

Generalized Additive Models

David L Miller

Overview

- What is a GAM?
- What is smoothing?
- How do GAMs work? (*Roughly*)
- Fitting and plotting simple models

What is a GAM?

Generalized Additive Models

- Generalized: many response distributions
- Additive: terms **add** together
- Models: well, it's a model...

To GAMs from GLMs and LMs

(Generalized) Linear Models

Models that look like:

$$y_i = \beta_0 + x_{1i}\beta_1 + x_{2i}\beta_2 + \dots + \epsilon_i$$

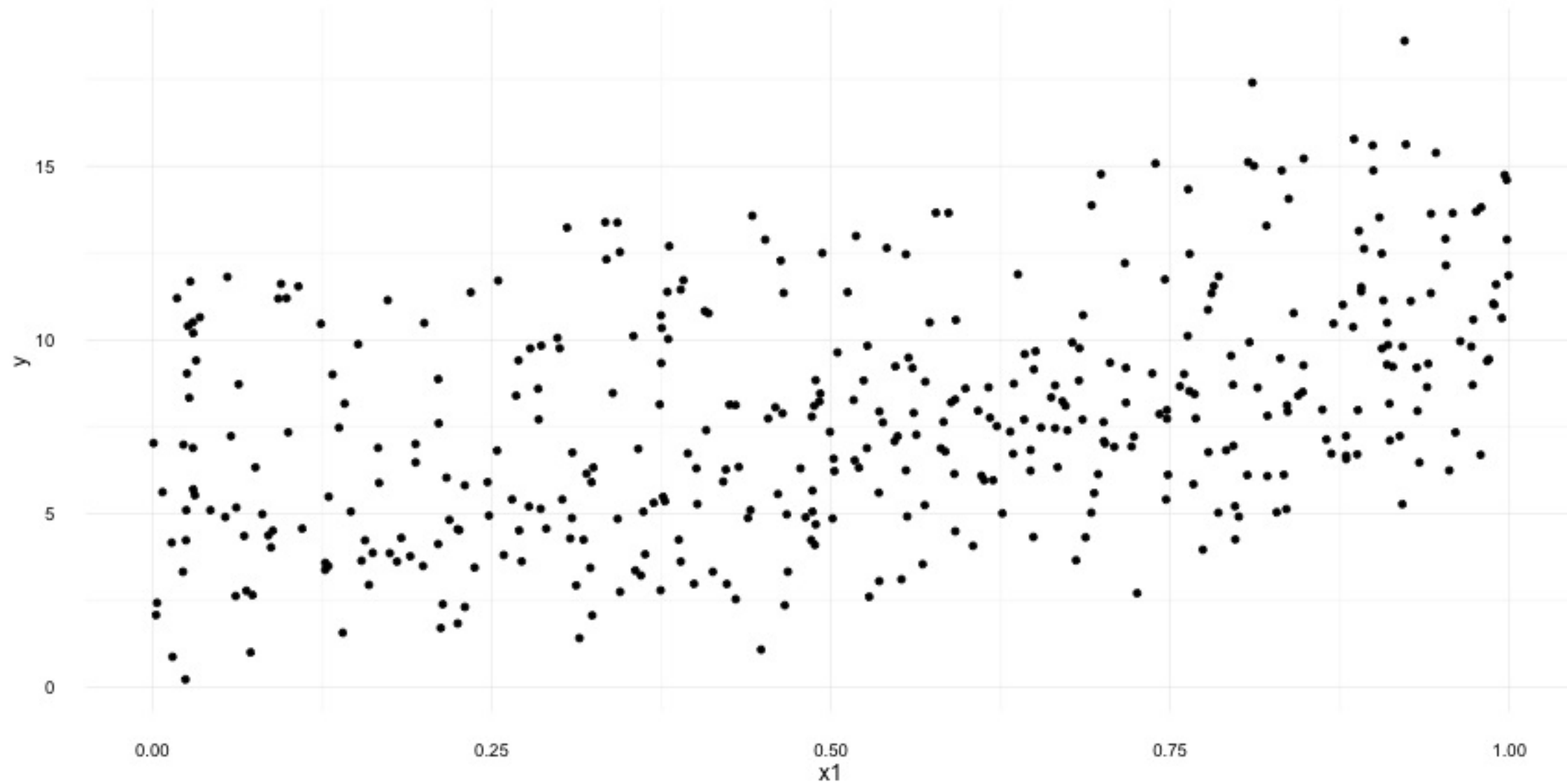
(describe the response, y_i , as linear combination of the covariates, x_{ji} , with an offset)

We can make $y_i \sim$ any exponential family distribution (Normal, Poisson, etc).

Error term ϵ_i is normally distributed (usually).

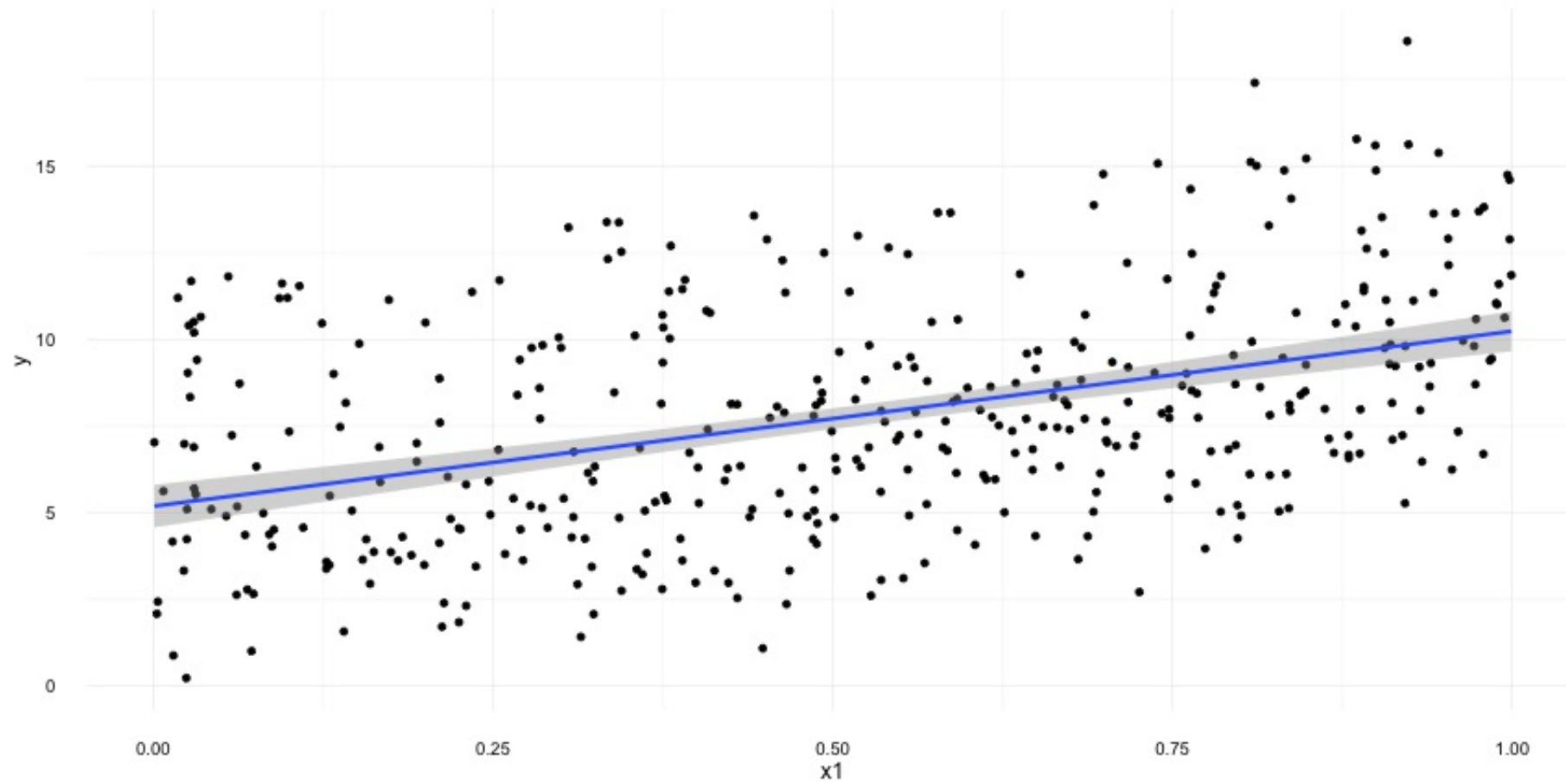
Why bother with anything more complicated?!

Is this linear?



Is this linear? Maybe?

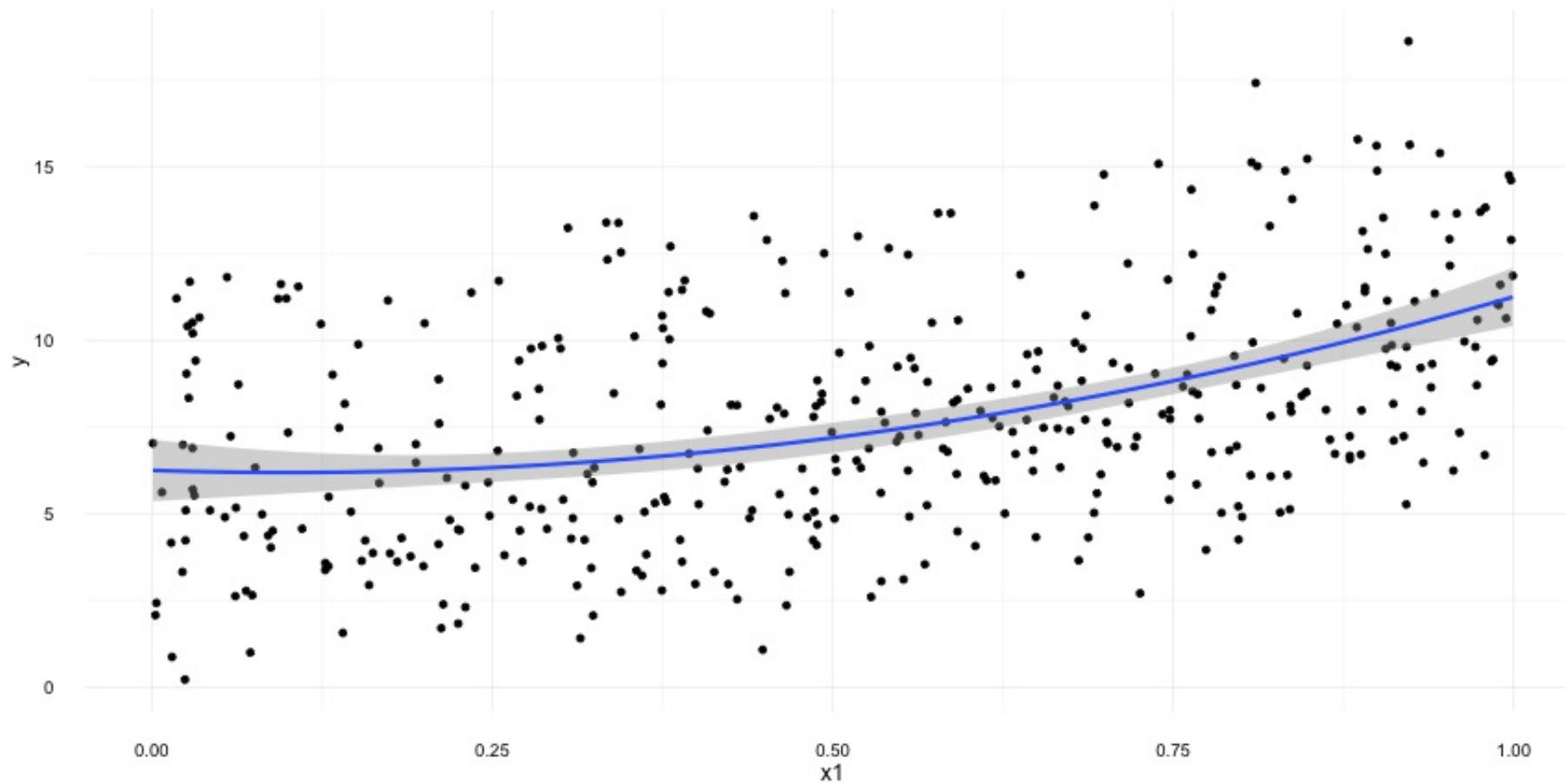
```
lm(y ~ x1, data=dat)
```



What can we do?

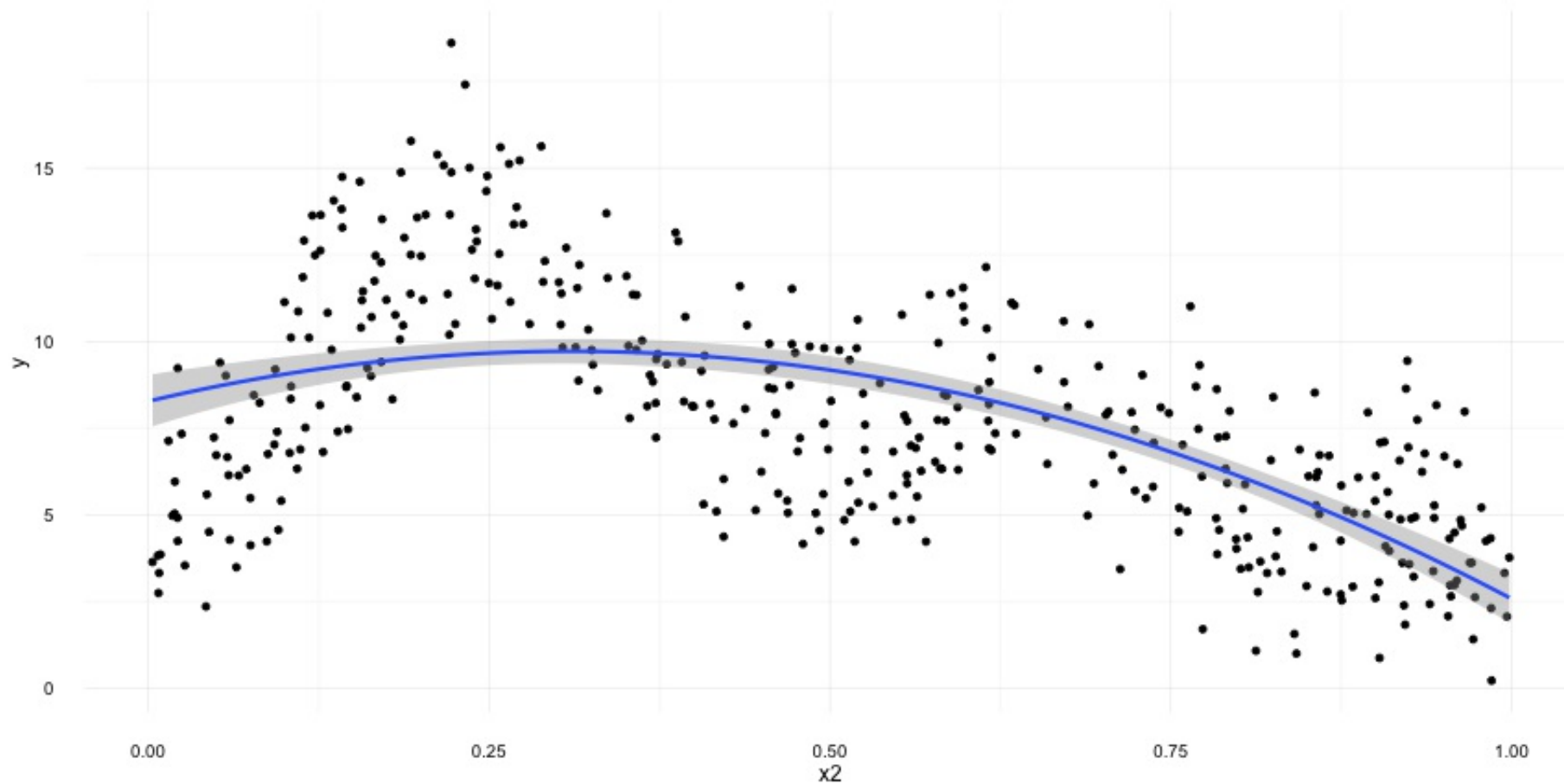
Adding a quadratic term?

```
lm(y ~ x1 + poly(x1, 2), data=dat)
```



Is this sustainable?

- Adding in quadratic (and higher terms) *can* make sense
- This feels a bit *ad hoc*
- Better if we had a **framework** to deal with these issues?



[drumroll]

What does a model look like?

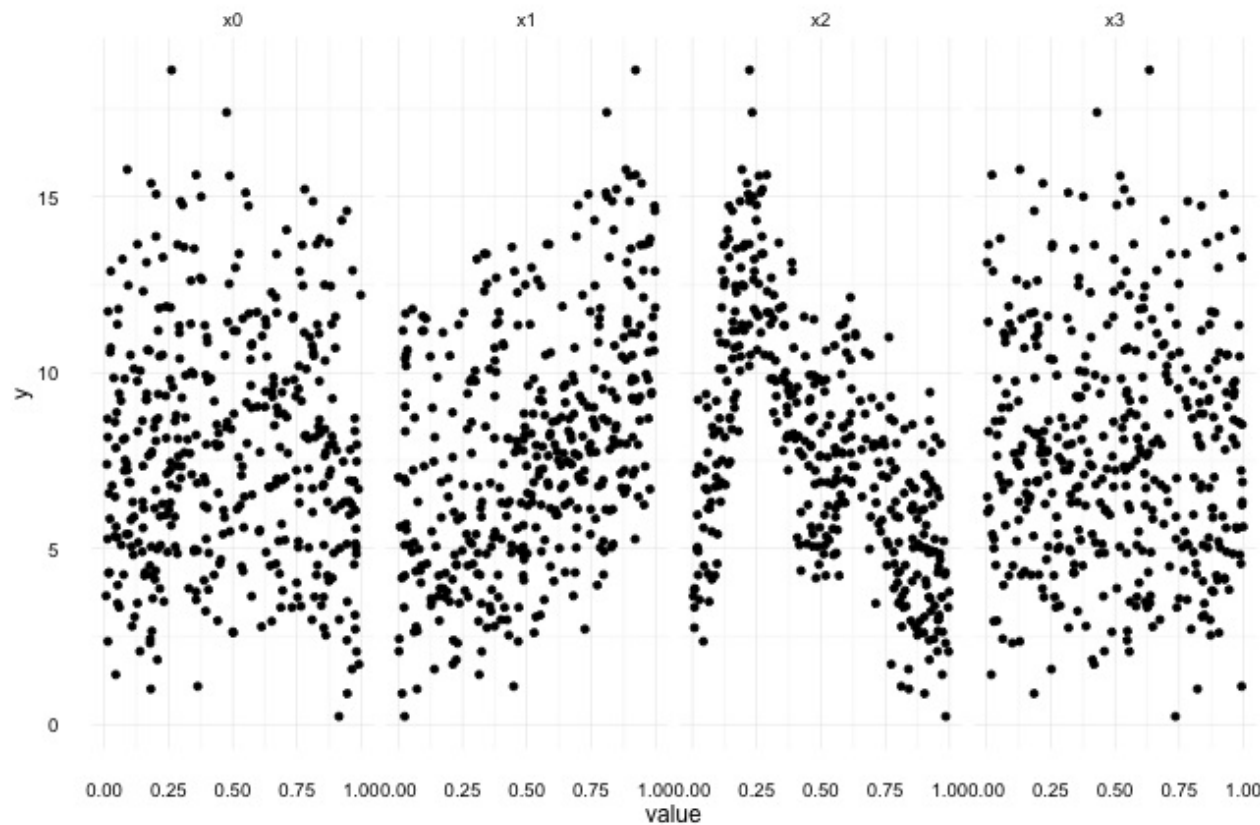
$$y_i = \beta_0 + \sum_j s_j(x_{ji}) + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$

Let's pretend that $y_i \sim \text{Normal}$

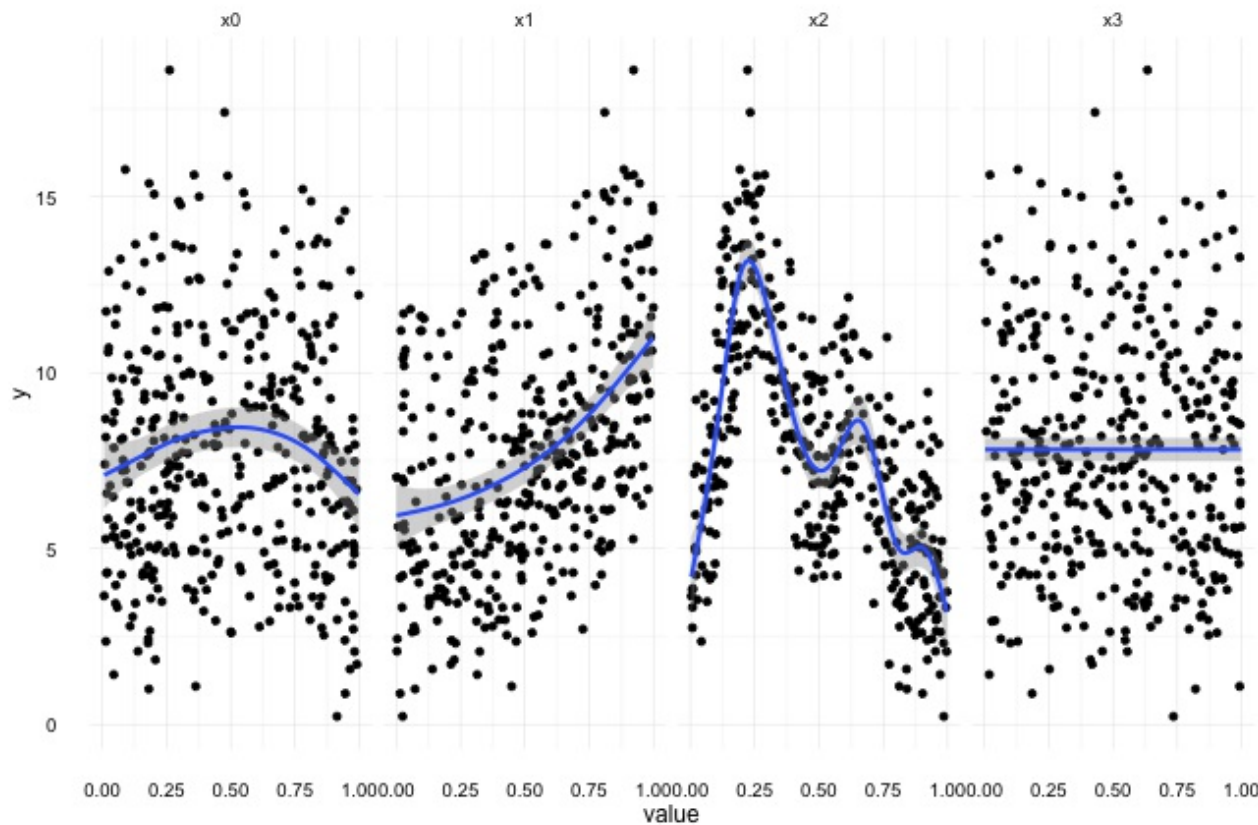
Remember that we're modelling the mean of this distribution!

Okay, but what about these "s" things?



- Think s =**smooth**
- Want to model the covariates flexibly
- Covariates and response not necessarily linearly related!
- Want some “wiggles”

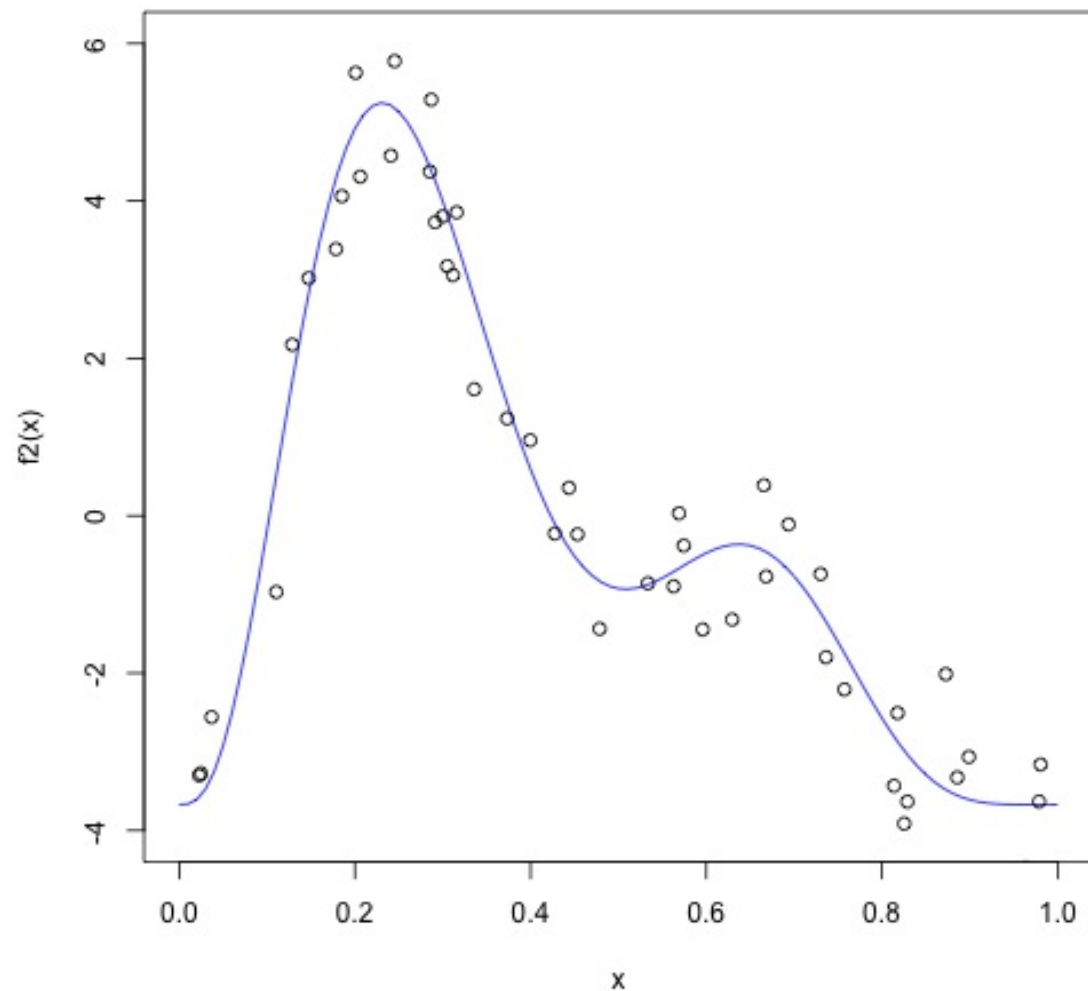
Okay, but what about these "s" things?



- Think $s = \text{smooth}$
- Want to model the covariates flexibly
- Covariates and response not necessarily linearly related!
- Want some “wiggles”

What is smoothing?

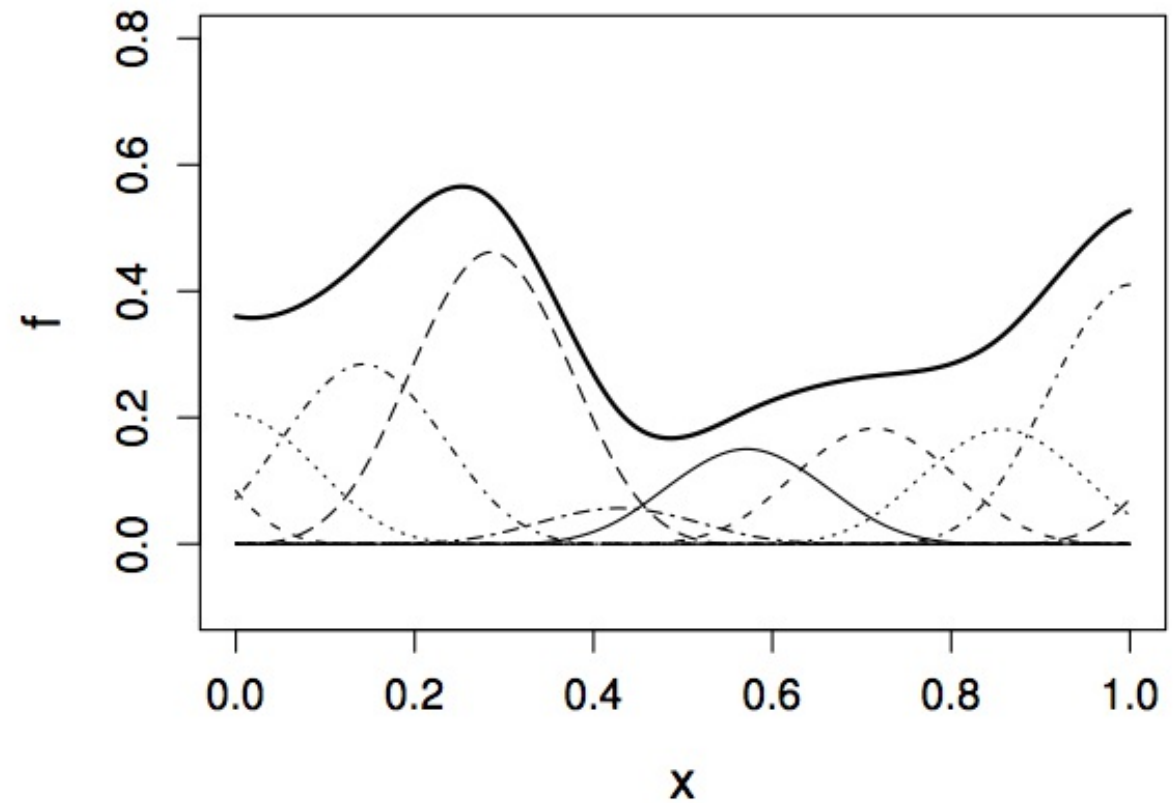
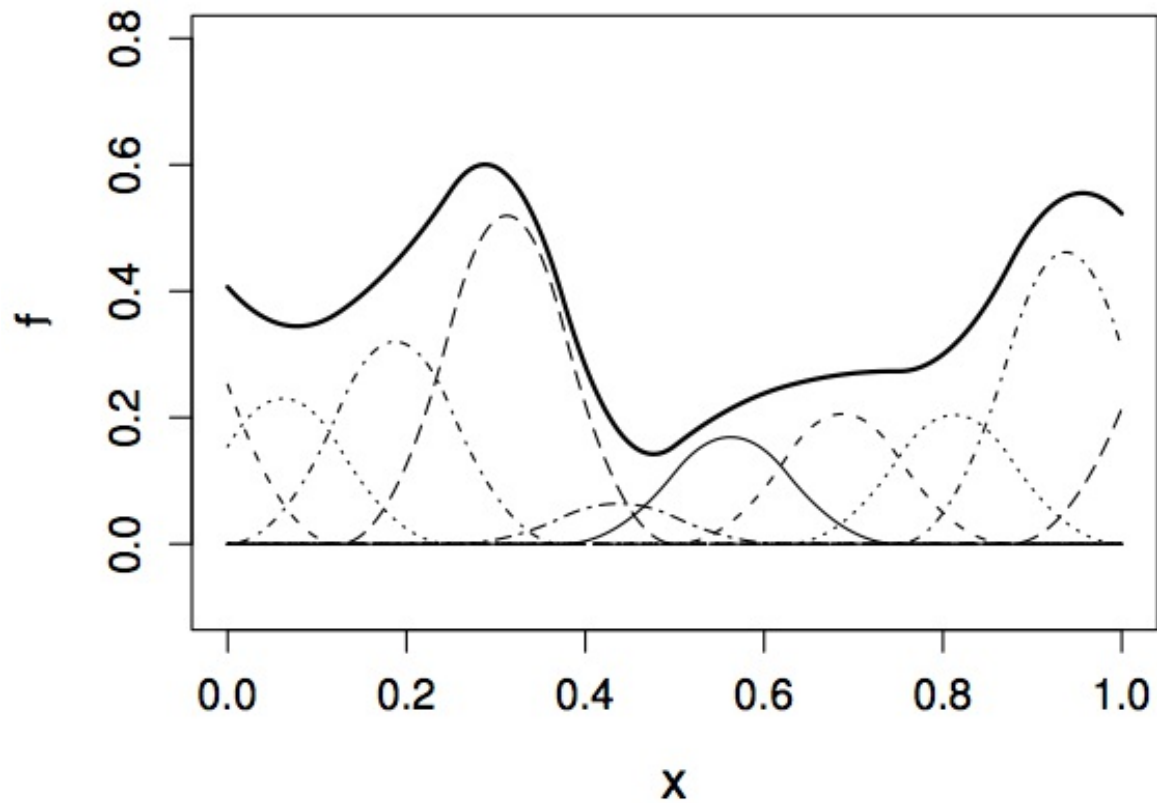
Straight lines vs. interpolation



- Want a line that is “close” to all the data
- Don't want interpolation – we know there is “error”
- Balance between interpolation and “fit”

Splines

- Functions made of other, simpler functions
- **Basis functions** b_k , estimate β_k
- $s(x) = \sum_{k=1}^K \beta_k b_k(x)$
- Makes the math(s) much easier



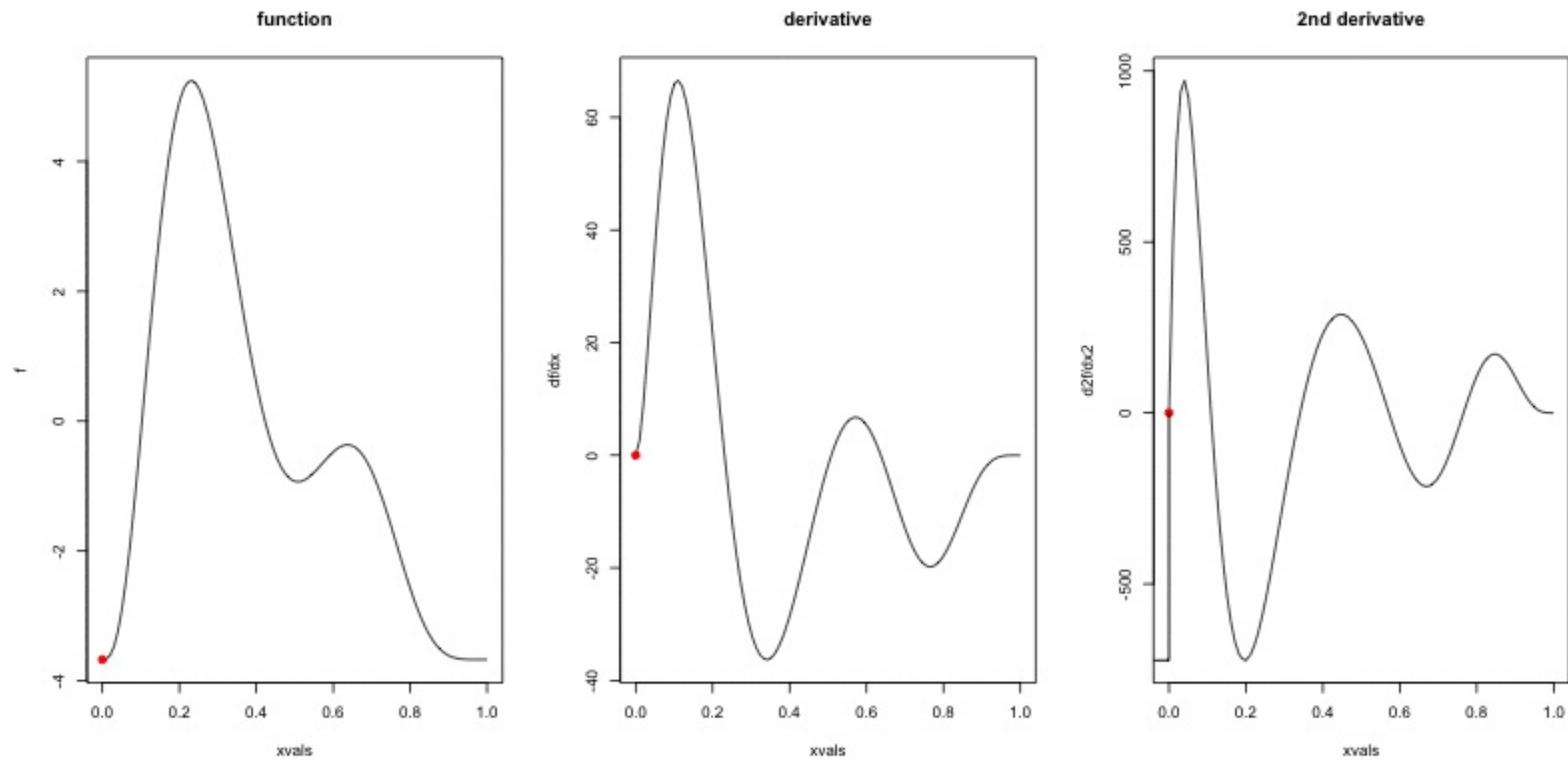
Design matrices

- We often write models as $X\beta$
- X is our data
- β are parameters we need to estimate
- For a GAM it's the same
- X has columns for each basis, evaluated at each observation

Measuring wigglyness

- Visually:
 - Lots of wiggles == NOT SMOOTH
 - Straight line == VERY SMOOTH
- How do we do this mathematically?
 - Derivatives!
 - (Calculus *was* a useful class afterall!)

Wigglyness by derivatives



What was that grey bit?

$$\int_{\mathbb{R}} \left(\frac{\partial^2 f(\mathbf{x})}{\partial^2 \mathbf{x}} \right)^2 d\mathbf{x}$$

(Take some derivatives of the smooth and integrate them over \mathbf{x})

(Turns out we can always write this as $\boldsymbol{\beta}^T S \boldsymbol{\beta}$, so the $\boldsymbol{\beta}$ is separate from the derivatives)

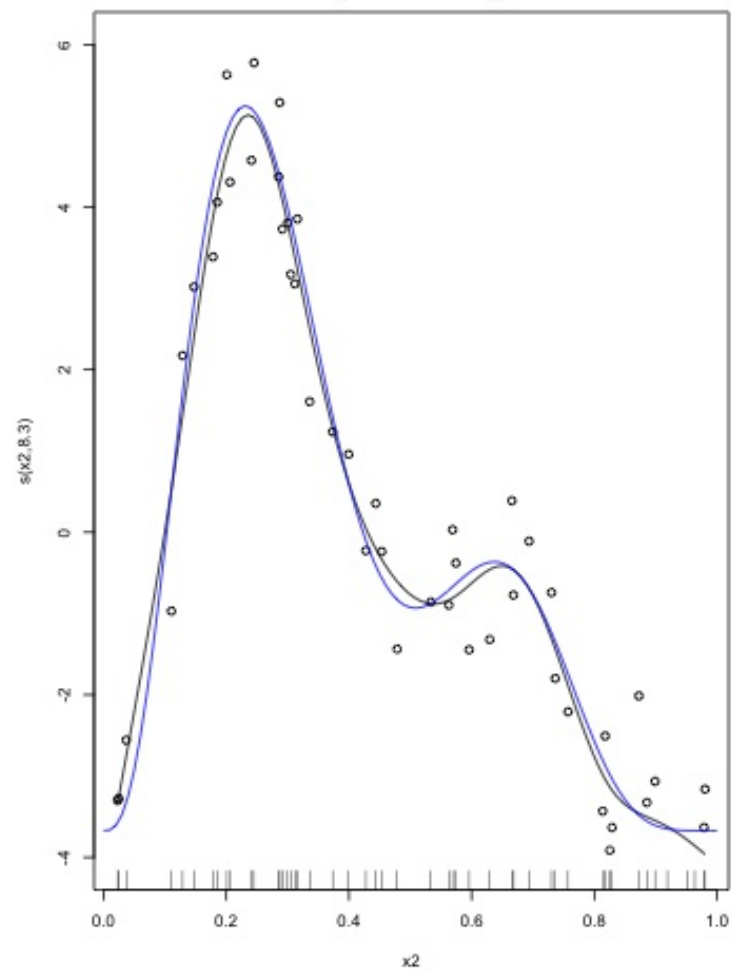
(Call S the **penalty matrix**)

Making wigglyness matter

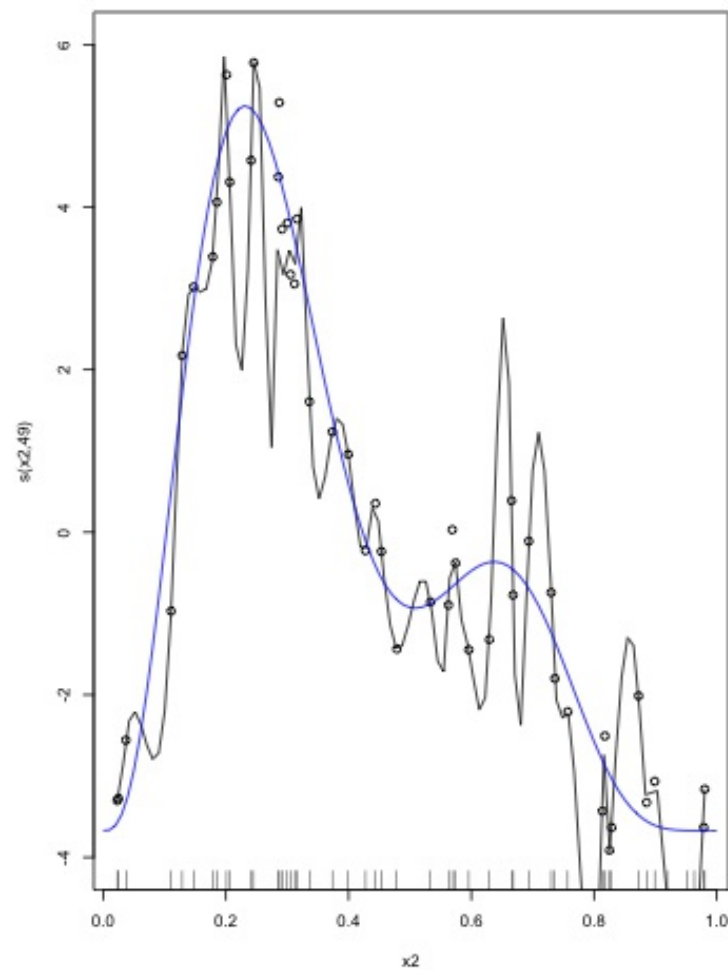
- $\beta^T S \beta$ measures wigglyness
- “Likelihood” measures closeness to the data
- Penalise closeness to the data...
- Use a **smoothing parameter** to decide on that trade-off...
 - $\lambda \beta^T S \beta$
- Estimate the β_k terms but penalise objective
 - “closeness to data” + penalty

Smoothing parameter

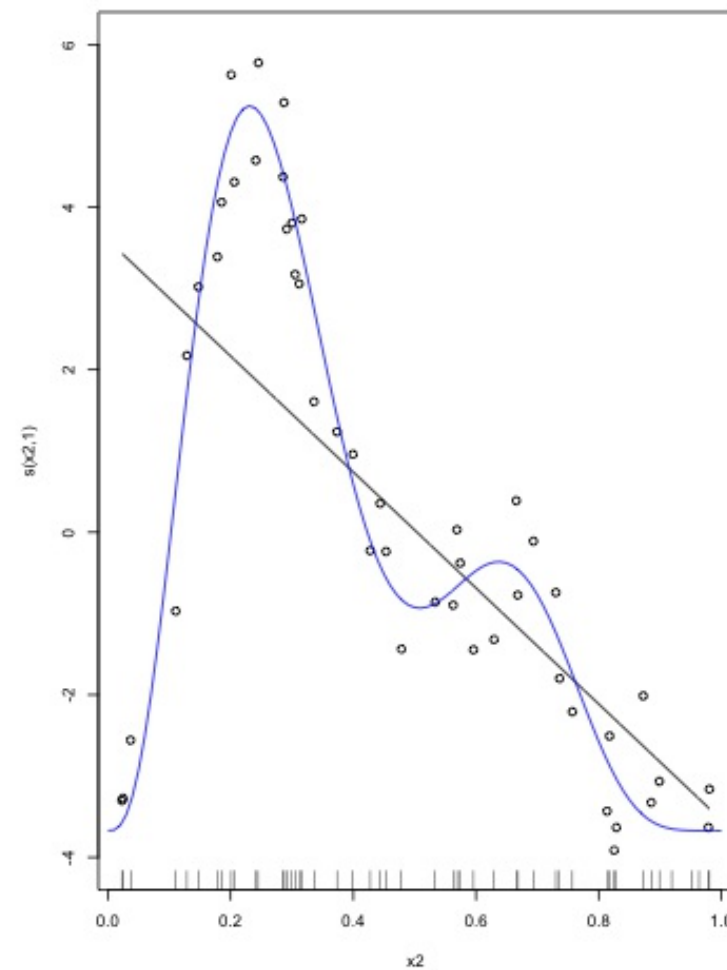
$\lambda = \text{just right}$



$\lambda = 0$

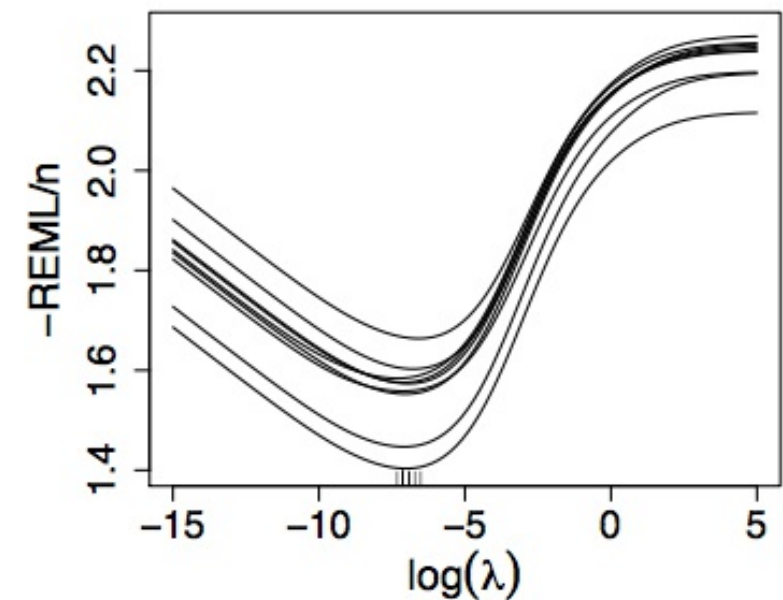
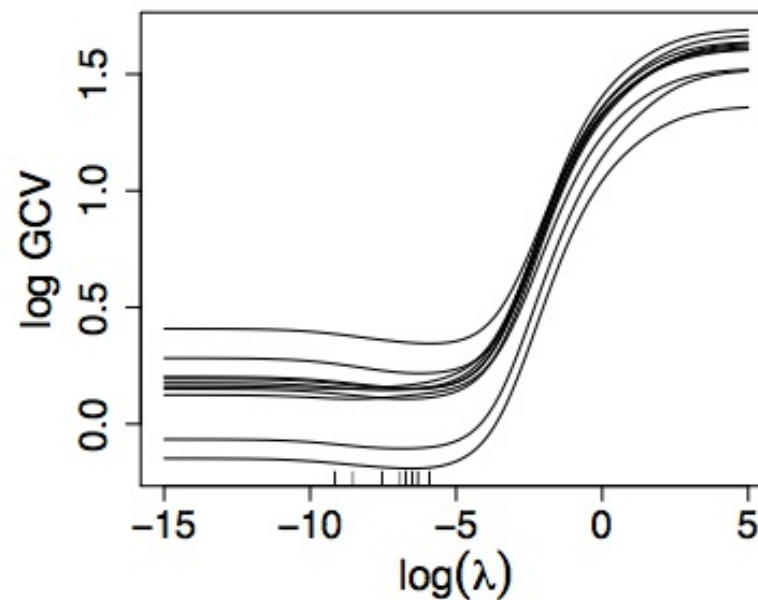
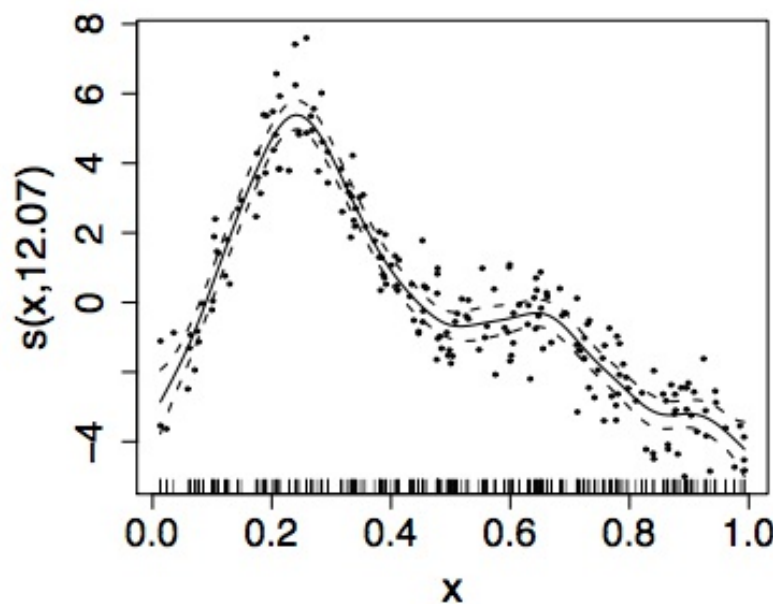


$\lambda = \infty$



Smoothing parameter selection

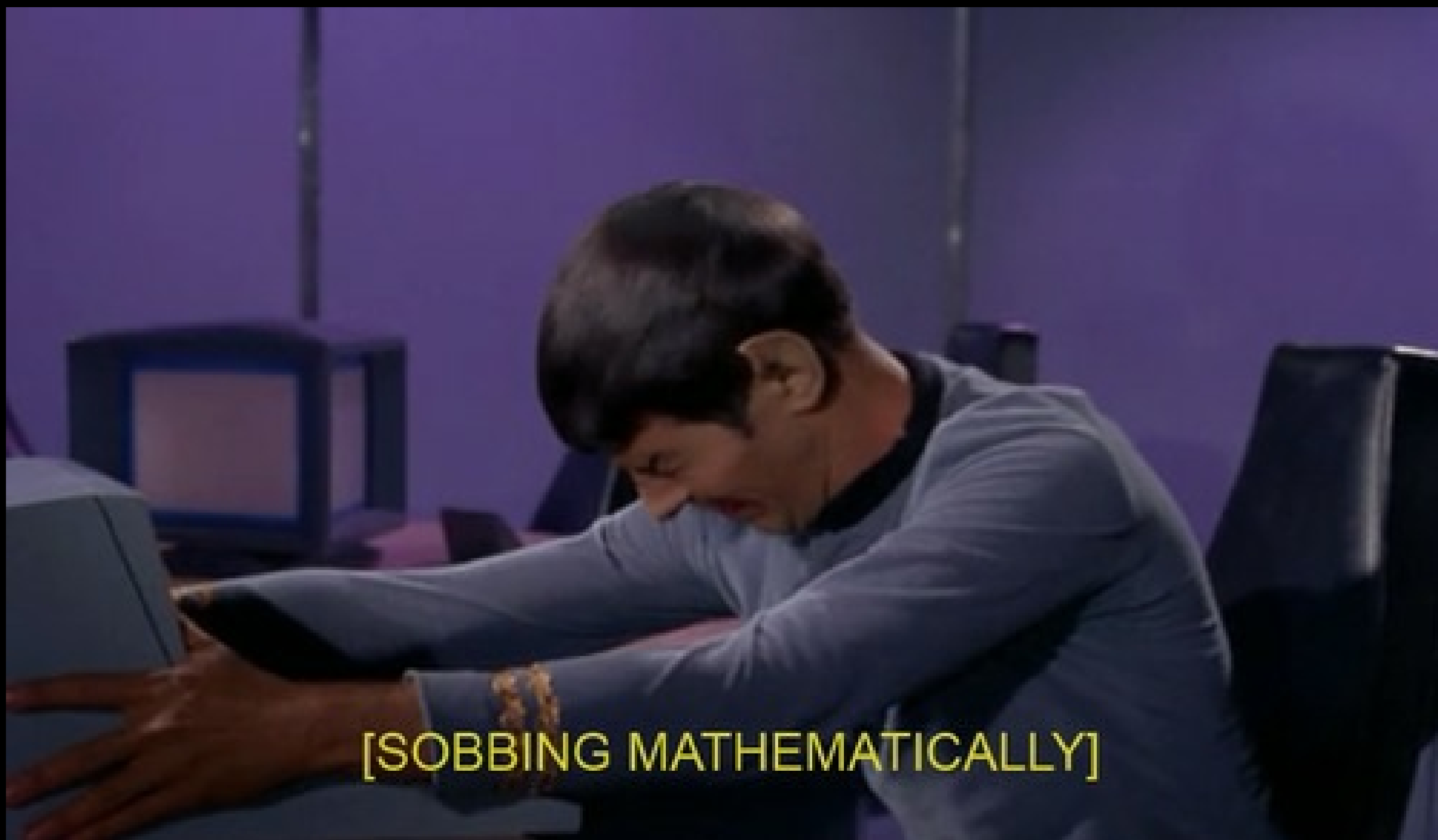
- Many methods: AIC, Mallow's C_p , GCV, ML, REML
- Recommendation, based on simulation and practice:
 - Use REML or ML
 - Reiss & Ogden (2009), Wood (2011)



Maximum wiggliness

- We can set **basis complexity** or “size” (k)
 - Maximum wigglyness
- Smooths have **effective degrees of freedom** (EDF)
- $\text{EDF} < k$
- Set k “large enough”
 - Penalty does the rest

More on this in a bit...



[SOBBING MATHEMATICALLY]

GAM summary

- Straight lines suck
- Want wiggles
- Use little functions (**basis functions**) to make big functions (**smooths**)
- Need to make sure your smooths are **wiggly enough**
- Use a **penalty** to trade off wiggleness/generalizability

Fitting GAMs in practice

Translating maths into R

A simple example:

$$y_i = \beta_0 + s(x) + s(w) + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$

Let's pretend that $y_i \sim \text{Normal}$

- inside the link: `formula = y ~ s(x) + s(w)`
- response distribution: `family=gaussian()`
- data: `data=some_data_frame`

Putting that together

```
my_model <- gam(y ~ s(x) + s(w),  
               family = gaussian(),  
               data = some_data_frame,  
               method = "REML")
```

- `method="REML"` uses REML for smoothness selection (default is `"GCV.Cp"`)

What about a practical
example?

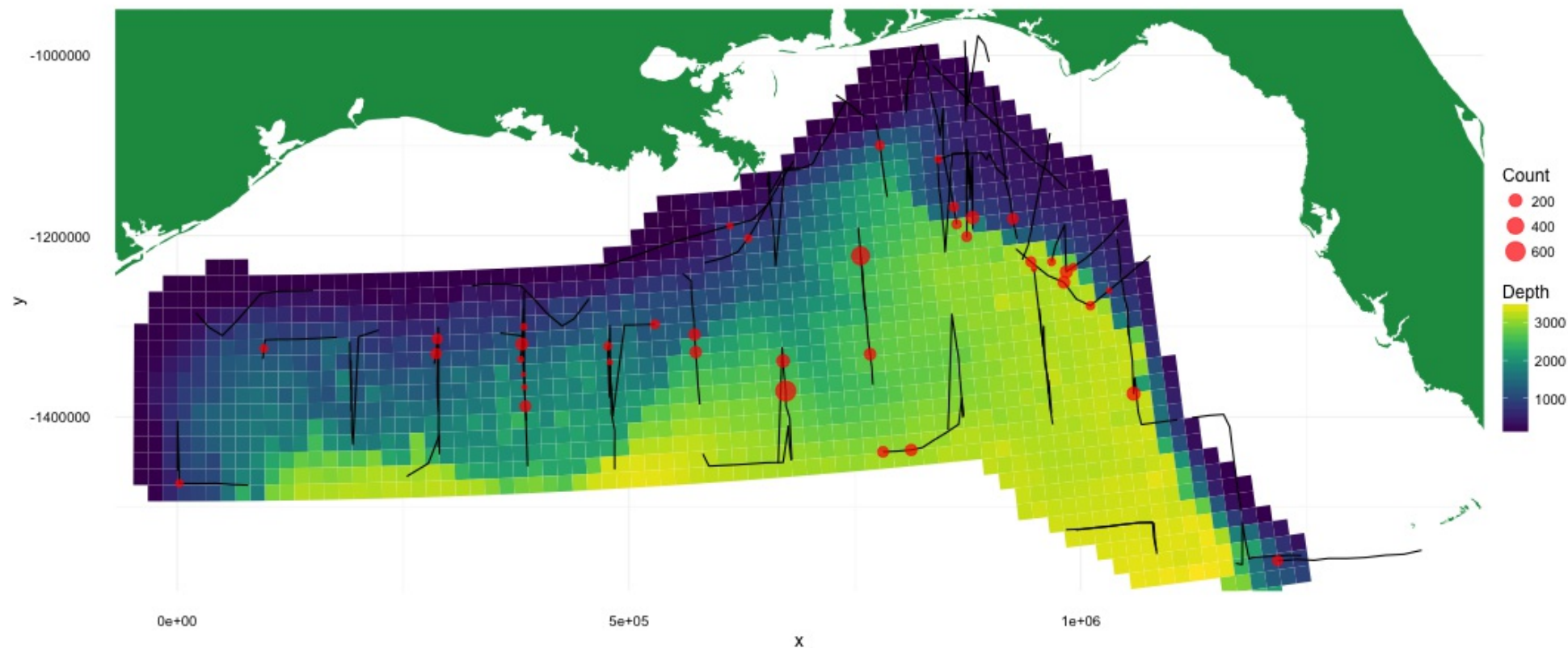
Pantropical spotted dolphins

- Example taken from Miller et al (2013)
- [Paper appendix](#) has a better analysis
- Simple example here, ignoring all kinds of important stuff!



Inferential aims

- How many dolphins are there?
- Where are the dolphins?
- What are they interested in?



A simple dolphin model

```
library(mgcv)
dolphins_depth <- gam(count ~ s(depth) + offset(off.set),
                      data = mexdolphins,
                      family = quasipoisson(),
                      method = "REML")
```

- count is a function of depth
- off.set is the effort expended
- we have count data, try quasi-Poisson distribution

What did that do?

```
summary(dolphins_depth)
```

```
Family: quasipoisson  
Link function: log
```

```
Formula:  
count ~ s(depth) + offset(off.set)
```

```
Parametric coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-18.2344	0.8949	-20.38	<2e-16 ***

```
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

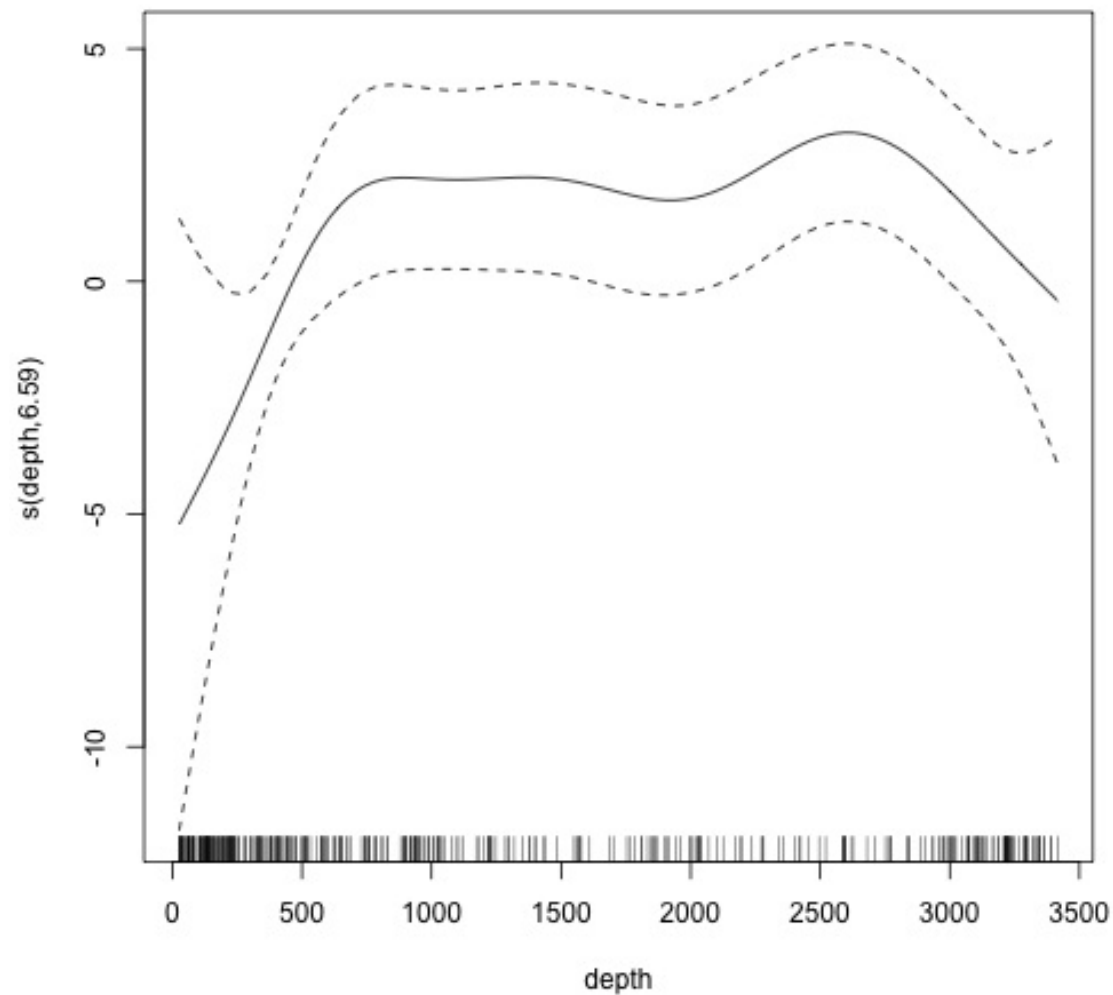
```
Approximate significance of smooth terms:
```

	edf	Ref.df	F	p-value
s(depth)	6.592	7.534	2.329	0.0224 *

```
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
R-sq.(adj) = 0.0545    Deviance explained = 26.4%  
-REML = 948.28    Scale est. = 145.34    n = 387
```

Plotting



- `plot(dolphins_depth)`
- Dashed lines indicate ± 2 standard errors
- Rug plot
- On the link scale
- EDF on y axis

Thin plate regression splines

- Default basis
- One basis function per data point
- Reduce # basis functions (eigendecomposition)
- Fitting on reduced problem
- Multidimensional
- Wood (2003)

Bivariate terms

- Assumed an additive structure
- No interaction
- We can specify $s(x, y)$ (and $s(x, y, z, \dots)$)
- (Assuming *isotropy* here...)

Adding a term

- Add a **surface** for location (x and y)
- Just use + for an extra term

```
dolphins_depth_xy <- gam(count ~ s(depth) + s(x, y) +  
  offset(off.set),  
                        data = mexdolphins,  
                        family=tw(), method="REML")
```

Summary

```
summary(dolphins_depth_xy)
```

Family: Tweedie(p=1.362)

Link function: log

Formula:

count ~ s(depth) + s(x, y) + offset(off.set)

Parametric coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-18.4281	0.4845	-38.03	<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(depth)	5.42	6.518	5.178	2.52e-05 ***
s(x,y)	2.22	2.424	1.645	0.143

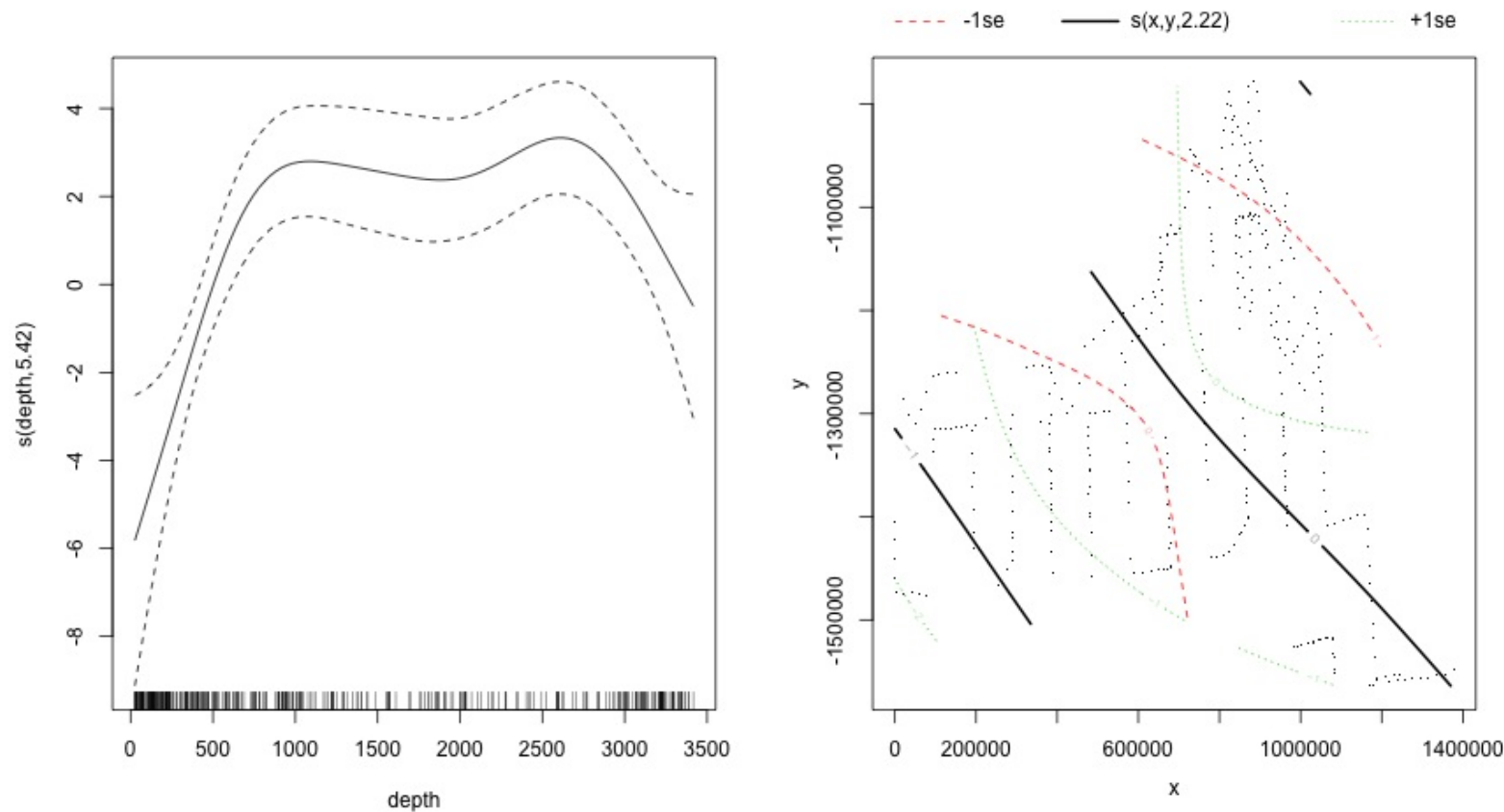
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.0448 Deviance explained = 36%

-REML = 338.31 Scale est. = 54.029 n = 387

Plotting

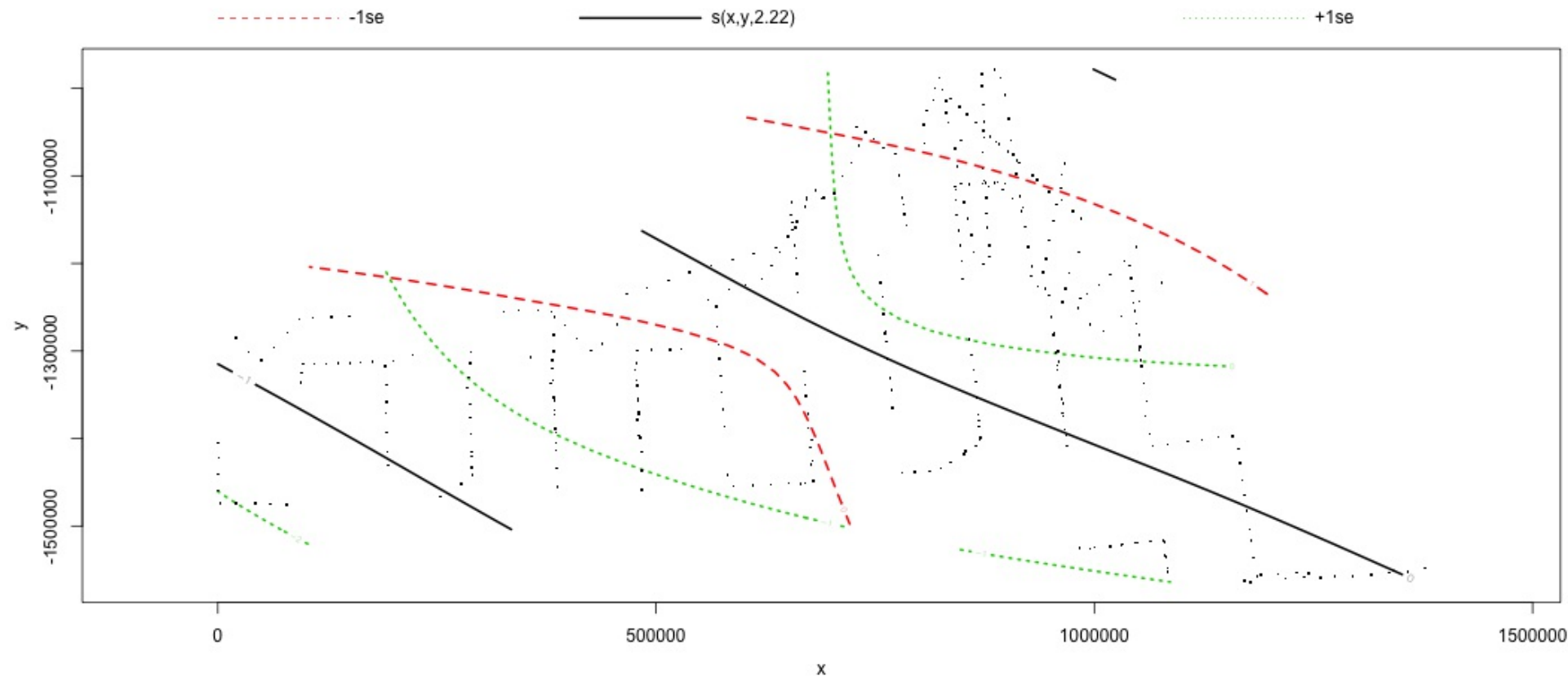
```
plot(dolphins_depth_xy, scale=0, pages=1)
```



- `scale=0`: each plot on different scale
- `pages=1`: plot together

Plotting 2d terms... erm...

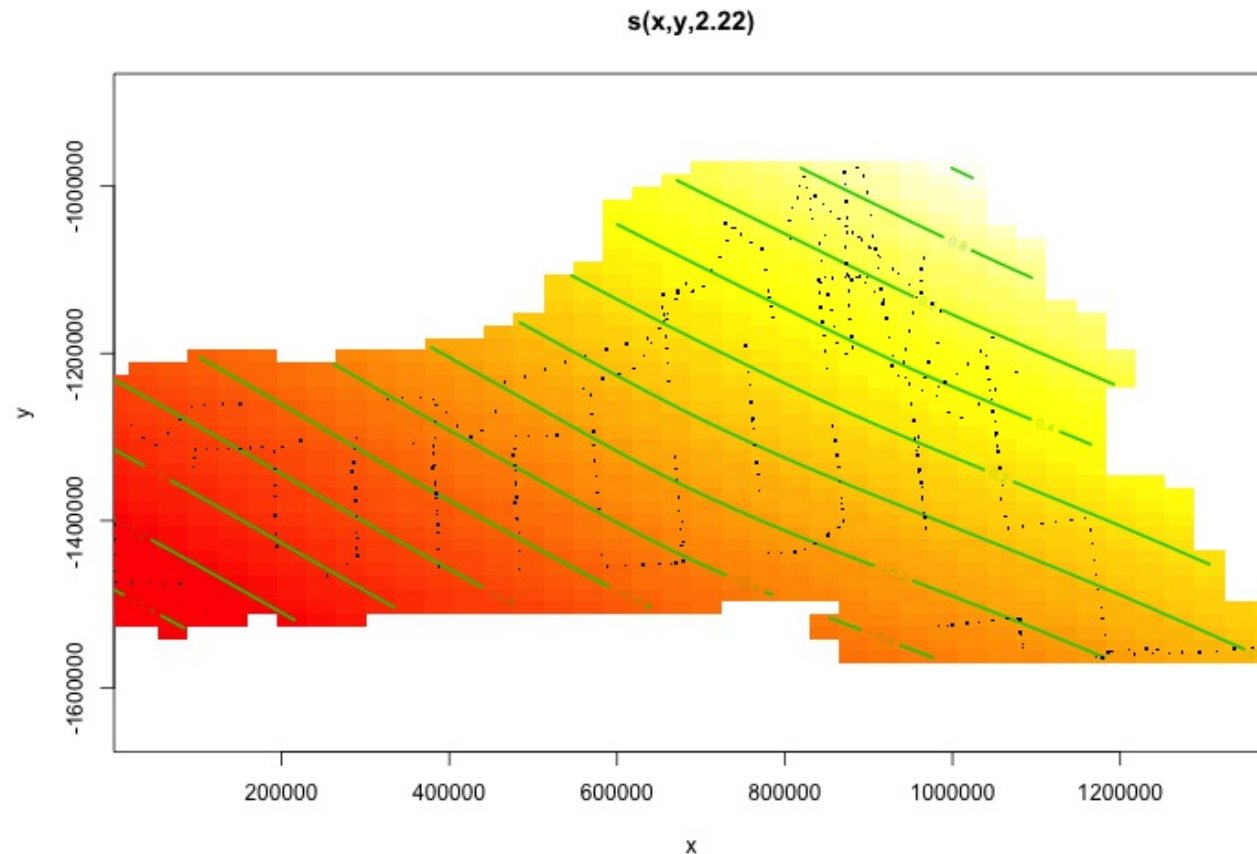
```
plot(dolphins_depth_xy, select=2, cex=2, asp=1, lwd=2)
```



- `select=` picks which smooth to plot

Let's try something different

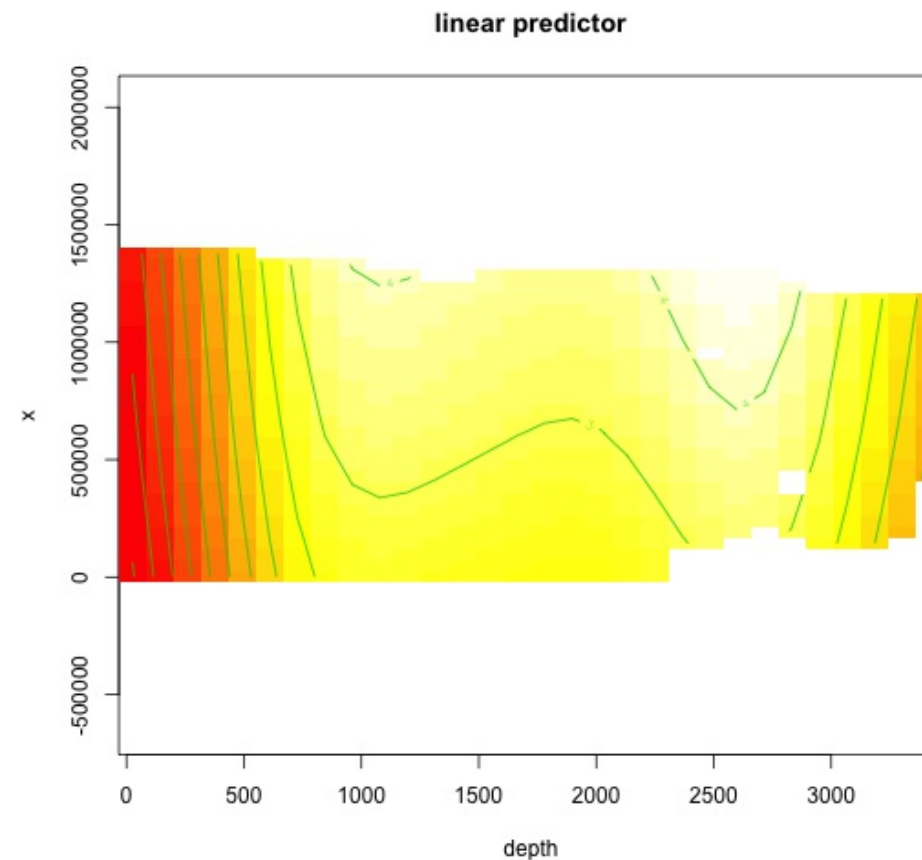
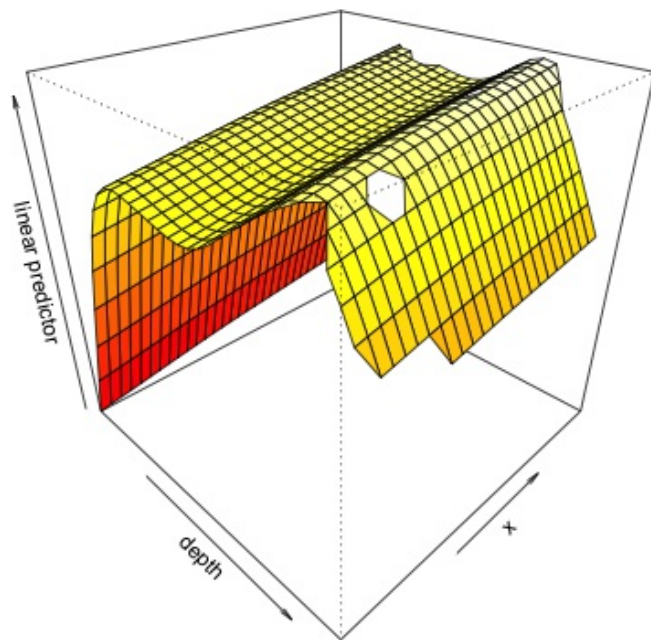
```
plot(dolphins_depth_xy, select=2, cex=2, asp=1, lwd=2, scheme=2)
```



- `scheme=2` much better for bivariate terms
- `vis.gam()` is much more general

More complex plots

```
par(mfrow=c(1,2))  
vis.gam(dolphins_depth_xy, view=c("depth","x"), too.far=0.1,  
phi=30, theta=45)  
vis.gam(dolphins_depth_xy, view=c("depth","x"),  
plot.type="contour", too.far=0.1, asp=1/1000)
```



Fitting/plotting GAMs summary

- `gam` does all the work
- very similar to `glm`
- `s` indicates a smooth term
- `plot` can give simple plots
- `vis.gam` for more advanced stuff

Prediction

What is a prediction?

- Evaluate the model, at a particular covariate combination
- Answering (e.g.) the question “at a given depth, how many dolphins?”
- Steps:
 1. evaluate the $s(\dots)$ terms
 2. move to the response scale (exponentiate? Do nothing?)
 3. (multiply any offset etc)

Example of prediction

- in maths:
 - Model: $\text{count}_i = A_i \exp(\beta_0 + s(x_i, y_i) + s(\text{Depth}_i))$
 - Drop in the values of x, y, Depth (and A)
- in R:
 - build a `data.frame` with x, y, Depth, A
 - use `predict()`

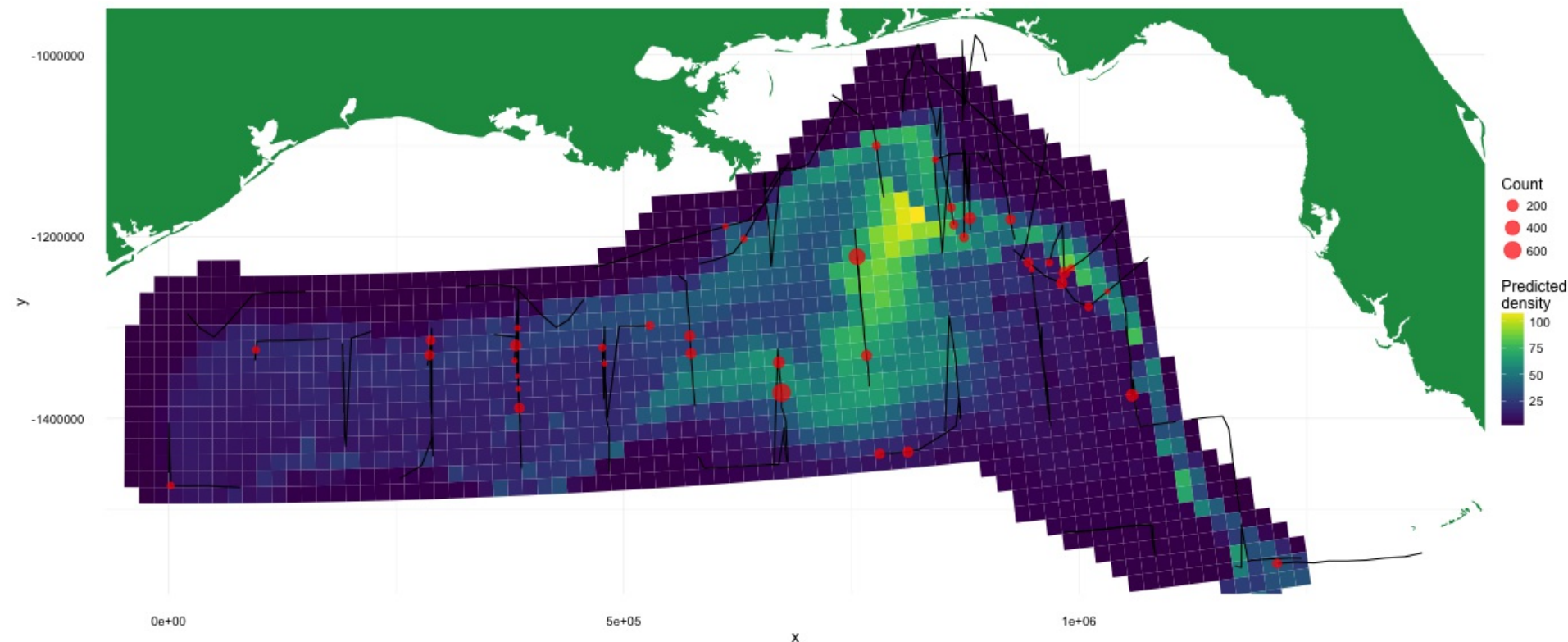
```
preds <- predict(my_model, newdat=my_data, type="response")
```

(`se.fit=TRUE` gives a standard error for each prediction)

Back to the dolphins...

Where are the dolphins?

```
dolphin_preds <- predict(dolphins_depth_xy, newdata=preddata,  
type="response")
```



(ggplot2 code included in the slide source)

Prediction summary

- Evaluate the fitted model at a given point
- Can evaluate many at once (`data.frame`)
- Don't forget the `type=...` argument!
- Obtain per-prediction standard error with `se.fit`

What about uncertainty?

Without uncertainty, we're not
doing statistics

Where does uncertainty come from?

- β : uncertainty in the spline parameters
- λ : uncertainty in the smoothing parameter
- Traditionally we've only addressed the former
- New tools let us address the latter...

Parameter uncertainty

From theory:

$$\boldsymbol{\beta} \sim N(\hat{\boldsymbol{\beta}}, \mathbf{V}_{\boldsymbol{\beta}})$$

*(caveat: the normality is only **approximate** for non-normal response)*

What does this mean? Variance for each parameter.

In mgcv: `vcov(model)` returns $\mathbf{V}_{\boldsymbol{\beta}}$.

What can we do this this?

- confidence intervals in `plot`
- standard errors using `se.fit`
- derived quantities? (see bibliography)



The lpmatrix, magic, etc

For regular predictions:

$$\hat{\eta}_p = L_p \hat{\beta}$$

form L_p using the prediction data, evaluating basis functions as we go.

(Need to apply the link function to $\hat{\eta}_p$)

But the L_p fun doesn't stop there...

[[mathematics intensifies]]

Variance and Ipmatrix

To get variance on the scale of the linear predictor:

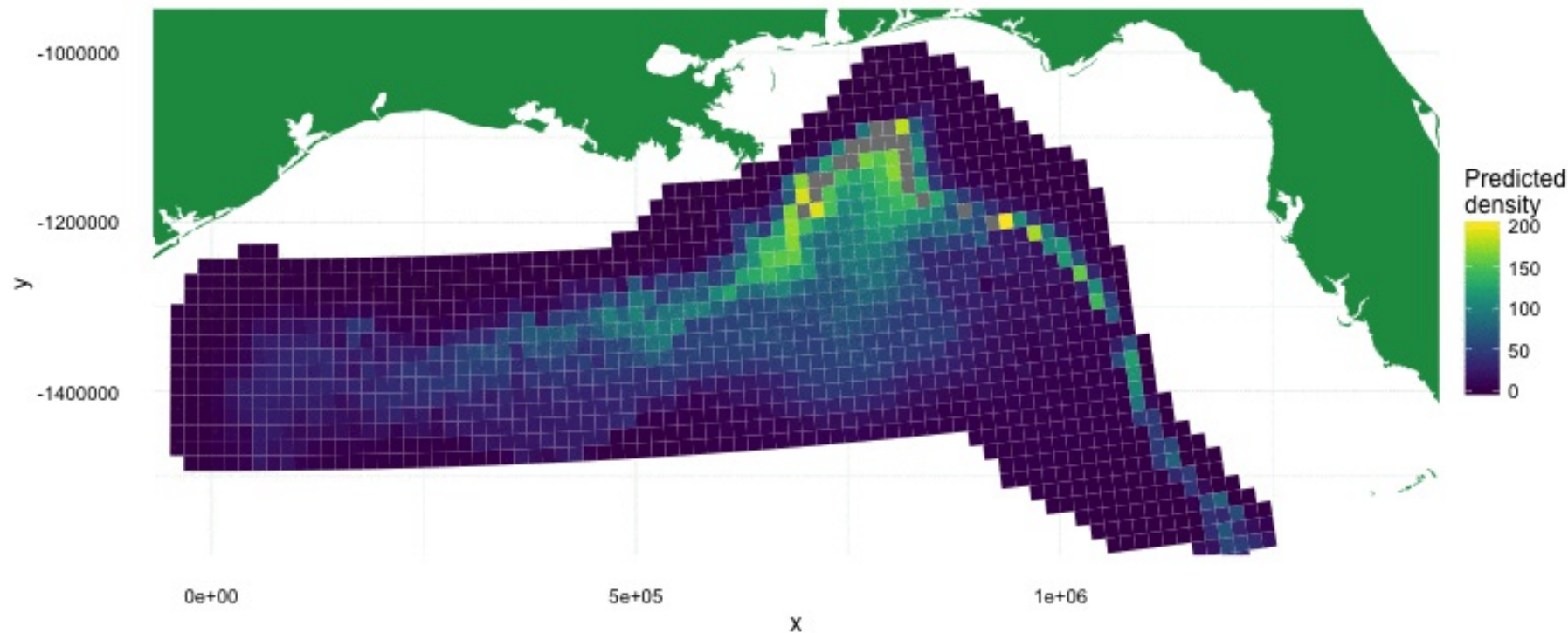
$$\mathbf{V}_{\hat{\eta}} = \mathbf{L}_p^T \mathbf{V}_{\hat{\beta}} \mathbf{L}_p$$

pre-/post-multiplication shifts the variance matrix from parameter space to linear predictor-space.

(Can then pre-/post-multiply by derivatives of the link to put variance on response scale)

Simulating parameters

- β has a distribution, we can simulate



Uncertainty in smoothing parameter

- Recent work by Simon Wood
- “smoothing parameter uncertainty corrected” version of $V_{\hat{\beta}}$
- In a fitted model, we have:
 - V_p what we got with `vcov`
 - V_c the corrected version
- Still experimental

Variance summary

- Everything comes from variance of parameters
- Need to re-project/scale them to get the quantities we need
- `mgcv` does most of the hard work for us
- Fancy stuff possible with a little maths
- Can include uncertainty in the smoothing parameter too

Okay, that was a lot of
information

Summary

- GAMs are GLMs plus some extra wiggles
- Need to make sure things are *just wiggly enough*
 - Basis + penalty is the way to do this
- Fitting looks like glm with extra $s()$ terms
- Most stuff comes down to matrix algebra, that mgcv shields you from
- To do fancy stuff, get inside the matrices