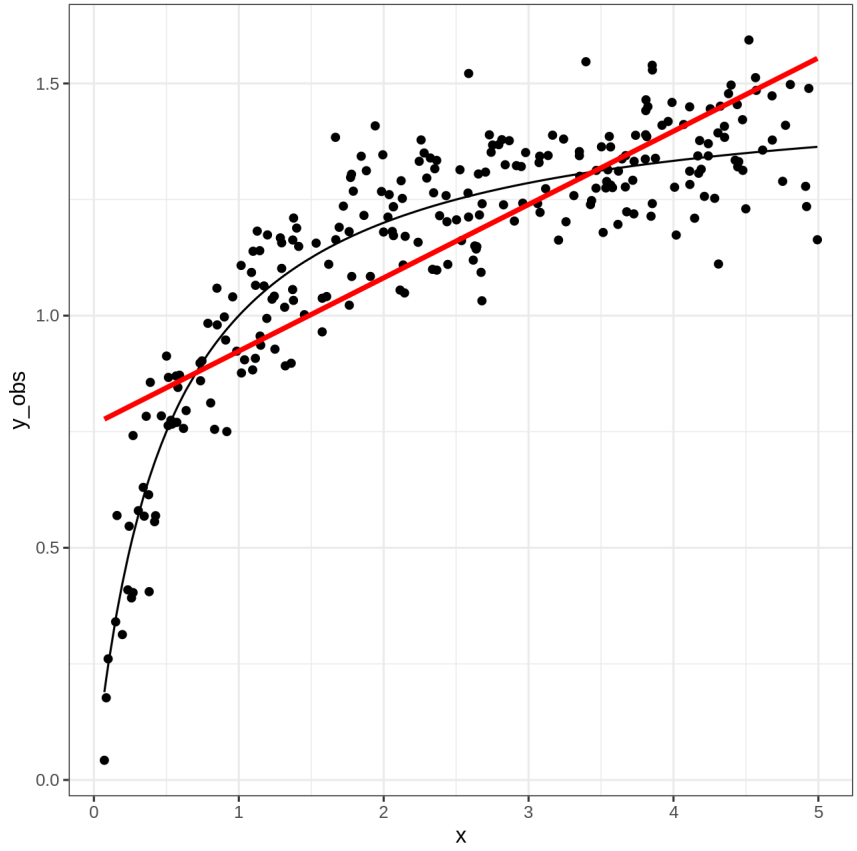
```
library(ggplot2)
set.seed(10)
n ← 250
x ← runif(n,0,5)
y_model ← 3*x/(1+2*x)
y_obs ← rnorm(n,y_model,0.1)
data_plot ← qplot(x, y_obs) +
  geom_line(aes(y=y_model)) +
  theme_bw()
data_plot
```

GAM



GAM

Trying to fit these data as a linear regression model, we would violate the assumptions listed above.



GAM

In GAM, the relationship between the response variable and the predictors is:

$$y = \alpha + s(x_1) + s(x_2) + \ldots + \epsilon$$

One big advantage of using GAM over a manual specification of the model is that the optimal shape, i.e. the degree of smoothness of `s(x)`, is determined automatically using a generalized cross-validation

GAM

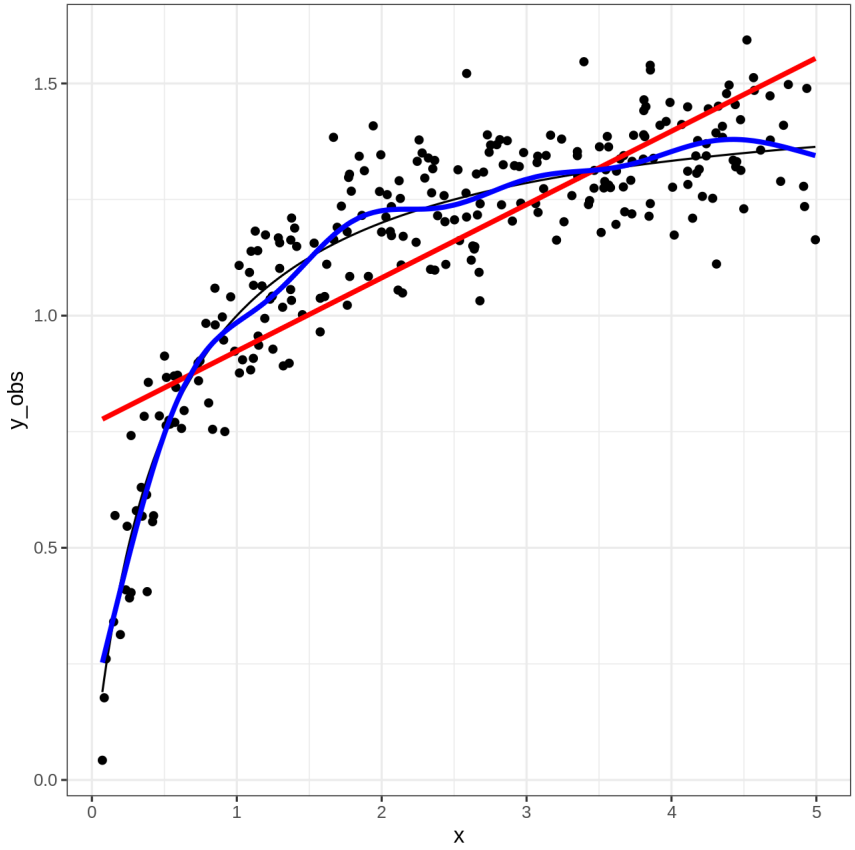
Let's try to fit the data using a smooth function with the function `mgcv::gam()`

```
library(mgcv)
gam_model ← gam(y_obs ~ s(x))
summary(gam_model)

data_plot ← data_plot +
     geom_line(colour = "blue", size = 1.2, aes(y = fitted(gam_model)))
data_plot
```

GAM

```
#
# Family: gaussian
# Link function: identity
#
# Formula:
# y_obs ~ s(x)
#
# Parametric coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 1.154422   0.006444   179.1   <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(x) 8.317  8.872 171.3  <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# R-sq.(adj) =  0.859   Deviance explained = 86.3%
# GCV = 0.010784  Scale est. = 0.010382  n = 250
```

GAM

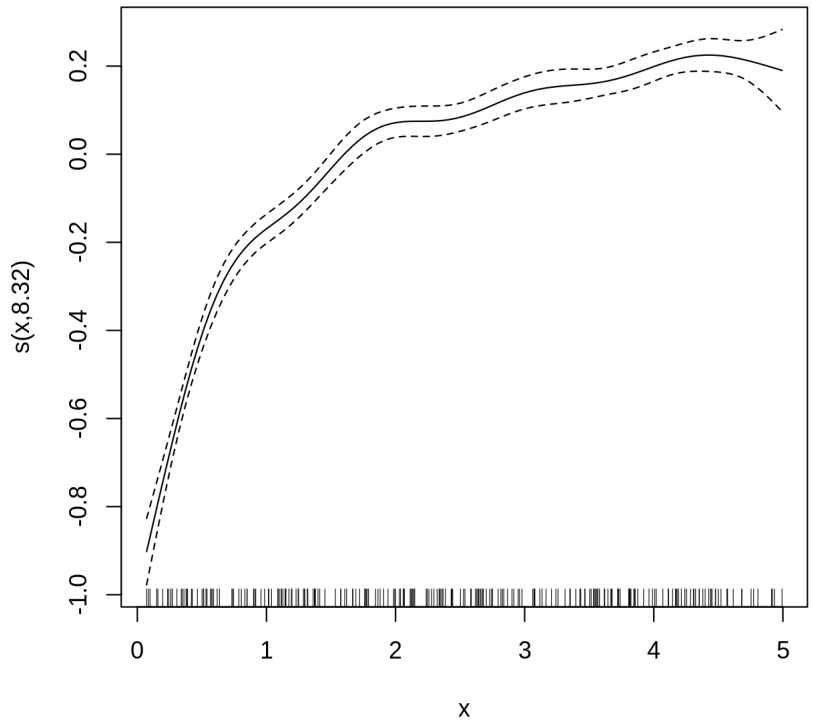


Note: as opposed to one fixed coefficient, \beta in linear regression, the smooth function can continually change over the range of the predictor x

GAM

The `mgcv` package also includes a default plot to look at the smooths:

```
plot(gam_model)
```



Test for linearity using GAM

We can use `gam()` and `anova()` to test whether an assumption of linearity is justified. To do so, we must simply set our smoothed model so that it is nested in our linear model.

```
linear_model ← gam(y_obs ~ x) # fit a regular linear model using gam()
nested_gam_model ← gam(y_obs ~ s(x) + x)
anova(linear_model, nested_gam_model, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y_obs ~ x
# Model 2: y_obs ~ s(x) + x
#   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
# 1    248.00     6.5846
# 2    240.13     2.4988 7.8721   4.0858 < 2.2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the model `y_obs~s(x)` gives exactly the same results as `y_obs~s(x)+x`. We used the s(x)+x to illustrate the nestedness of the model, but the +x can be omitted.

Challenge 1

We will now try this comparison test with some new simulated data, just to get a handle on it.

```
n ← 250
x_test ← runif(n, -5, 5)
y_test_fit ← 4 * dnorm(x_test)
y_test_obs ← rnorm(n, y_test_fit, 0.2)
```

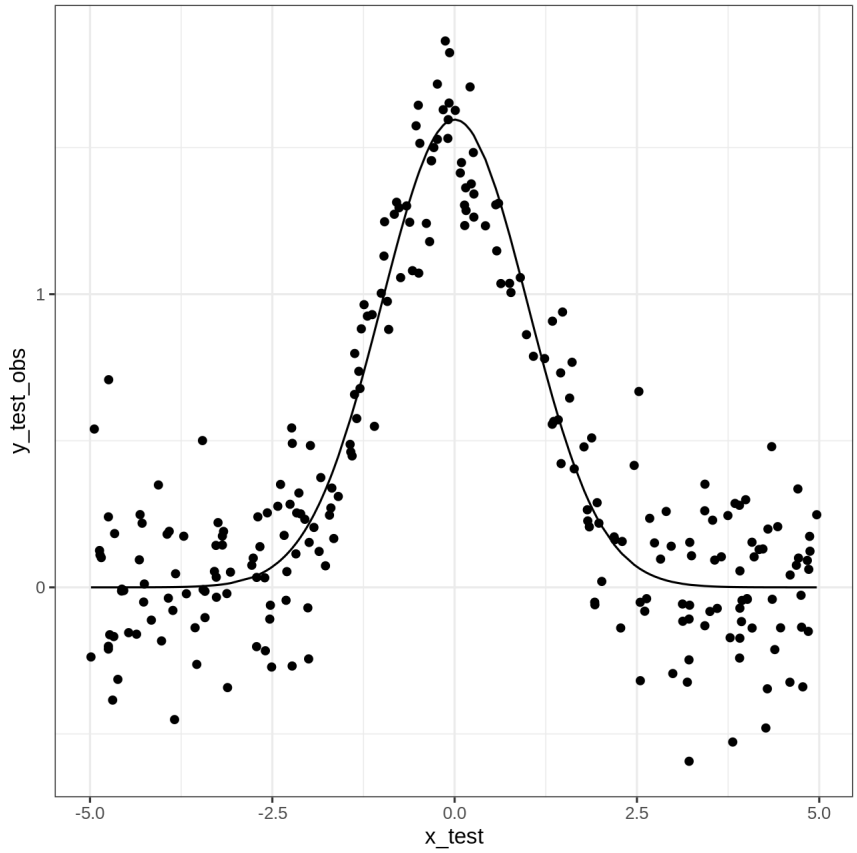
1. Fit a linear and smoothed GAM model to the relation between `x_test` and `y_test_obs`.
2. Determine if linearity is justified for this data.
3. What is the estimated degrees of freedom of the smoothed term?

Challenge 1 - Solution

```
linear_model_test ← gam(y_test_obs ~ x_test)
nested_gam_model_test ← gam(y_test_obs ~ s(x_test) + x_test)

anova(linear_model_test, nested_gam_model_test, test="Chisq")
# Analysis of Deviance Table
#
# Model 1: y_test_obs ~ x_test
# Model 2: y_test_obs ~ s(x_test) + x_test
#   Resid. Df Resid. Dev     Df Deviance  Pr(>Chi)
# 1     248.0     78.995
# 2     240.1     10.420 7.8988   68.574 < 2.2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Challenge 1 - Solution

```
qplot(x_test, y_test_obs) +
  geom_line(aes(y = y_test_fit)) +
  theme_bw()
```



Challenge 1 - Solution

```
nested_gam_model_test
#
# Family: gaussian
# Link function: identity
#
# Formula:
# y_test_obs ~ s(x_test) + x_test
#
# Estimated degrees of freedom:
# 7.51  total = 9.4
#
# GCV score: 0.04500348     rank: 10/11
```

Answer Yes non-linearity is justified. The estimated degrees of freedom (edf) are >> 1 (we'll get back to this soon).

3. Multiple smooth terms

GAM with multiple variables

GAMs make it easy to include both smooth and linear terms, multiple smoothed terms, and smoothed interactions.

For this section, we will use simulated data generated using `mgcv::gamSim()`.

```
# ?gamSim
gam_data ←  gamSim(eg = 5)
# Additive model + factor
head(gam_data)
#           y x0         x1         x2         x3
# 1  4.723147  1 0.02573032 0.70706571 0.69248543
# 2  8.886671  2 0.83272144 0.84997218 0.88974095
# 3 11.196905  3 0.66302652 0.88025265 0.08469529
# 4 10.886068  4 0.11126873 0.80087554 0.15109792
# 5 12.270534  1 0.87969756 0.37692184 0.51467778
# 6  9.020910  2 0.12441532 0.05154493 0.86526950
```

We will try to model the response `y` using the predictors `x0` to `x3`.

GAM with multiple variables

Let's start with a basic model, with one smoothed term (x1) and one categorical predictor (x0, which has 4 levels).

```
basic_model ← gam(y ~ x0 + s(x1), data = gam_data)
basic_summary ← summary(basic_model)
basic_summary$p.table
#             Estimate Std. Error   t value     Pr(>|t|)
# (Intercept) 8.550030  0.3655849 23.387258 1.717989e-76
# x02         2.418682  0.5165515  4.682364 3.908046e-06
# x03         4.486193  0.5156501  8.700072 9.124666e-17
# x04         6.528518  0.5204234 12.544629 1.322632e-30

basic_summary$s.table
#            edf   Ref.df        F      p-value
# s(x1) 1.923913 2.406719 42.43242 1.338683e-19
```

The `p.table` provides the significance table for each linear term

The `s.table` provides the significance table for each smoothed term.

Note on estimated degrees of freedom

```
basic_summary$s.table
#            edf   Ref.df        F      p-value
# s(x1) 1.923913 2.406719 42.43242 1.338683e-19
```

The `edf` shown in the `s.table` is the estimated degrees of freedom – essentially, a larger edf value implies more complex wiggly splines.

- A value close to 1 tend to be close to a linear term.
- A high value (8–10 or higher) means that the spline is highly non-linear.

> In our basic model the edf of smooth function s(x1) is ~2, which suggests a non-linear curve.

Note on estimated degrees of freedom

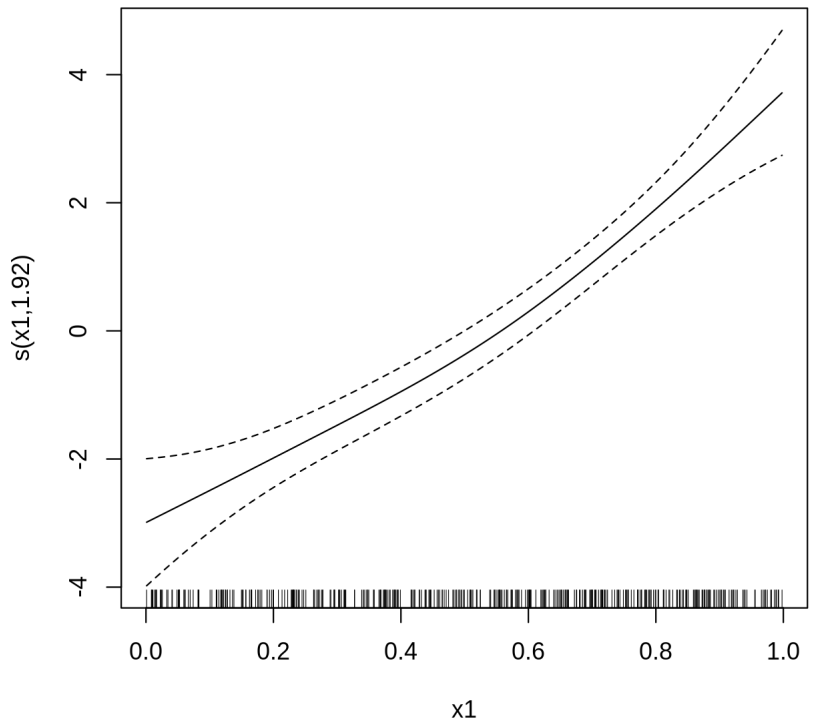
The edf in GAM is different from the degrees of freedom in a linear regression.

In linear regression, the *model* degrees of freedom is equivalent to the number of non-redundant free parameters, p, in the model (and the *residual* degrees of freedom are given by n-p).

We will revisit the edf later in this workshop.

GAM with multiple variables

```
plot(basic_model)
```



GAM with multiple variables

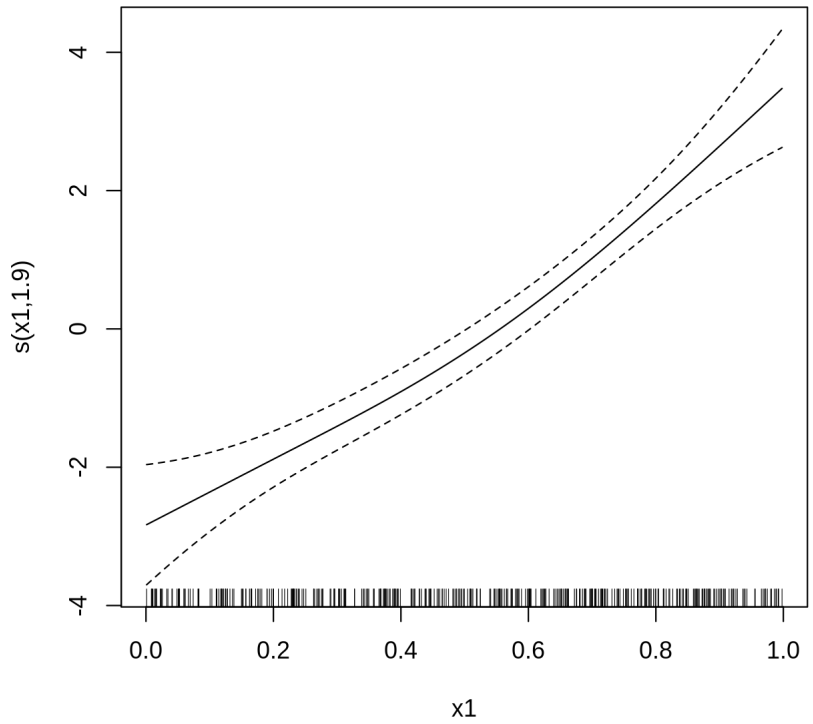
We can add a second term, `x2`, but specify a linear relationship with `y`

```
two_term_model ← gam(y ~ x0 + s(x1) + x2, data = gam_data)
two_term_summary ← summary(two_term_model)
two_term_summary$p.table
#              Estimate Std. Error    t value     Pr(>|t|)
# (Intercept) 11.400658  0.4177614  27.289879 9.396207e-93
# x02          2.314405  0.4552138   5.084216 5.723467e-07
# x03          4.487653  0.4543299   9.877520 1.063008e-20
# x04          6.596149  0.4585778  14.383925 5.468771e-38
# x2          -5.825948  0.5436671 -10.716021 1.114046e-23

two_term_summary$s.table
#            edf   Ref.df        F      p-value
# s(x1) 1.900864 2.377544 49.85908 2.287393e-22
```

GAM with multiple variables

We can add a second term, `x2`, but specify a linear relationship with `y`

```
plot(two_term_model)
```



GAM with multiple variables

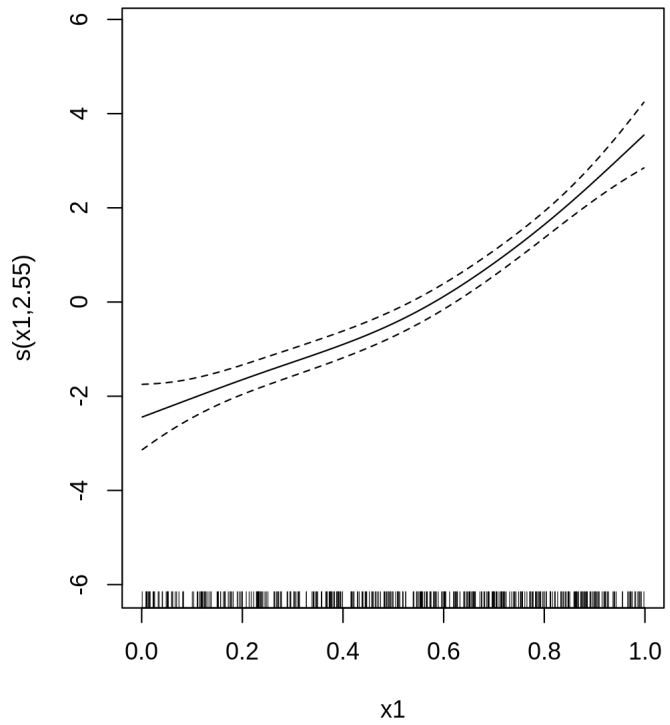
We can also explore whether the relationship between `y` and `x2` is non-linear

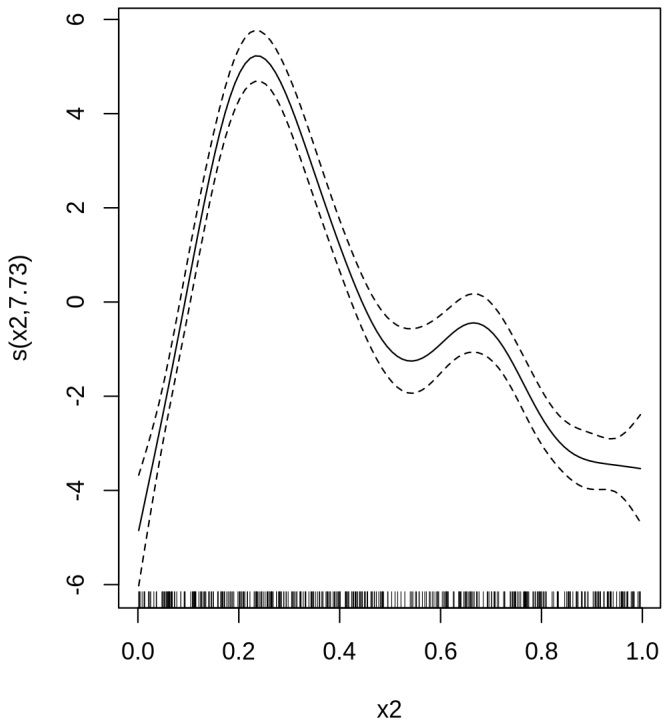
```
two_smooth_model ← gam(y ~ x0 + s(x1) + s(x2), data = gam_data)
two_smooth_summary ← summary(two_smooth_model)
two_smooth_summary$p.table
#             Estimate Std. Error   t value      Pr(>|t|)
# (Intercept) 8.937862  0.2217506 40.305927 2.373755e-140
# x02         2.008045  0.3137690  6.399756  4.518133e-10
# x03         3.832496  0.3143049 12.193562  3.758930e-29
# x04         6.041521  0.3145299 19.208098  3.520507e-58

two_smooth_summary$s.table
#            edf   Ref.df        F       p-value
# s(x1) 2.546757 3.175726 68.10051  9.199287e-40
# s(x2) 7.726989 8.582003 81.55441 2.326028e-120
```

GAM with multiple variables

We can also explore whether the relationship between `y` and `x2` is non-linear

```
plot(two_smooth_model, page = 1)
```





GAM with multiple variables

As before, we can perform an ANOVA to test if the smoothed term is necessary

```
anova(basic_model, two_term_model, two_smooth_model, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y ~ x0 + s(x1)
# Model 2: y ~ x0 + s(x1) + x2
# Model 3: y ~ x0 + s(x1) + s(x2)
#   Resid. Df Resid. Dev      Df Deviance  Pr(>Chi)
# 1    393.59     5231.6
# 2    392.62     4051.3 0.97082   1180.2 < 2.2e-16 ***
# 3    384.24     1839.5 8.38019   2211.8 < 2.2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The best fit model is the model with both smooth terms for `x1` and `x2`

Challenge 2

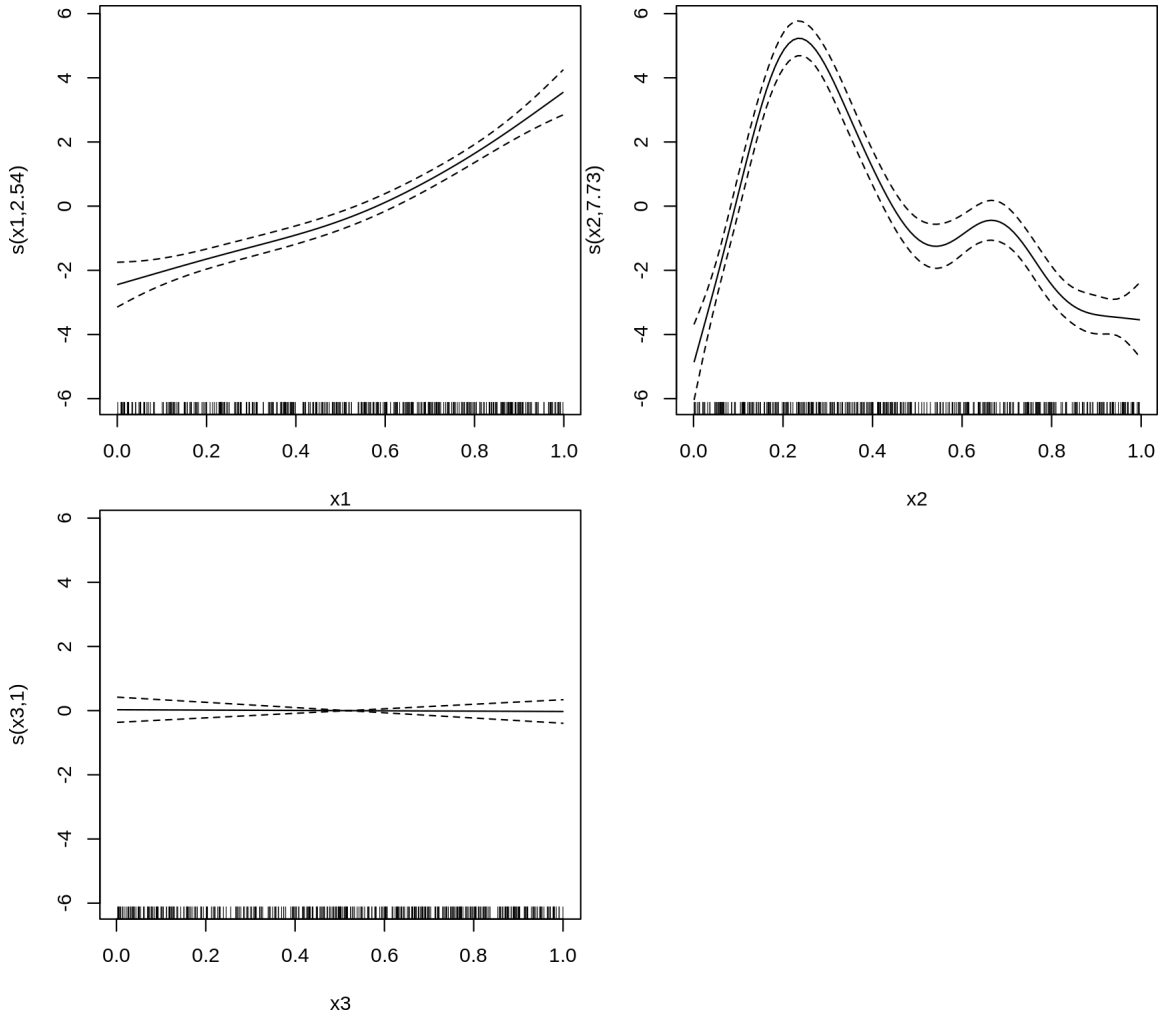
1. Create 2 new models, with `x3` as a linear and smoothed term.
2. Determine if `x3` is an important term to include using plots, coefficient tables and the anova function.

Challenge 2 - Solution

```
three_term_model ← gam(y ~ x0 + s(x1) + s(x2) + x3, data = gam_data)
three_smooth_model ← gam(y~x0 + s(x1) + s(x2) + s(x3), data = gam_data)
three_smooth_summary ← summary(three_smooth_model)
```

Challenge 2 - Solution

```
plot(three_smooth_model, page = 1)
```



Challenge 2 - Solution

```
three_smooth_summary$s.table
#            edf   Ref.df           F       p-value
# s(x1) 2.542296 3.170441 67.75922314  1.577435e-39
# s(x2) 7.731424 8.584355 80.90188472 2.006819e-119
# s(x3) 1.000000 1.000000  0.02039697  8.865087e-01

# edf = 1 therefore term is linear.

anova(two_smooth_model, three_term_model, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y ~ x0 + s(x1) + s(x2)
# Model 2: y ~ x0 + s(x1) + s(x2) + x3
#   Resid. Df Resid. Dev      Df Deviance Pr(>Chi)
# 1    384.24     1839.5
# 2    383.25     1839.3 0.99707  0.18818   0.8418

# term x3 is not significant, it should be dropped!
```

4. Interactions

GAM with interaction terms

There are 2 ways to include interactions between variables:

- for 2 smoothed variables : `s(x1, x2)`
- for one smoothed variable and one linear variable (either factor or continuous): use the `by` argument `s(x1, by = x2)`
  - When `x2` is a factor, you have a smooth term that vary between different levels of `x2`
  - When `x2` is continuous, the linear effect of `x2` varies smoothly with `x1`
  - When `x2` is a factor, the factor needs to be added as a main effect in the model

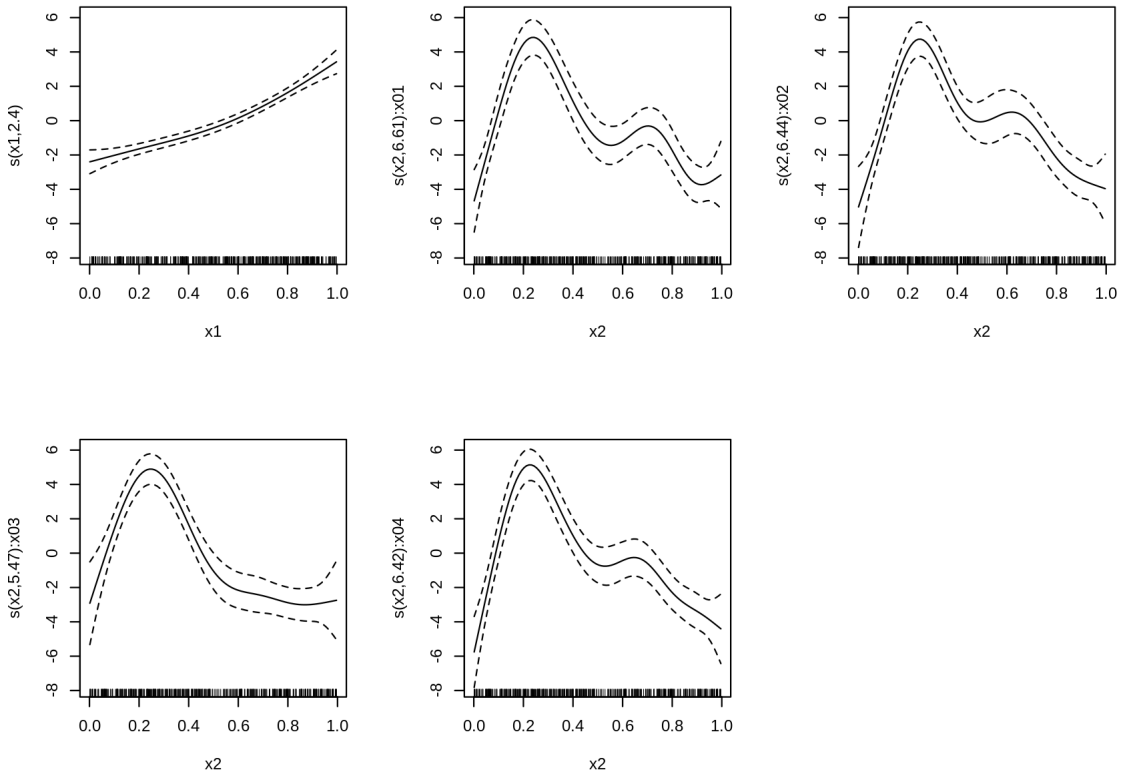
GAM with interaction terms

We will examine interaction effect using our categorical variable `x0` and ask whether the non-linear smoother `s(x2)` varies across different levels of `x0`.

```
factor_interact ← gam(y ~ x0 + s(x1) + s(x2, by = x0), data = gam_data)

summary(factor_interact)$s.table
#                edf   Ref.df        F      p-value
# s(x1)     2.401350 2.996664 67.18567 6.863193e-38
# s(x2):x01 6.606775 7.708292 21.40363 3.077275e-27
# s(x2):x02 6.436261 7.571201 19.96801 4.916484e-25
# s(x2):x03 5.467172 6.618716 28.75426 2.871338e-32
# s(x2):x04 6.422564 7.574388 26.43680 2.075813e-33
```

GAM with interaction terms

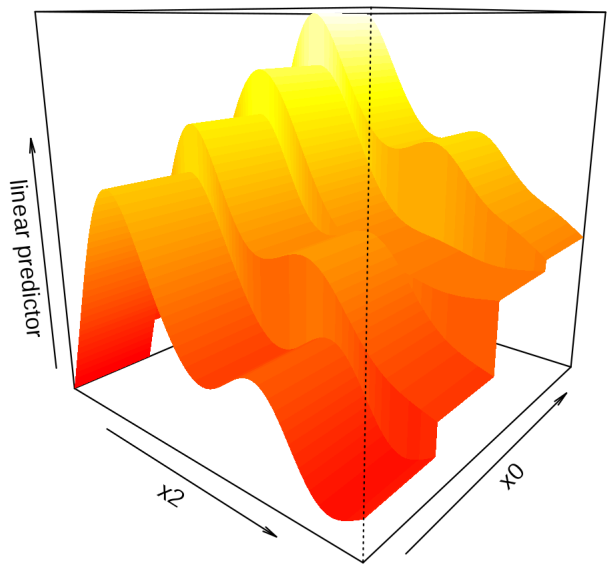
```
plot(factor_interact, page = 1)
```



GAM with interaction terms

We can also visualise our model in 3D using `vis.gam`, where `theta` is the degree rotation on the x-y plane

```
vis.gam(factor_interact, view = c("x2","x0"), theta = 40, n.grid = 500, border =
```



GAM with interaction terms

Let's perform a model comparison using ANOVA to determine if the interaction term is necessary

```
anova(two_smooth_model, factor_interact, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y ~ x0 + s(x1) + s(x2)
# Model 2: y ~ x0 + s(x1) + s(x2, by = x0)
#   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
# 1    384.24     1839.5
# 2    363.53     1740.9 20.712   98.608   0.4482
```

From the plots, we saw that the shape of the smooth terms were comparable among the 4 levels of `x0`. The anova test confirms this as well ($p > 0.05$).

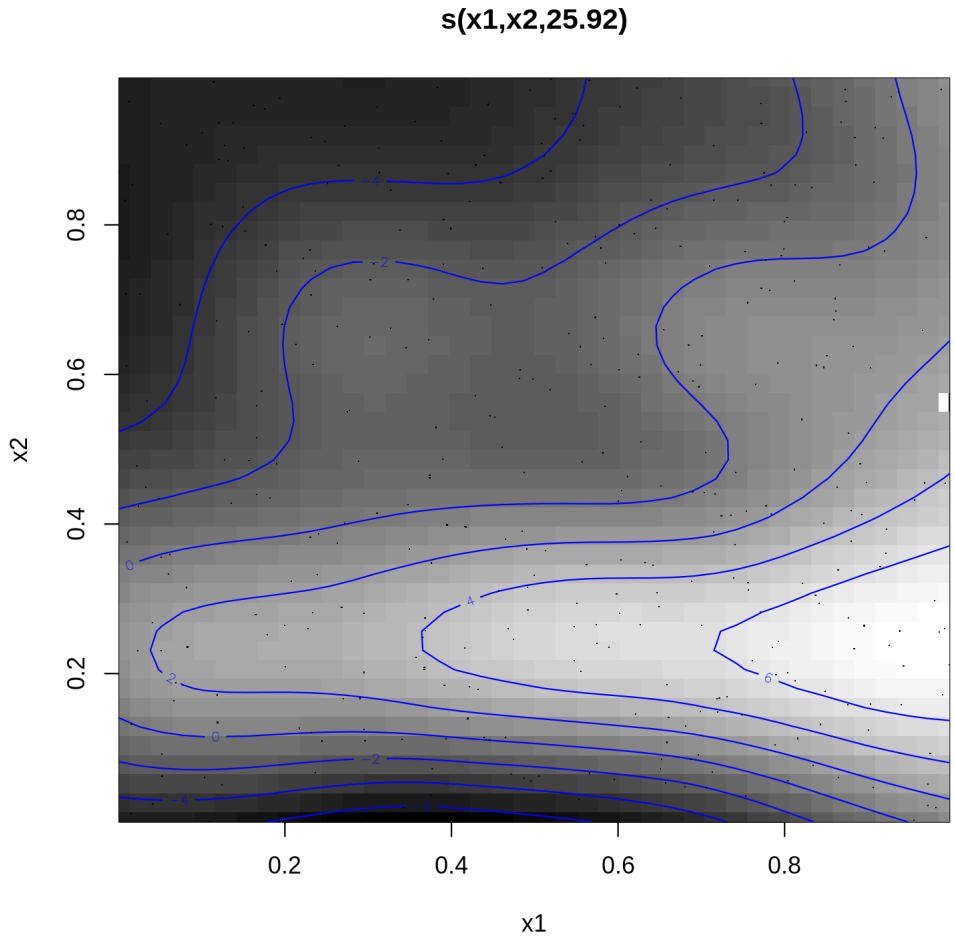
GAM with interaction terms

Finally we'll look at the interactions between 2 smoothed terms, `x1` and `x2`.

```
smooth_interact ← gam(y~x0 + s(x1, x2), data = gam_data)
summary(smooth_interact)$s.table
#                edf   Ref.df        F       p-value
# s(x1,x2) 25.91803 28.42892 36.40735 8.635007e-165
```

GAM with interaction terms

```
plot(smooth_interact, page = 1, scheme = 3)
```



GAM with interaction terms

```
vis.gam(smooth_interact, view = c("x1", "x2"), theta=40, n.grid = 500, border = N
```

GAM with interaction terms

```
anova(two_smooth_model, smooth_interact, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y ~ x0 + s(x1) + s(x2)
# Model 2: y ~ x0 + s(x1, x2)
#   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
# 1    384.24     1839.5
# 2    367.57     1710.3 16.671    129.2  0.04063 *
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The interaction between `s(x1)` and `s(x2)` is significant and the 2D plot nicely illustrates this non-linear interactions, where `y` is large at high values of `x1` but low to mid-values of `x2`.

5. Changing basis

Expanding on the basic GAM

It is possible to expand further on the basic GAM model with:

1. more complicated smooths, by changing the basis,
2. other distributions: anything you can do with a GLM using the family argument,
3. mixed effect models, using `gamm` or the `gamm4` package.

We will now go over this 3 expansions.

Other smooth functions

To model a non-linear smooth variable or surface, 3 different smooth functions are available:

`s()` ➔ for modeling a 1-dimensional smooth or for modeling interactions among variables measured using the same unit and the same scale

`te()` ➔ for modeling 2- or n-dimensional interaction surfaces of variables that are not on the same scale. Includes main effects.

`ti()` ➔ for modeling 2- or n-dimensional interaction surfaces that do not include the main effects.

Parameters of smooth functions

The smooth functions have several parameters that can be set to change their behavior. The most often-used parameters are :

`k` ➔ number of 'knots'

- determines the upper bound of the number of base functions used to build the curve.
- constrains the wigglyness of a smooth.
- the number of base functions is reflected in the edf value.
- the default k for `s()` is ~9, and for `te()` and `ti()` is 5 per dimension.
- k should be < the number of unique data points.

Parameters of smooth functions

The smooth functions have several parameters that can be set to change their behavior. The most often-used parameters are :

`d` ➔ specifies that predictors in the interaction are on the same scale or dimension (only used in `te()` and `ti()`).

- For example, in `te(Time, width, height, d=c(1,2))`, indicates that `width` and `height` are one the same scale, but not `Time`.

`bs` ➔ specifies the type of underlying base functions.

- the default for `s()` is `tp` (thin plate regression spline) and for `te()` and `ti()` is `cr` (cubic regression spline).

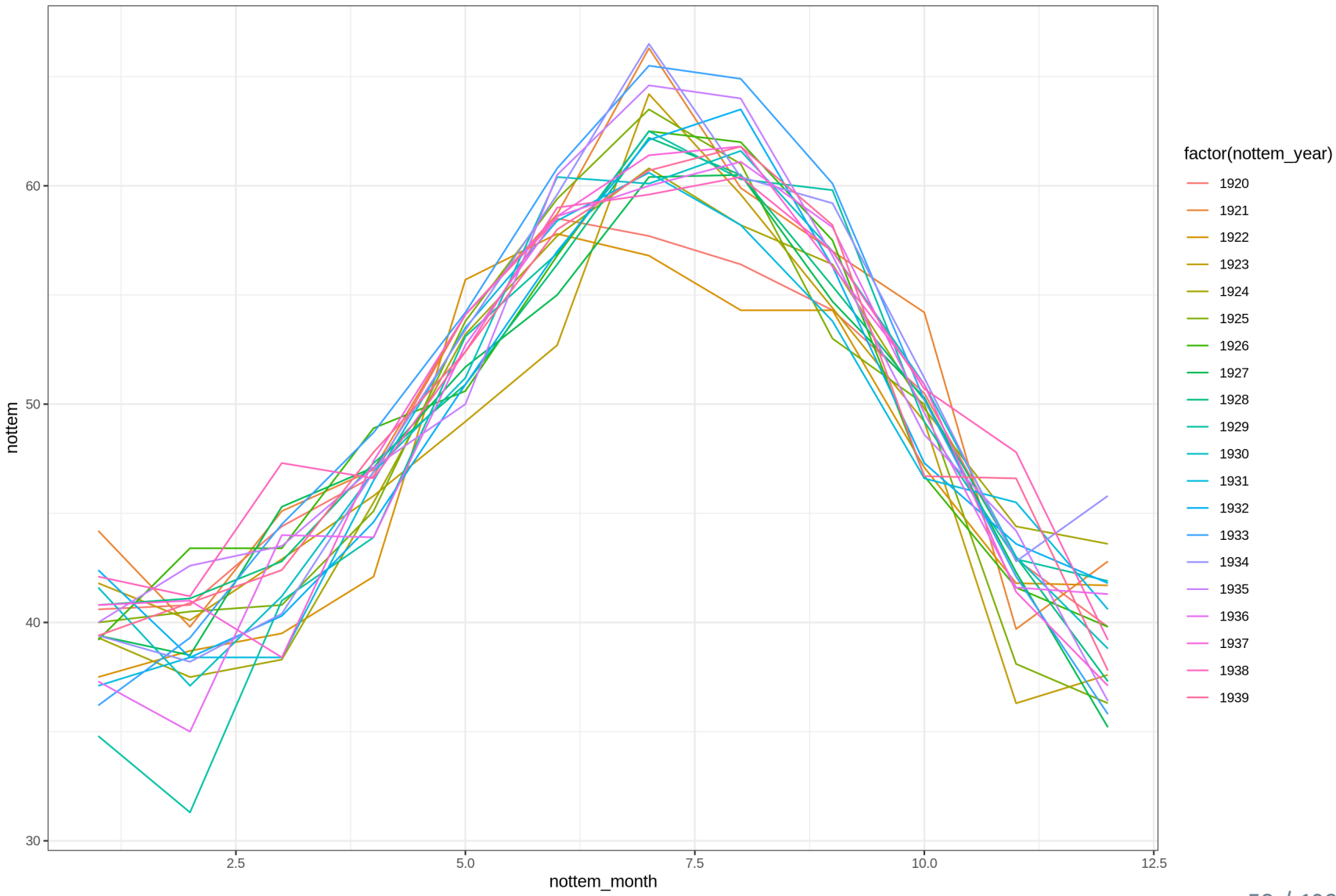
Example smooth for cyclical data

Cyclical data is a good example where changing basis is useful: you want the predictor to match at the ends.

Let's use a time series of climate data, with monthly measurements, and see if there's a temporal trend in yearly temperature.

```
data(nottem) # Nottingham temperature time series
n_years ← length(nottem)/12
nottem_month ← rep(1:12, times = n_years)
nottem_year ← rep(1920:(1920 + n_years - 1), each = 12)
qplot(nottem_month, nottem, colour = factor(nottem_year), geom = "line") +
  theme_bw()
```

Example smooth for cyclical data



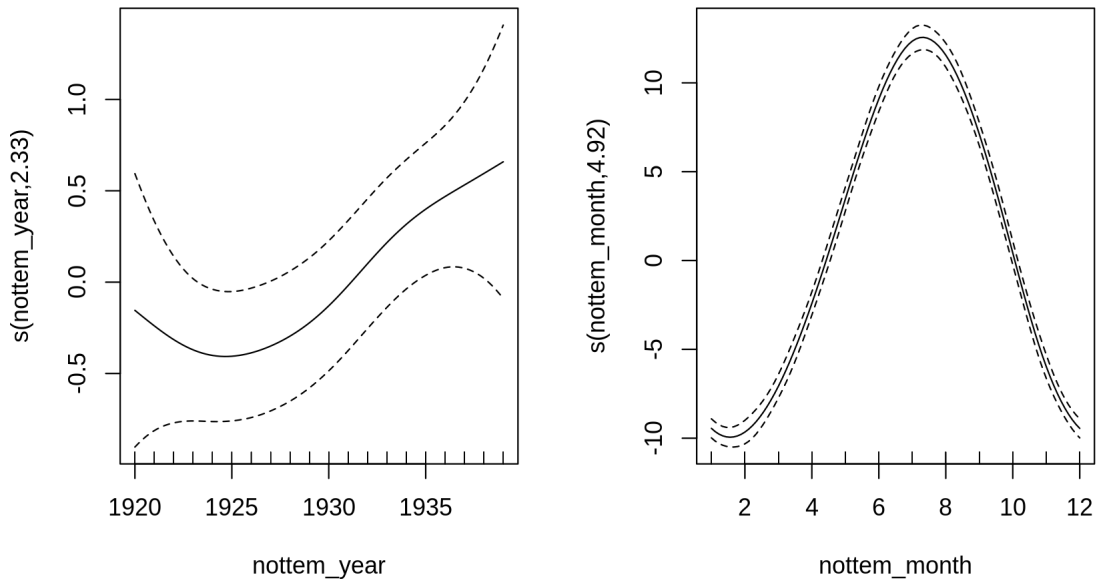
Example smooth for cyclical data

We can model both the cyclic change of temperature across months and the non-linear trend through years, using a cyclical cubic spline, or `cc`, for the month variable and a regular smooth for the year variable.

```
year_gam ← gam(nottem ~ s(nottem_year) + s(nottem_month, bs = "cc"))
summary(year_gam)$s.table
#                      edf   Ref.df           F    p-value
# s(nottem_year)  2.333879 2.906998   2.528043 0.07266502
# s(nottem_month) 4.923943 8.000000 390.029032 0.00000000
```

Example smooth for cyclical data

```
plot(year_gam, page = 1, scale = 0)
```



There is about 1-1.5 degree rise in temperature over the period, but within a given year there is about 20 degrees variation in temperature, on average. The actual data vary around these values and that is the unexplained variance.

6. Other distributions

GAM using other distributions

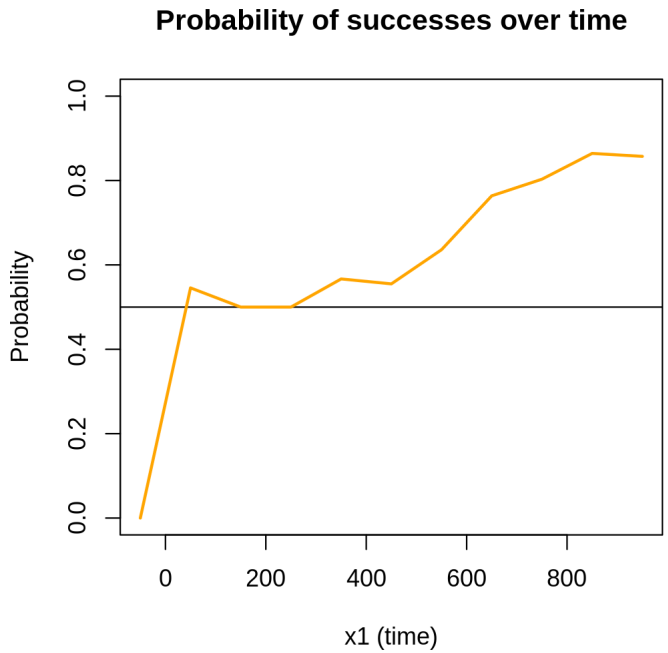
Let's now take a look on how to use GAMs when the response variable does not follow a normal distributions and is either count or proportion data (e.g., Gamma, binomial, Poisson, negative binomial).

We will use an example dataset where a binomial distribution is needed; the response variable represents the number of successes vs failures over the course of an experiment.

```
gam_data3 ← read.csv("data/other_dist.csv")
str(gam_data3)
# 'data.frame':    514 obs. of  4 variables:
#  $ prop : num  1 1 1 1 0 1 1 1 1 1 ...
#  $ total: int  4 20 20 18 18 18 20 20 20 20 ...
#  $ x1   : int  550 650 750 850 950 650 750 850 950 550 ...
#  $ fac  : Factor w/ 4 levels "f1","f2","f3",..: 1 1 1 1 1 1 1 1 1 1 ...
```

GAM using other distributions

```
plot(range(gam_data3$x1), c(0,1), type = "n",
     main = "Probability of successes over time",
     ylab = "Probability", xlab = "x1 (time)")
abline(h = 0.5)

avg ← aggregate(prop ~ x1, data=gam_data3, mean)
lines(avg$x1, avg$prop, col = "orange", lwd = 2)
```



GAM using other distributions

We will test if this trend is linear or not using a logistic GAM (we use a binomial family distribution given that our response is proportion data).

```
prop_model ← gam(prop ~ s(x1), data = gam_data3, weights = total, family = "binc
prop_summary ← summary(prop_model)
prop_summary$p.table
#             Estimate Std. Error  z value Pr(>|z|)
# (Intercept) 1.173978 0.02709613 43.32641        0
prop_summary$s.table
#            edf   Ref.df   Chi.sq       p-value
# s(x1) 4.591542 5.615235 798.9407 1.677701e-164
```

What does the intercept represent in this model?

What does the smooth term indicate?

GAM using other distributions

```
#             Estimate Std. Error  z value Pr(>|z|)
# (Intercept) 1.173978 0.02709613 43.32641        0
```

What does the intercept represent in this model?

Recall that the model uses the count data to calculate the logit, which is the log odds ratio between successes and failures:

- If successes = failures, the ratio = 1 and the logit is 0 (log(1) = 0).
- If successes > failures, the ratio > 1 and the logit has a positive value (log(2) = 0.69).
- If successes < failures, the ratio < 1 and the logit has a negative value (log(.5) = -0.69).

> Here, the estimated intercept coefficient is positive, which means that there are more successes than failures overall.

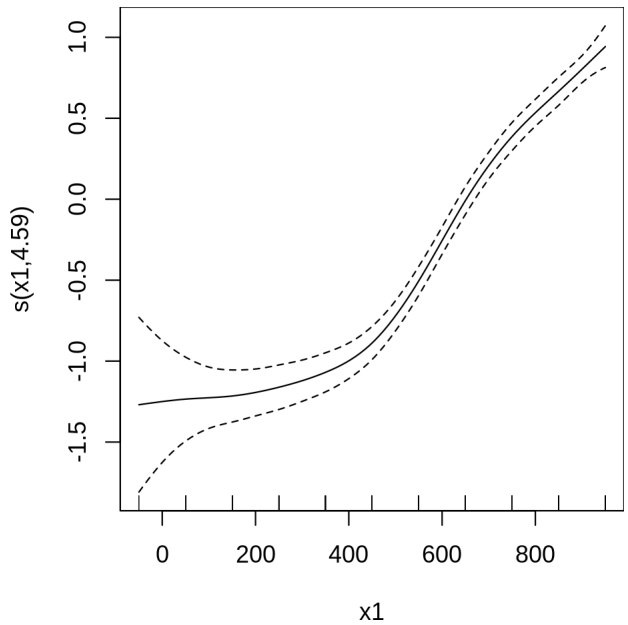
GAM using other distributions

```
#             edf   Ref.df   Chi.sq       p-value
# s(x1) 4.591542 5.615235 798.9407 1.677701e-164
```

What does the smooth term indicate?

This represents how the log odds of successes vs failures changes over time (x1).

> As the edf > 1, the proportion of successes increases faster over time



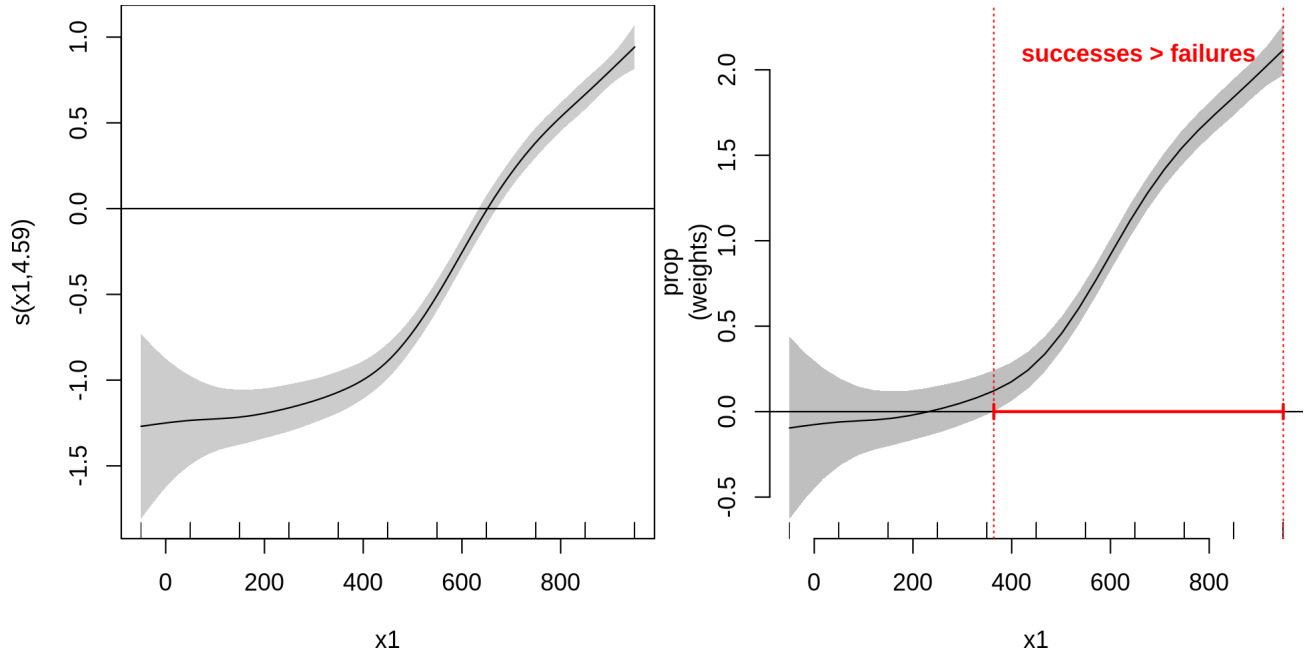
Visualizing the trend over time

There are different ways this relationship can be represented graphically:

- **partial effects** are the isolated effects of one particular predictor or interaction. If you visualize your GAM model with `plot()`, you get the partial effects.
- **summed effects** are the predicted response measures for a given value or level of predictors. If you visualize your GAM model with `itsadug::plot_smooth()`, you get the summed effects.

Visualizing the trend over time

What do these plots tell us about successes vs failures?



Contribution / partial effect

Over time the log odds increases, so over time successes increase and failures decrease.

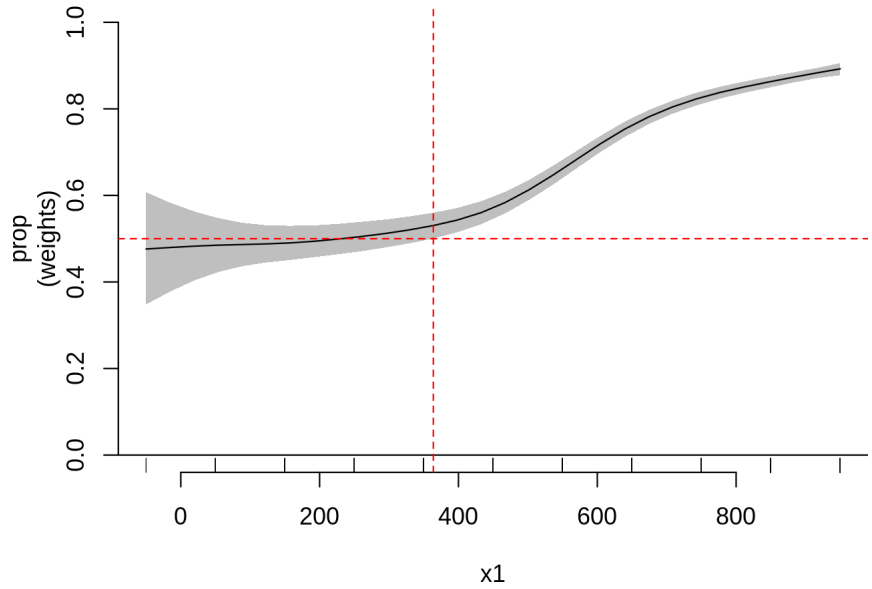
Fitted values, summed effect, intercept included

Equal amounts of successes and failures up to x1=400.

Visualizing the trend over time

Lastly, to help interpret the results, we could transform the summed effects back to proportions with the function `itsadug::plot_smooth()`:

```
plot_smooth(prop_model, view = "x1", main = "",
            transform = plogis, ylim = c(0,1), print.summary = F)
abline(h = 0.5, v = diff$start, col = 'red', lty = 2)
```



As in the logit plot, the proportion of successes increases above 0.5 at x1=400.

7. Quick intro to GAMM

Dealing with non-independence

When observations are not independent, GAMs can be used to either incorporate:

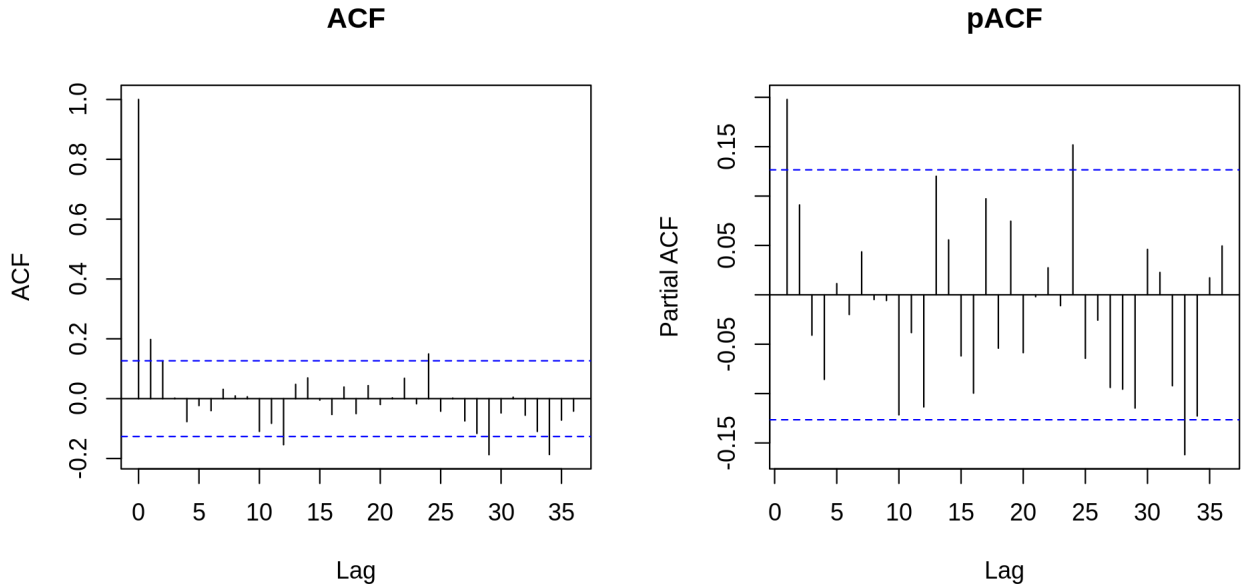
- a serial correlation structure to model residual autocorrelation (autoregressive AR, moving average MA, or a combination of the two ARMA),
- random effects that model independence among observations from the same site.

Model with correlated errors

Let's have a look at a model with temporal autocorrelation in the residuals. We will revisit the Nottingham temperature model and test for correlated errors using the (partial) autocorrelation function.

```
par(mfrow = c(1,2))
acf(resid(year_gam), lag.max = 36, main = "ACF")
pacf(resid(year_gam), lag.max = 36, main = "pACF")
```

Model with correlated errors



ACF (and pACF) provide the cross correlation (and partial correlation) of a time series with itself at different time lags, and are used to identify after how many time steps observations start to be independent.

The ACF plot of our model residuals suggests a significant lag of 1, and perhaps a lag of 2. Therefore, a low-order AR model is likely needed.

Model with correlated errors

We can test for autocorrelation by adding AR structures to the model: AR(1) (correlation at 1 time step) and AR(2) (correlation at 2 time steps).

```
year_gam ← gamm(nottem ~ s(nottem_year) + s(nottem_month, bs = "cc"))
year_gam_AR1 ← gamm(nottem ~ s(nottem_year) + s(nottem_month, bs = "cc"),
                    correlation = corARMA(form = ~ 1|nottem_year, p = 1),
                  data = data.frame(nottem, nottem_year, nottem_month))
year_gam_AR2 ← gamm(nottem ~ s(nottem_year) + s(nottem_month, bs = "cc"),
                    correlation = corARMA(form = ~ 1|nottem_year, p = 2),
                  data = data.frame(nottem, nottem_year, nottem_month))
anova(year_gam$lme, year_gam_AR1$lme, year_gam_AR2$lme)
#                  Model df      AIC      BIC    logLik   Test  L.Ratio p-value
# year_gam$lme         1  5 1109.908 1127.311 -549.9538
# year_gam_AR1$lme     2  6 1101.218 1122.102 -544.6092 1 vs 2 10.68921  0.0011
# year_gam_AR2$lme     3  7 1101.598 1125.962 -543.7988 2 vs 3  1.62082  0.2030
```

AR(1) provides a significant increase in fit over the naive model (LRT = 10.69, p = 0.0011), but little improvement with AR(2) (LRT = 1.62, p = 0.203).

Mixed modelling

As we saw in the section changing basis, `bs` specifies the type of underlying base function. For random intercepts and linear random slopes we use `bs = "re"`, but for random smooths we use `bs = "fs"`.

3 different types of random effects in GAMMs (`fac` ➔ factor coding for the random effect; `x0` ➔ continuous fixed effect):

- **random intercepts** adjust the height of other model terms with a constant value: `s(fac, bs="re")`
- **random slopes** adjust the slope of the trend of a numeric predictor: `s(fac, x0, bs="re")`
- **random smooths** adjust the trend of a numeric predictor in a nonlinear way: `s(x0, fac, bs="fs", m=1)`, where the argument m=1 sets a heavier penalty for the smooth moving away from 0, causing shrinkage to the mean.

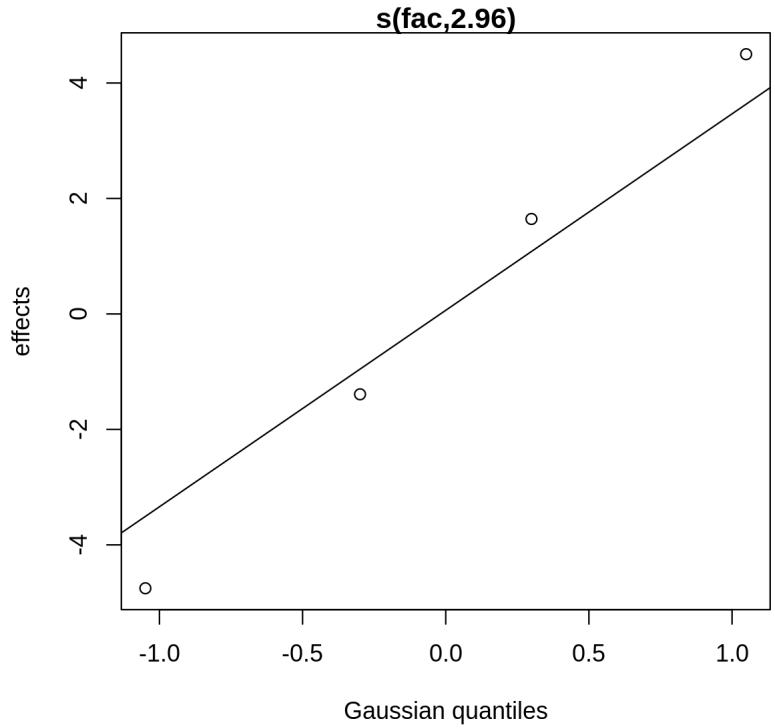
GAMM with a random intercept

As before, we will use the `gamSim()` function to generate a dataset, here with a random effect, then run a model with a random intercept using `fac` as the random factor.

```
gam_data2 ← gamSim(eg = 6)
# 4 term additive + random effectGu & Wahba 4 term additive model
str(gam_data2)
# 'data.frame':    400 obs. of  11 variables:
#  $ y  : num  8.72 10.75 15.3 16.79 11.55 ...
#  $ x0 : num  0.277 0.698 0.379 0.869 0.568 ...
#  $ x1 : num  0.407 0.3657 0.235 0.4764 0.0666 ...
#  $ x2 : num  0.9221 0.0491 0.0245 0.9052 0.6511 ...
#  $ x3 : num  0.0254 0.6572 0.2899 0.0727 0.8257 ...
#  $ f  : num  6.8 10.42 12.57 15.44 9.39 ...
#  $ f0 : num  1.528 1.625 1.857 0.799 1.955 ...
#  $ f1 : num  2.26 2.08 1.6 2.59 1.14 ...
#  $ f2 : num  0.0183 0.716 0.1143 0.0486 3.2902 ...
#  $ f3 : num  0 0 0 0 0 0 0 0 0 0 ...
#  $ fac: Factor w/ 4 levels "1","2","3","4": 1 2 3 4 1 2 3 4 1 2 ...
```

GAMM with a random intercept

```
gamm_intercept ← gam(y ~ s(x0) + s(fac, bs = "re"), data = gam_data2)
summary(gamm_intercept)$s.table
#             edf   Ref.df         F      p-value
# s(x0)  3.044725 3.776643  2.481658 3.774254e-02
# s(fac) 2.960269 3.000000 95.146996 2.696335e-54
plot(gamm_intercept, select = 2)
```

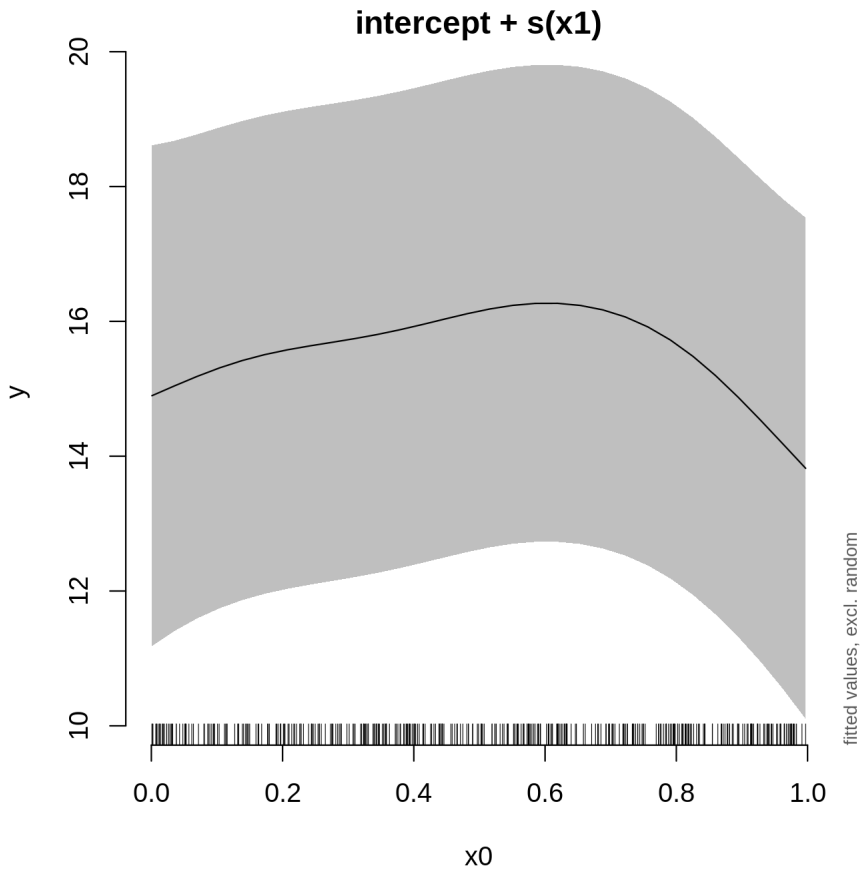


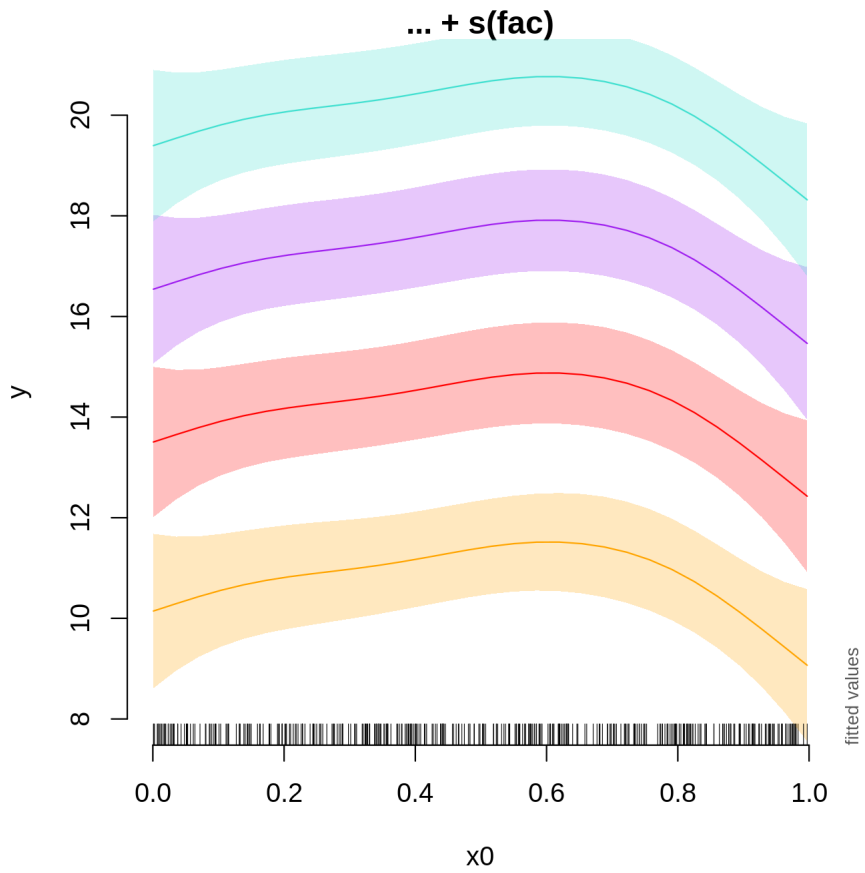
GAMM with a random intercept

We can plot the summed effects for the `x0` without random effects, and then plot the predictions of all 4 levels of the random `fac` effect:

```
par(mfrow = c(1,2), cex = 1.1)

plot_smooth(gamm_intercept, view = "x0", rm.ranef = T,
            main = "intercept + s(x1)")

plot_smooth(gamm_intercept, view = "x0", cond = list(fac="1"),
            main = " ... + s(fac)", col = 'orange', ylim = c(8,21))

plot_smooth(gamm_intercept, view = "x0", cond = list(fac = "2"), add = T, col = '

plot_smooth(gamm_intercept, view="x0", cond = list(fac = "3"), add = T, col = 'pu

plot_smooth(gamm_intercept, view="x0", cond = list(fac = "4"), add = T, col = 'tu
```

GAMM with a random intercept





fac1 fac2 fac3 fac4

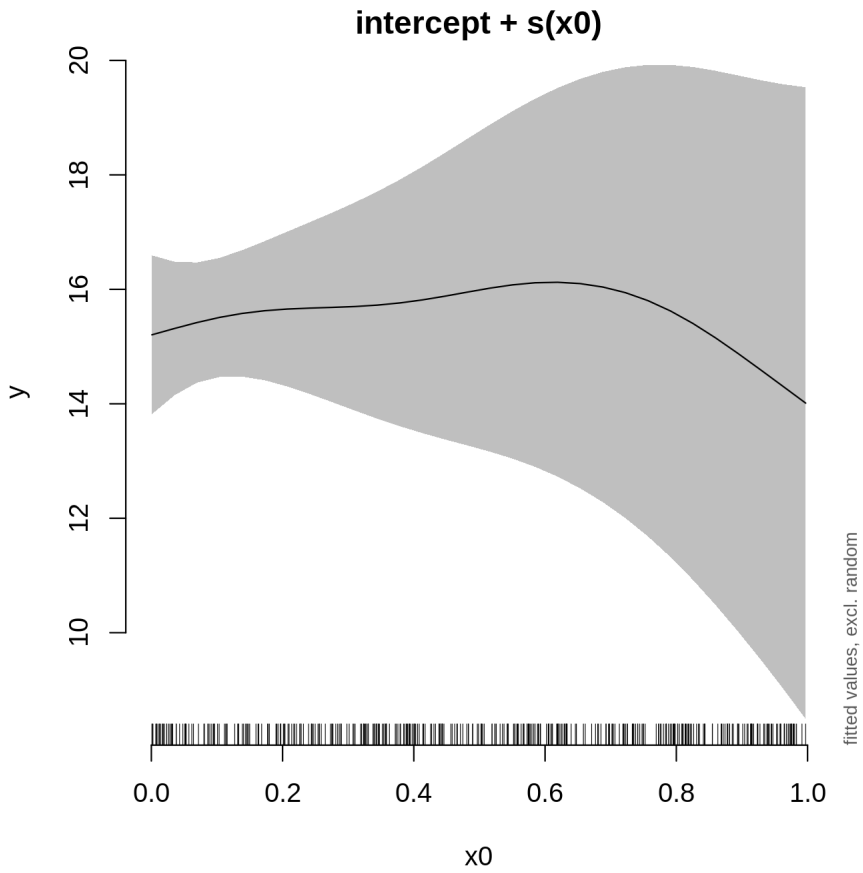
GAMM with a random slope

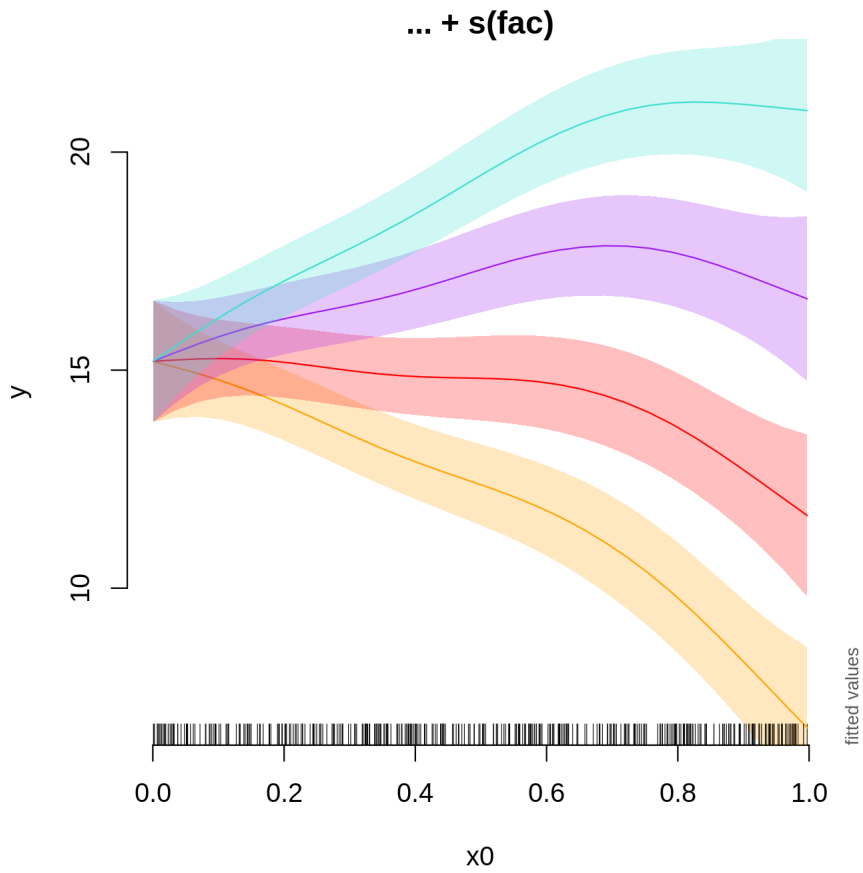
```
gamm_slope ← gam(y ~ s(x0) + s(x0, fac, bs = "re"), data = gam_data2)

summary(gamm_slope)$s.table
#                edf   Ref.df         F      p-value
# s(x0)     2.961019 3.673378  1.444919 1.804908e-01
# s(x0,fac) 2.946695 3.000000 72.392941 7.346091e-42
```

GAMM with a random slope

```
par(mfrow = c(1,2), cex = 1.1)

plot_smooth(gamm_slope, view = "x0", rm.ranef = TRUE, main = "intercept + s(x0)")

plot_smooth(gamm_slope, view = "x0", cond = list(fac = "1"),
            main = " ... + s(fac)", col = 'orange', ylim = c(7,22))

plot_smooth(gamm_slope, view = "x0", cond = list(fac = "2"), add = T, col = 'red'

plot_smooth(gamm_slope, view = "x0", cond = list(fac = "3"), add = T, col = 'purp

plot_smooth(gamm_slope, view = "x0", cond = list(fac = "4"), add = T, col = 'turq
```

GAMM with a random slope





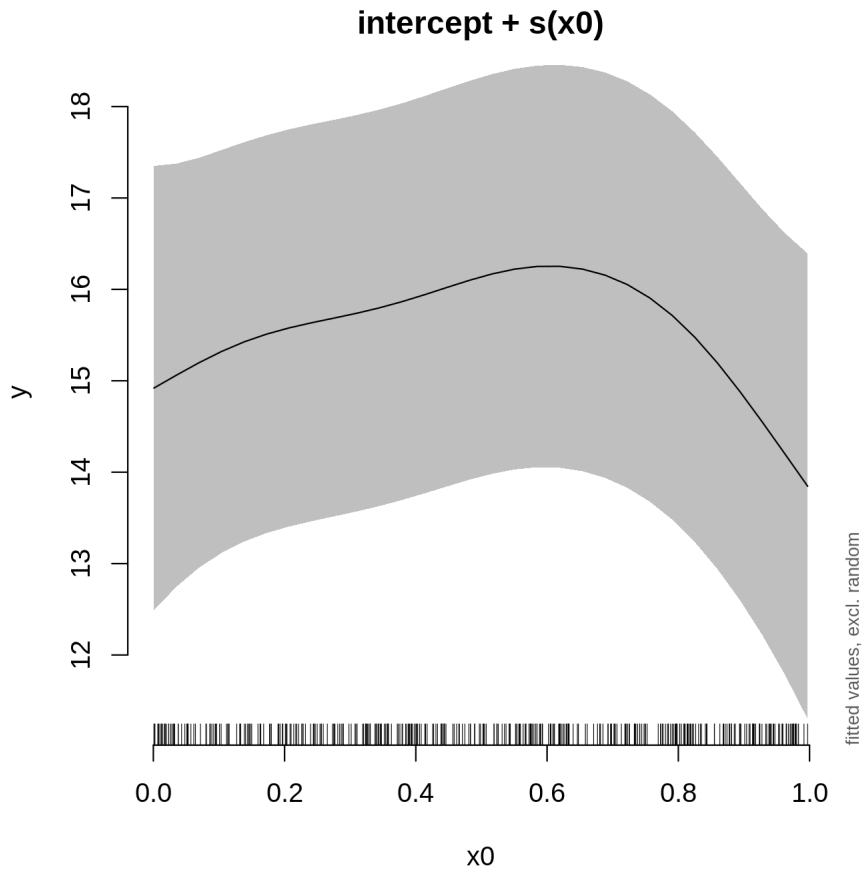
GAMM with a random intercept and slope

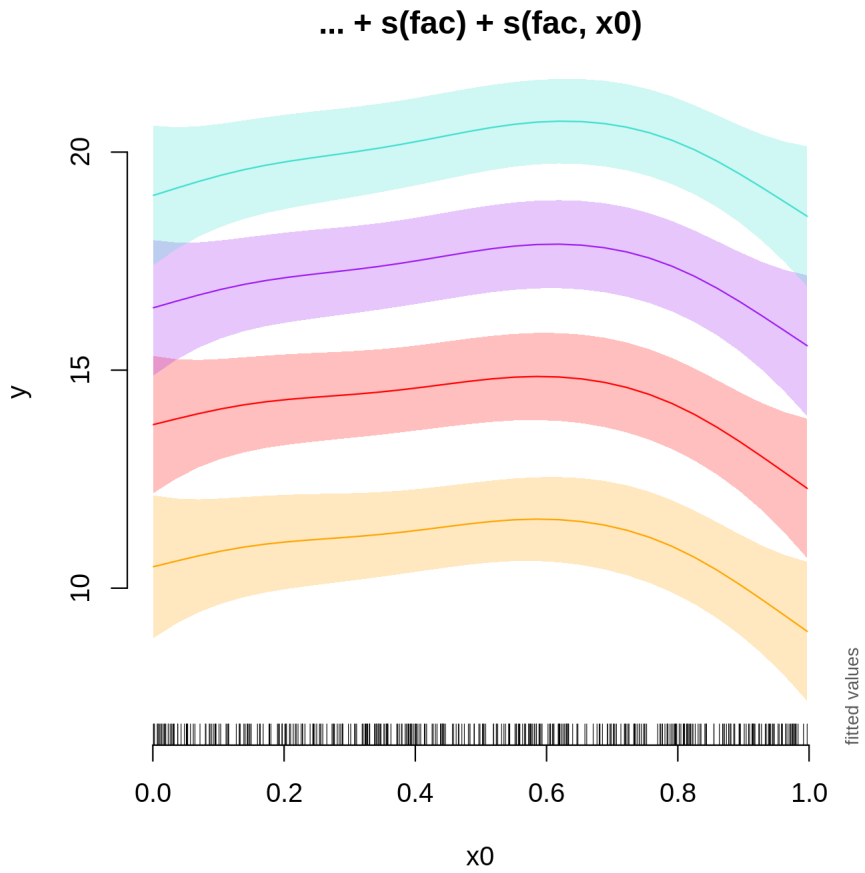
```
gamm_int_slope ← gam(y ~ s(x0) + s(fac, bs = "re") + s(fac, x0, bs = "re"),
                     data = gam_data2)

summary(gamm_int_slope)$s.table
#                 edf   Ref.df           F      p-value
# s(x0)     3.0121856 3.735986   2.399772 4.299127e-02
# s(fac)    2.8151182 3.000000 154.712380 7.093989e-31
# s(fac,x0) 0.7259132 3.000000   9.273901 7.964598e-02
```

GAMM with a random intercept and slope

```
par(mfrow = c(1,2), cex = 1.1)

plot_smooth(gamm_int_slope, view = "x0", rm.ranef = T, main = "intercept + s(x0)"

plot_smooth(gamm_int_slope, view = "x0", cond = list(fac = "1"),
            main="... + s(fac) + s(fac, x0)", col = 'orange', ylim = c(7,22))

plot_smooth(gamm_int_slope, view = "x0", cond = list(fac = "2"), add = T, col='re

plot_smooth(gamm_int_slope, view = "x0", cond = list(fac = "3"), add = T, col = '

plot_smooth(gamm_int_slope, view = "x0", cond = list(fac = "4"), add = T, col = '
```

GAMM with a random intercept and slope

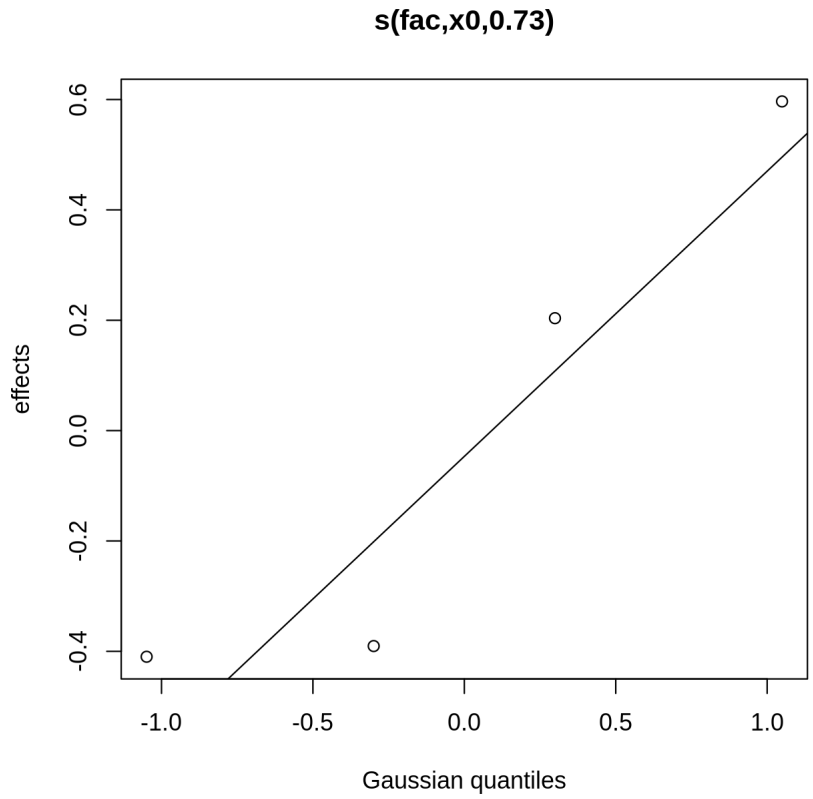




GAMM with a random intercept and slope

Note that the random slope is static in this case:

```
plot(gamm_int_slope, select = 3)
```



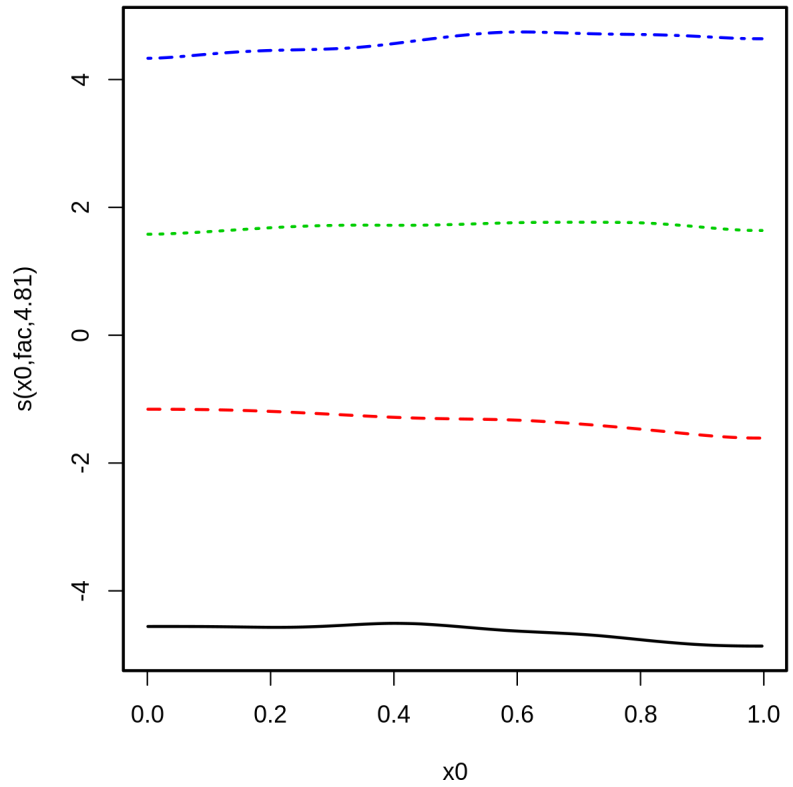
GAMM with a random smooth

```
gamm_smooth ← gam(y ~ s(x0, fac, bs = "fs", m = 1), data = gam_data2)

summary(gamm_smooth)$s.table
#                edf Ref.df        F      p-value
# s(x0,fac) 4.814005     35 8.122226 4.077091e-57
```

GAMM with a random smooth

Here, if the random slope varied along `x0`, we would see different curves for each level:

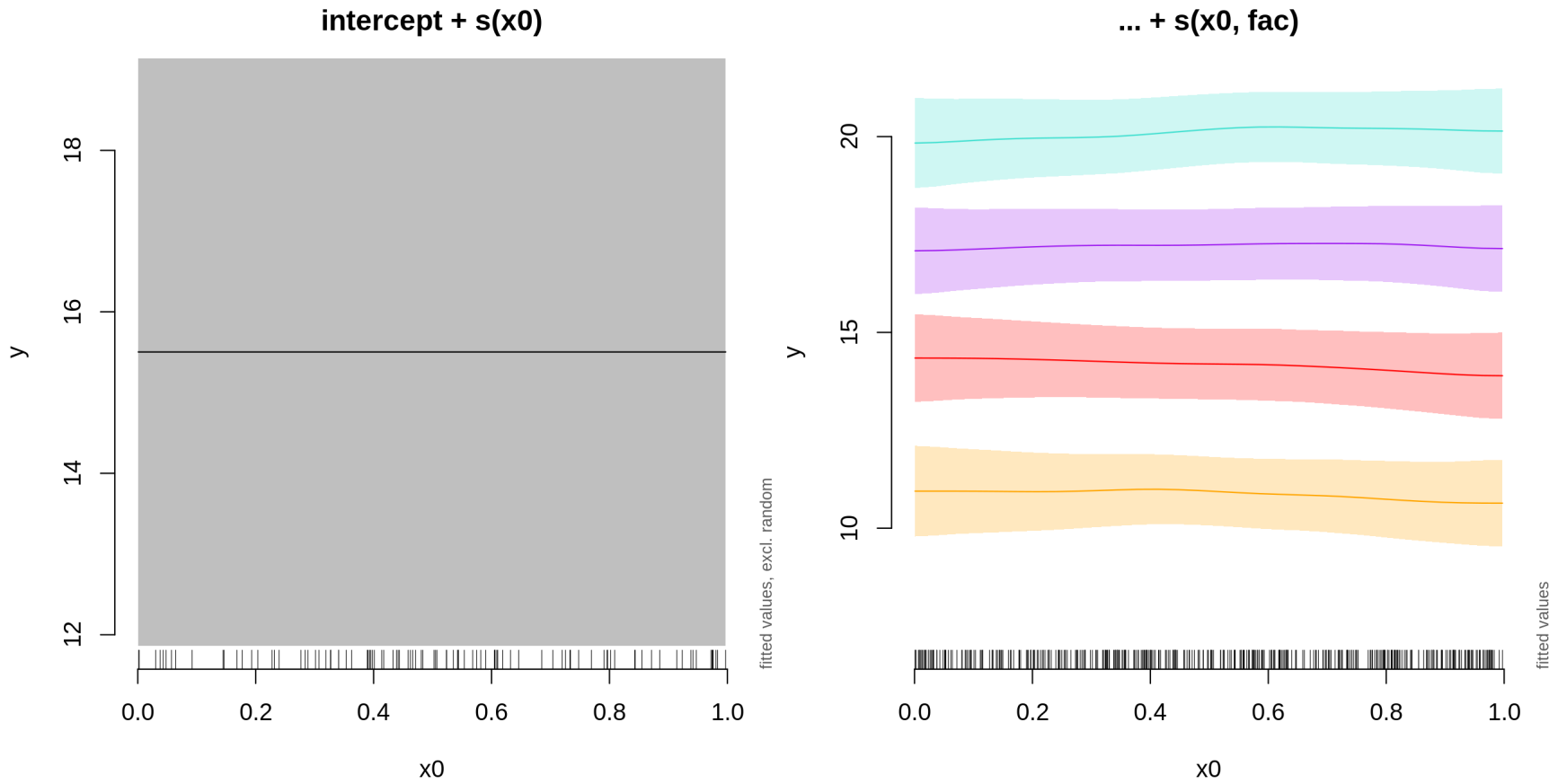
```
plot(gamm_smooth, select = 1)
```



GAMM with a random smooth

```
par(mfrow = c(1,2), cex = 1.1)

plot_smooth(gamm_smooth, view = "x0", rm.ranef = T, main = "intercept + s(x0)")

plot_smooth(gamm_smooth, view = "x0", cond = list(fac = "1"),
            main=" ... + s(x0, fac)", col = 'orange', ylim = c(7,22))

plot_smooth(gamm_smooth, view = "x0", cond = list(fac = "2"), add = T, col='red')

plot_smooth(gamm_smooth, view = "x0", cond = list(fac = "3"), add = T, col = 'pur

plot_smooth(gamm_smooth, view = "x0", cond = list(fac = "4"), add = T, col = 'tur
```

GAMM with a random smooth



Here, if the random slope varied along `x0`*, we would see different curves for each level.*

GAMM

All of the mixed models from this section can be compared using `anova()` to determine the best fit model

```
anova(gamm_intercept, gamm_slope, gamm_int_slope, gamm_smooth, test = "Chisq")
# Analysis of Deviance Table
#
# Model 1: y ~ s(x0) + s(fac, bs = "re")
# Model 2: y ~ s(x0) + s(x0, fac, bs = "re")
# Model 3: y ~ s(x0) + s(fac, bs = "re") + s(fac, x0, bs = "re")
# Model 4: y ~ s(x0, fac, bs = "fs", m = 1)
#   Resid. Df Resid. Dev       Df Deviance  Pr(>Chi)
# 1    392.22     6554.0
# 2    392.33     7290.6 -0.10372  -736.60 6.687e-13 ***
# 3    391.11     6532.7  1.21687   757.94 2.551e-11 ***
# 4    392.64     6690.5 -1.52776  -157.89  0.004796 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

8. GAM behind the scene

A closer look at GAM

We will now take a few minutes to look at what GAMs are doing behind the scenes. Lets first consider a model containing one smooth function of one covariate, x_i:

$$y_i = f(x_i) + \varepsilon_i$$

To estimate the smooth function f, we need to represented the above equation in such a way that it becomes a linear model. This can be done by choosing a basis, $b_i(x)$, defining the space of functions of which f is an element:

$$f(x) = \sum_{i=1}^{q} b_i(x) \times \beta_i$$

Example: a polynomial basis

Suppose that f is believed to be a 4th order polynomial, so that the space of polynomials of order 4 and below contains f. A basis for this space would then be:

$$b_1(x) = 1, b_2(x) = x, b_3(x) = x^2, b_4(x) = x^3, b_5(x) = x^4$$

so that $f(x)$ becomes:

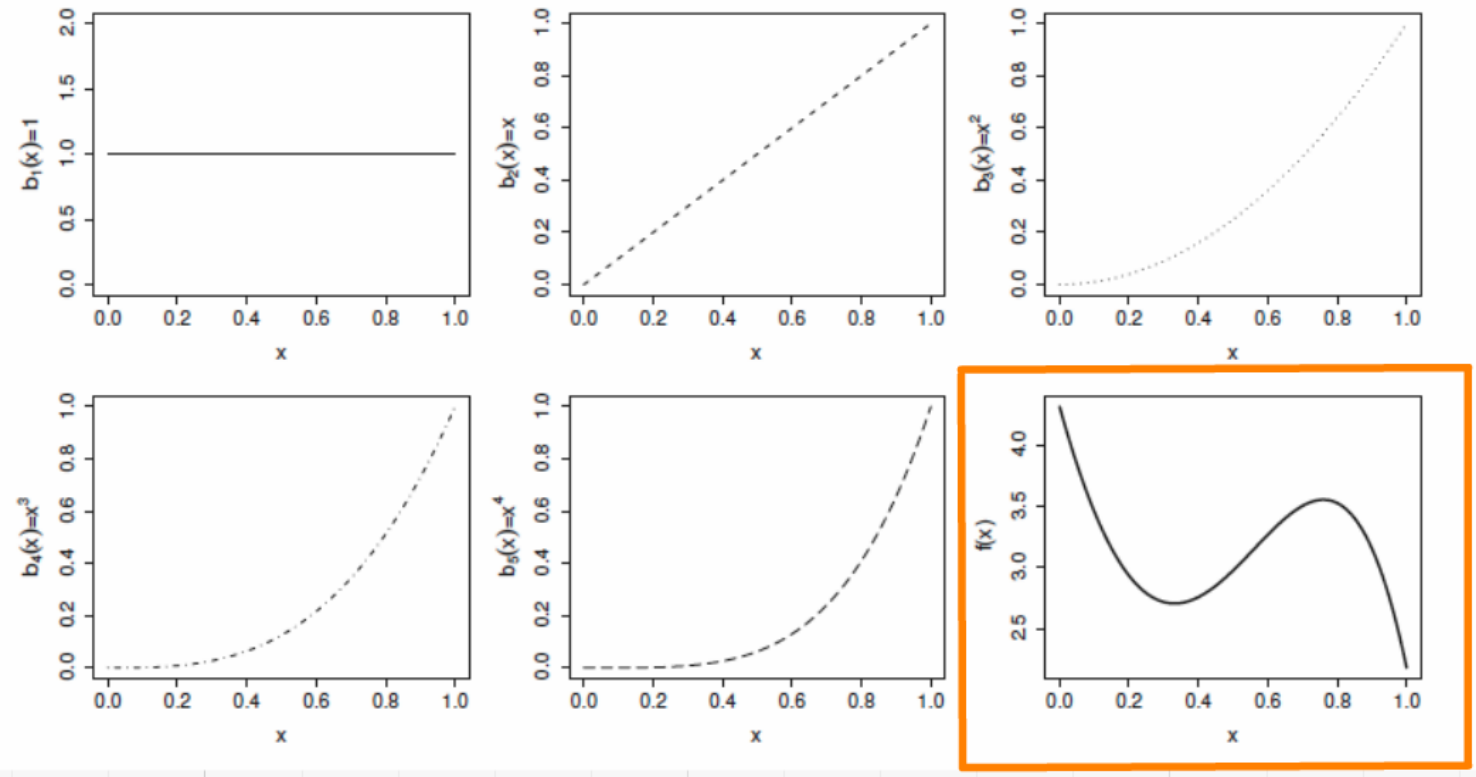
$$f(x) = \beta_1 + x_i\beta_2 + x_i^2\beta_3 + x_i^3\beta_4(x) + x_i^4\beta_5$$

and the full model now becomes:

$$y_i = \beta_1 + x_i\beta_2 + x_i^2\beta_3 + x_i^3\beta_4(x) + x_i^4\beta_5 + \varepsilon_i$$

Example: a polynomial basis

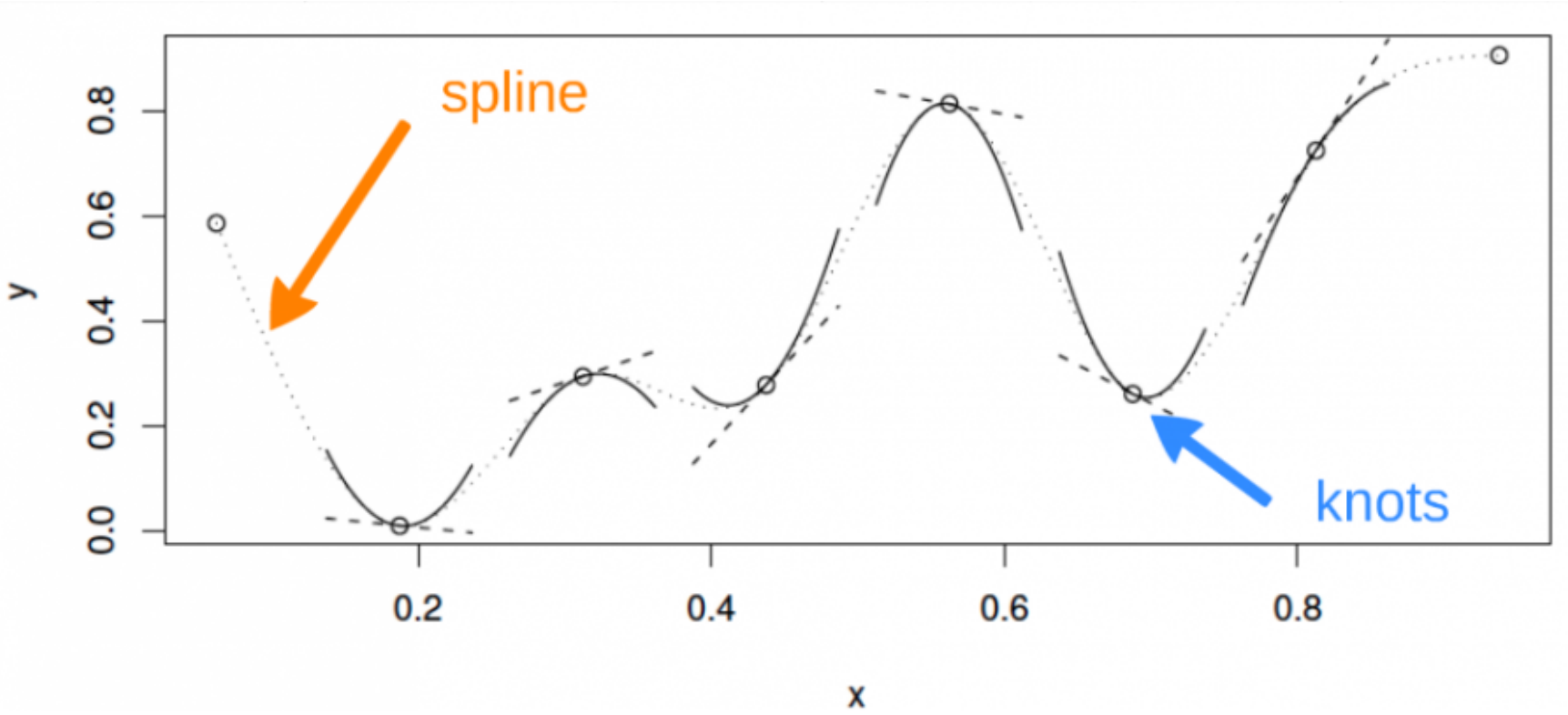
The basis functions are each multiplied by a real valued parameter, β_i, and are then summed to give the final curve $f(x)$.



By varying the β_i we can vary the form of $f(x)$ to produce any polynomial function of order 4 or lower.

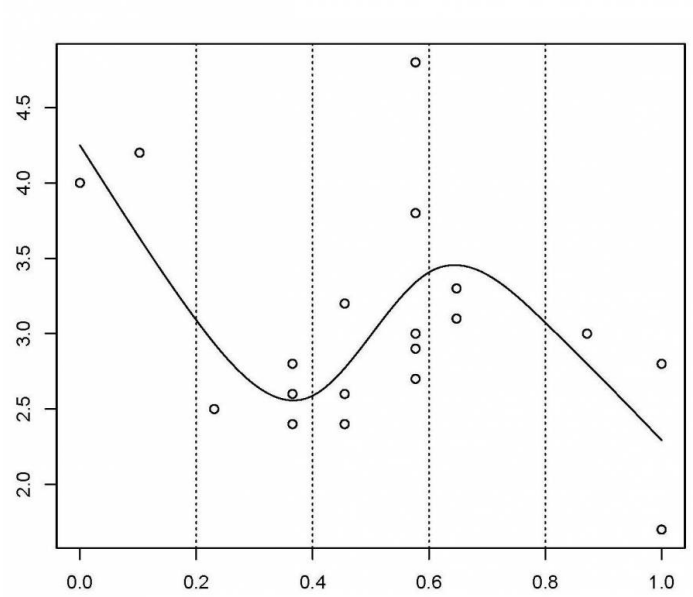
Example: a cubic spline basis

A cubic spline is a curve constructed from sections of a cubic polynomial joined together so that they are continuous in value. Each section of cubic has different coefficients.



Example: a cubic spline basis

Here's a representation of a smooth function using a rank 5 cubic spline basis with knot locations at increments of 0.2:



Here, the knots are evenly spaced through the range of observed x values. However, the choice of the degree of model smoothness is controlled by the the number of knots, which was arbitrary.

Is there a better way to select the knot locations?

Controlling the degree of smoothing with penalized regression splines

Instead of controlling smoothness by altering the number of knots, we keep that fixed to size a little larger than reasonably necessary, and control the model's smoothness by adding a "wiggleness" penalty.

So, rather than fitting the model by minimizing (as with least squares regression):

$$||y - XB||^2$$

it can be fit by minimizing:

$$||y - XB||^2 + \lambda \int_0^1 [f''(x)]^2 dx$$

As λ goes to infty, the model becomes linear.

Controlling the degree of smoothing with penalized regression splines

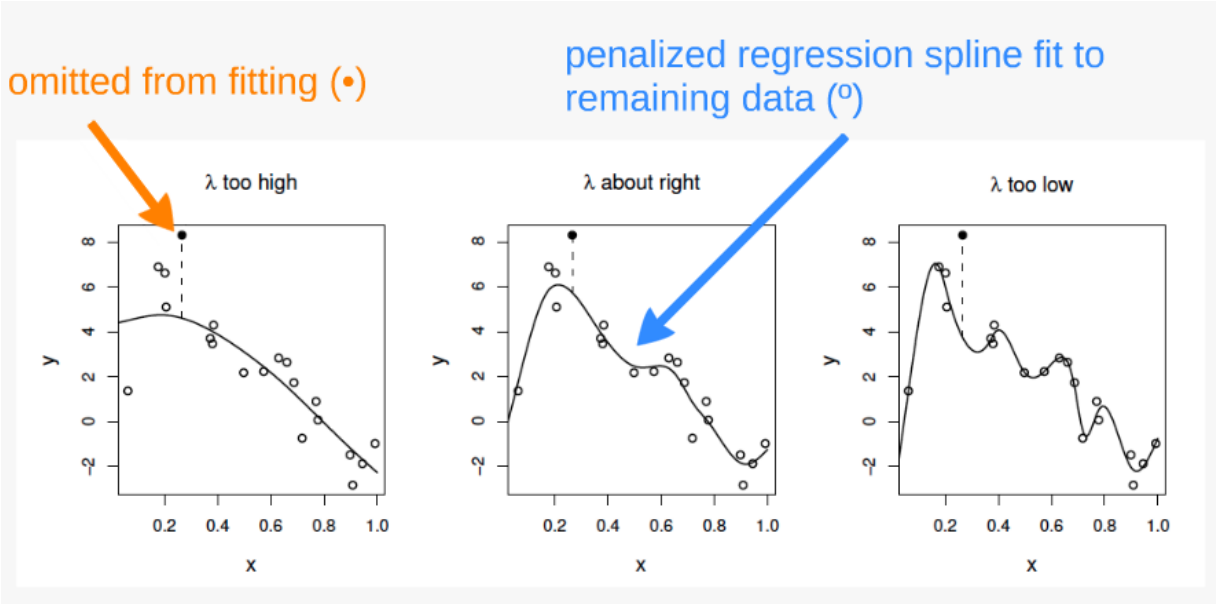
If λ is too high then the data will be over smoothed, and if it is too low then the data will be under smoothed.

Ideally, it would be good to choose λ so that the predicted $\hat{f}$ is as close as possible to f. A suitable criterion might be to choose λ to minimize:

$$M = 1/n \times \sum_{i=1}^{n} (\hat{f}_i - f_i)^2$$

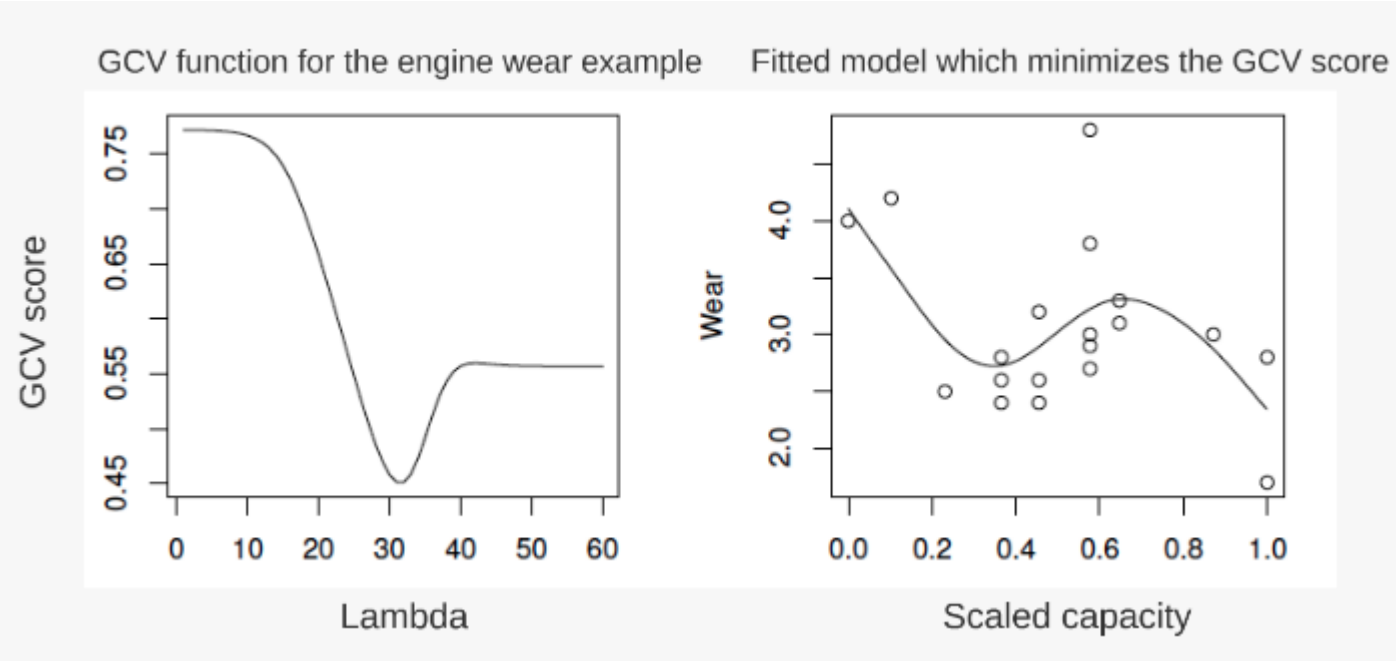
Since f is unknown, M is estimated using a generalized cross validation technique that leaves out each datum from the data in turn and considers the average ability of models fitted to the remaining data to predict the left out datum.

Principle behind cross validation



1. fits many of the data poorly and does no better with the missing point.
2. fits the underlying signal quite well, smoothing through the noise and the missing datum is reasonably well predicted.
3. fits the noise as well as the signal and the extra variability induced causes it to predict the missing datum rather poorly.

Principle behind cross validation



Brief note on estimated degrees of freedom (edf)

How many degrees of freedom does a fitted GAM have?

Instead of providing the output of the cross-validation in terms of λ (model complexity), the GAM function in the `mgcv` package uses a term called the estimated degrees of freedom (edf)

Because the number of free parameters in GAMs is difficult to define, the edf are related to λ, such that the greater the penalty, the smaller the edf.

For example, if the arbitrarily large number of knots is k = 10, then k-1 sets the upper limit on the edf associated with a smooth term. This number then decreases as the penalty λ increases until the best fit penalty is found by cross-validation.

Ressources

There's a great deal more out there on GAM... this was just the very surface.

Simon Wood, the author of the `mgcv` package has a very useful website with introductory talks and notes on how to use GAM.

He's also written a book, Generalized Additive Models: An Introduction with R, which we used as reference for this workshop.

Material from this workshop were also obtained from the following fantastic blogs and tutorials:

- From the bottom of the heap
- Overview GAMM analysis of time series data
- Advanced Analysis of Time series data

Finally, the help pages, available through `?gam` in R, are an excellent resource.

Thank you for attending this workshop!