# Generalized Additive Models

Gavin L. Simpson

February, 2017

## Generalized additive models

## Generalized additive models

### Generalized additive models

- Generalized — conditions distribution of $y$ from (extended) exponential family of distributions
- Additive — assume nothing more that separate model component sum
- Models — it's a model

Earlier we looked at models that look like

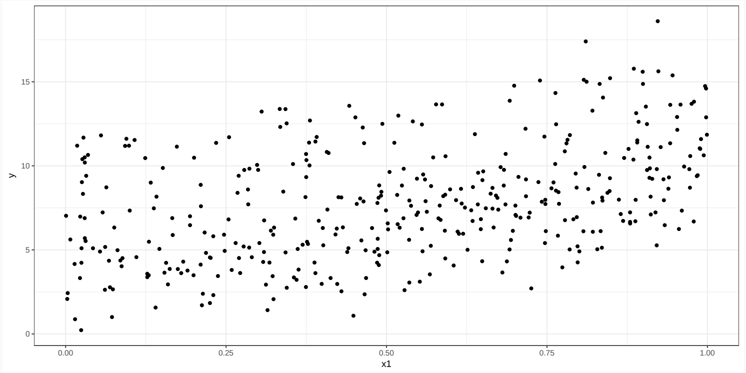
$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \varepsilon_i$$

$\hat{y}_i$ is a linear combination of covariates

GLMs extend this model to allow $y_i$ to be distributed any member of the exponential family

Conditional distribution of $y_i|\mathbf{X}$

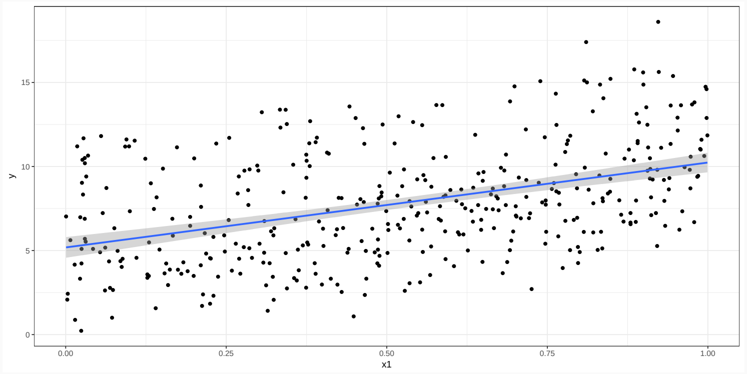
## Is this linear?



## Is this linear? Maybe?

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

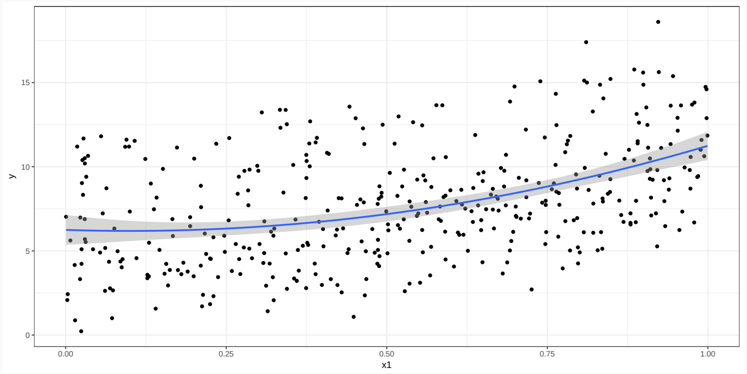
```
> ggplot(dat, aes(y=y, x=x1)) + geom_point() +
+     geom_smooth(method="lm")
```



## What can we do?

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

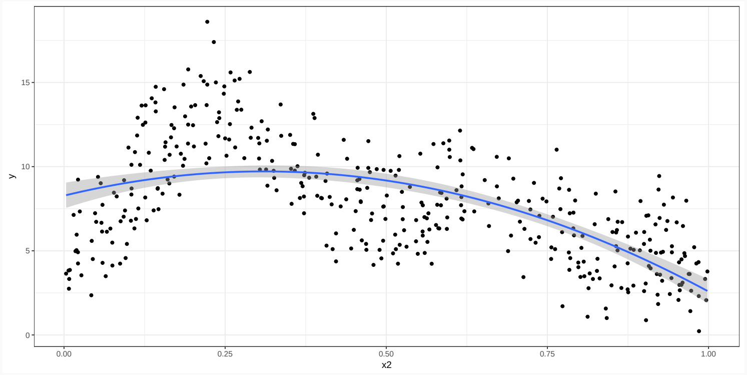
```
> ggplot(dat, aes(y=y, x=x1)) + geom_point() +
+     geom_smooth(method="lm", formula = y~poly(x, 2))
```



## How practical is this?

- 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?

```
> ggplot(dat, aes(y=y, x=x2)) + geom_point() +
+     geom_smooth(method="lm", formula = y~poly(x, 2))
```



## What does a GAM look like?

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

where $\epsilon_i \sim N(0, \sigma^2)$, $y_i \sim$ Normal (for now)

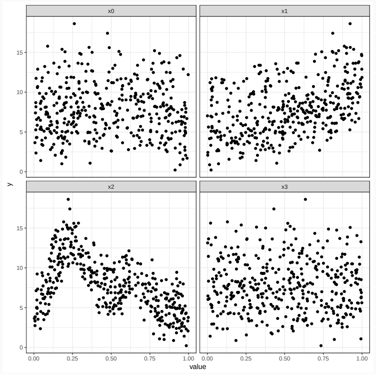
Remember that we're modelling the mean of this distribution!

Call the above equation the **linear predictor**

## What are these $s_j$?

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

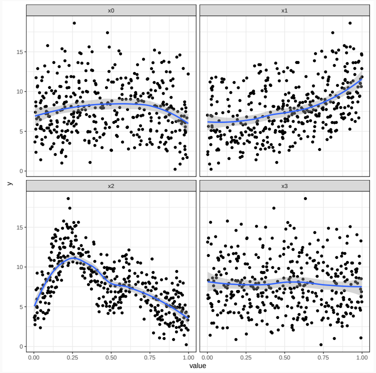
```
> spdat <- melt(dat, id.vars = c("y"))
> p <- ggplot(spdat,aes(y=y,x=value)) +
+       geom_point() +
+       facet_wrap(~variable, nrow=2)
> print(p)
```



## What are these $s_j$?

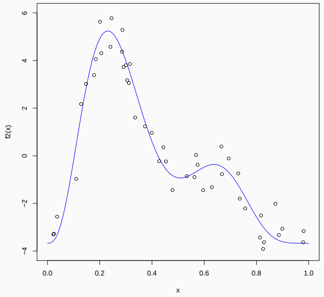
- Think $s$ = **smooth**
- Want to model the covariates flexibly
- Covariates and response not necessarily linearly related!
- Want some "wiggles"

```
> p <- p + geom_smooth()
> print(p)
```



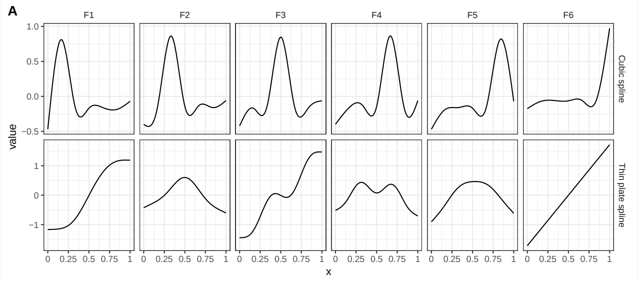
## Straight lines vs. interpolation

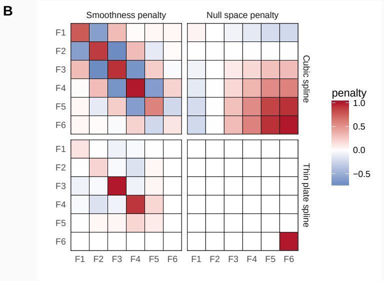
- Want a line that is “close” to all the data
- Don’t want interpolation – we know there is “error”
- Balance between interpolation and “fit”



## Splines

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





## Design matrices

We often write models as $X\boldsymbol{\beta}$

- $X$ is our data
- $\boldsymbol{\beta}$ are parameters we need to estimate

For a GAM it's the same

- $X$ has columns for each basis, evaluated at each observation
- again, this is the linear predictor

## Measuring wigglyness

Visually:

- Lots of wiggles == NOT SMOOTH
- Straight line == VERY SMOOTH

How do we do this mathematically?

- Derivatives!
- (Calculus *was* a useful class after all!)

## Wiggliness by derivatives

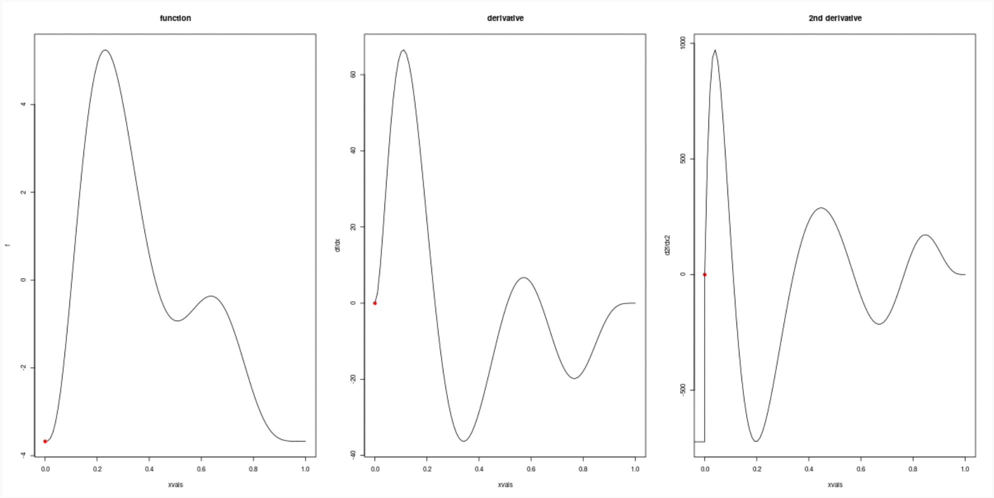


Figure 1: Animation of derivatives

## What was that grey bit?

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

Take some derivatives of the smooth and integrate them over $x$

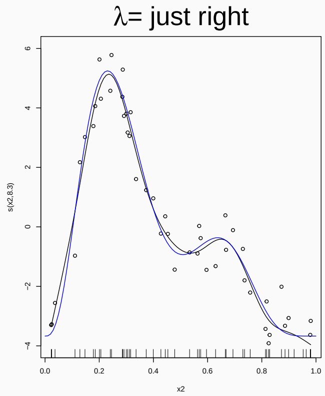
*Turns out* we can always write this as $\boldsymbol{\beta}^\mathsf{T} S \boldsymbol{\beta}$, so the $\boldsymbol{\beta}$ is separate from the derivatives

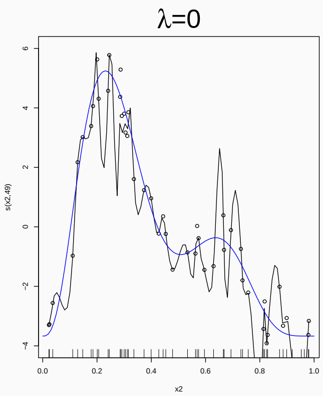
Call $S$ the **penalty matrix**

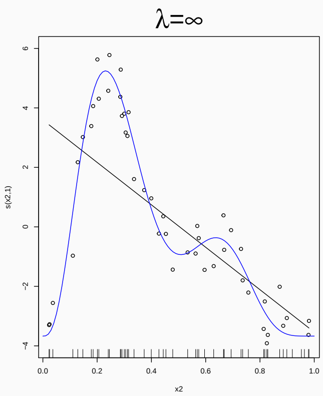
## Making wigglyness matter

- $\boldsymbol{\beta}^\intercal S \boldsymbol{\beta}$ measures wigglyness
- “Likelihood” measures closeness to the data
- Penalize closeness to the data…
- Use a **smoothing parameter** to decide on that trade-off…
- $\lambda \beta^\intercal S \beta$
- Estimate the $\beta_k$ terms but penalise objective
- “closeness to data” + penalty

## Smoothing parameter







## 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)

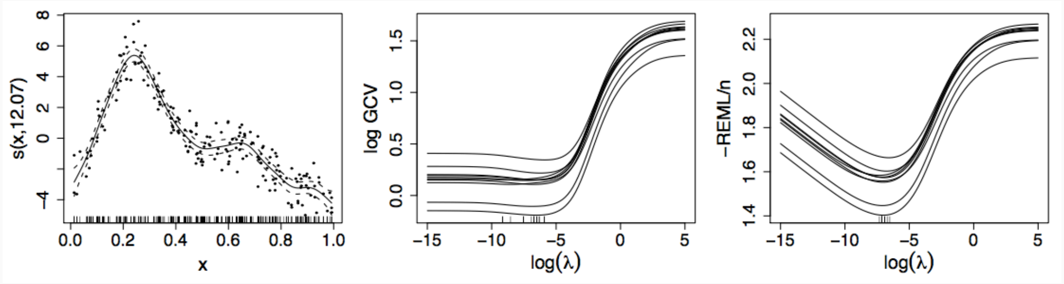


Figure 2: REML smoothness selection

## Maximum wiggliness

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

More on this in a bit...

## GAM summary

- Straight lines suck — the world is not linear — we 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 wiggliness/generality

## 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 $y_i \sim$ Normal

- linear predictor: `formula = y ~ s(x) + s(w)`
- response distribution: `family = gaussian()`
- data: `data=some_data_frame`

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

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

## Checking the basis size

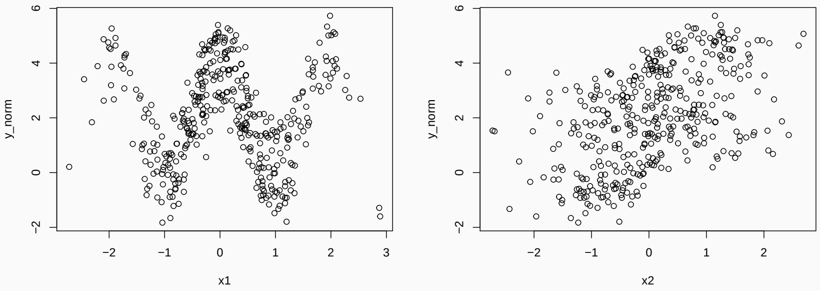
## How well does the model fit?

- Many choices: k, family, type of smoother, …
- How do we assess how well our model fits?

# Example functions

```
> set.seed(2)
> n <-  400
> x1 <-  rnorm(n)
> x2 <- rnorm(n)
> y_val <- 1 + 2 * cos(pi * x1) + 2 / (1 + exp(-5 * (x2)))
> y_norm <- y_val + rnorm(n, 0, 0.5)
```

```
> layout(matrix(1:2, ncol=2))
> plot(x1,y_norm); plot(x2,y_norm)
> layout(1)
```



## Basis size (k)

- Set `k` per term
- e.g. `s(x, k=10)` or `s(x, y, k=100)`
- Penalty removes "extra" wigglyness
    - *up to a point!*
- But computation is slower with bigger `k`

## Checking basis size — gam.check()

```
> norm_model_1 <- gam(y_norm ~ s(x1,k=4) + s(x2,k=4), method = "REML")
> gam.check(norm_model_1)
```

```
Method: REML   Optimizer: outer newton
full convergence after 8 iterations.
Gradient range [-0.0003467788,0.0005154578]
(score 736.9402 & scale 2.252304).
Hessian positive definite, eigenvalue range [0.000346021,198.5041].
Model rank =  7 / 7

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(x1) 3.000 1.002   0.125    0.00
s(x2) 3.000 2.914   1.045    0.82
```

## Checking basis size — gam.check()

```
> norm_model_2 <- gam(y_norm ~ s(x1, k=12) + s(x2, k=4), method = "REML")
> gam.check(norm_model_2)
```

```
Method: REML   Optimizer: outer newton
full convergence after 11 iterations.
Gradient range [-5.658609e-06,5.392657e-06]
(score 345.3111 & scale 0.2706205).
Hessian positive definite, eigenvalue range [0.967727,198.6299].
Model rank =  15 / 15

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(x1) 11.000 10.837   0.989    0.36
s(x2)  3.000  2.984   0.861    0.00
```

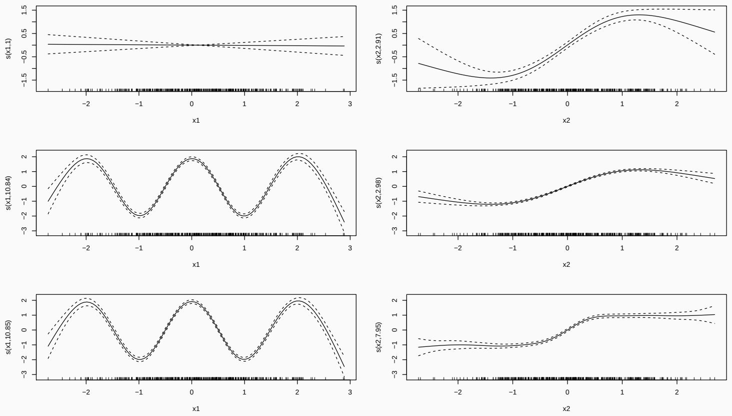
## Checking basis size — `gam.check()`

```
> norm_model_3 <- gam(y_norm~s(x1,k=12)+s(x2,k=12),method= "REML")
> gam.check(norm_model_3)
```

```
Method: REML   Optimizer: outer newton
full convergence after 8 iterations.
Gradient range [-5.103686e-05,-1.089481e-08]
(score 334.2084 & scale 0.2485446).
Hessian positive definite, eigenvalue range [2.812257,198.6868].
Model rank =  23 / 23

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(x1) 11.000 10.848   0.976    0.30
s(x2) 11.000  7.948   0.946    0.14
```

## Checking basis size — gam.check()

```
> layout(matrix(1:6,ncol=2, byrow = TRUE))
> op <- par(mar = c(5,4,2,2) + 0.1)
> plot(norm_model_1)
> plot(norm_model_2)
> plot(norm_model_3)
> par(op)
> layout(1)
```



## Model selection

## Overview

- Model selection
- Shrinkage smooths
- Shrinkage via double penalty (`select = TRUE`)
- Confidence intervals for smooths
- *p* values
- `anova()`
- AIC

## Model selection

Model (or variable) selection — and important area of theoretical and applied interest

- In statistics we aim for a balance between *fit* and *parsimony*
- In applied research we seek the set of covariates with strongest effects on $y$

We seek a subset of covariates that improves *interpretability* and *prediction accuracy*

## Shrinkage & additional penalties

## Shrinkage & additional penalties

Smoothing parameter estimation allows selection of a wide range of potentially complex functions for smooths…

But, cannot remove a term entirely from the model because the penalties used act only on the *range space* of a spline basis. The *null space* of the basis is unpenalized.

- **Null space** — the basis functions that are smooth (constant, linear)
- **Range space** — the basis functions that are wiggly

## Shrinkage & additional penalties

**mgcv** has two ways to penalize the null space, i.e. to do selection

- *double penalty approach* via `select = TRUE`
- *shrinkage approach* via special bases for thin plate and cubic splines

Other shrinkage/selection approaches are available

## Double-penalty shrinkage

$\mathsf{S}_j$ is the smoothing penalty matrix & can be decomposed as

$$\mathsf{S}_j = \mathsf{U}_j \Lambda_j \mathsf{U}_j^T$$

where $\mathsf{U}_j$ is a matrix of eigenvectors and $\Lambda_j$ a diagonal matrix of eigenvalues (i.e. this is an eigen decomposition of $\mathsf{S}_j$).

$\Lambda_j$ contains some **0**s due to the spline basis null space — no matter how large the penalty $\lambda_j$ might get no guarantee a smooth term will be suppressed completely.

To solve this we need an extra penalty…

## Double-penalty shrinkage

Create a second penalty matrix from $\mathsf{U}_j$, considering only the matrix of eigenvectors associated with the zero eigenvalues

$$\mathsf{S}_j^* = \mathsf{U}_j^* \mathsf{U}_j^{*T}$$

Now we can fit a GAM with two penalties of the form

$$\lambda_j \beta^T \mathsf{S}_j \beta + \lambda_j^* \beta^T \mathsf{S}_j^* \beta$$

Which implies two sets of penalties need to be estimated.

In practice, add `select = TRUE` to your `gam()` call

## Shrinkage

The double penalty approach requires twice as many smoothness parameters to be estimated. An alternative is the shrinkage approach, where $\mathsf{S}_j$ is replaced by

$$\tilde{\mathsf{S}}_j = \mathsf{U}_j \tilde{\Lambda}_j \mathsf{U}_j^T$$

where $\tilde{\Lambda}_j$ is as before except the zero eigenvalues are set to some small value $\epsilon$.

This allows the null space terms to be shrunk by the standard smoothing parameters.

Use `s(..., bs = "ts")` or `s(..., bs = "cs")` in mgcv

## Empirical Bayes...?

$\mathsf{S}_j$ can be viewed as prior precision matrices and $\lambda_j$ as improper Gaussian priors on the spline coefficients.

The impropriety derives from $\mathsf{S}_j$ not being of full rank (zeroes in $\Lambda_j$).

Both the double penalty and shrinkage smooths remove the impropriety from the Gaussian prior

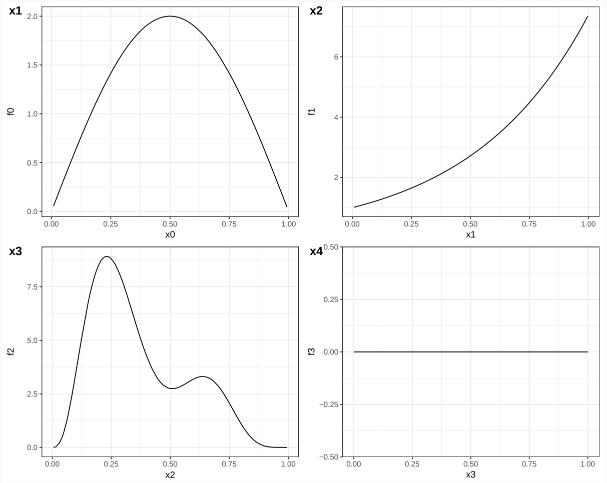
## Empirical Bayes…?

- **Double penalty** — makes no assumption as to how much to shrink the null space. This is determined from the data via estimation of $\lambda_j^*$
- **Shrinkage smooths** — assumes null space should be shrunk less than the wiggly part

Marra & Wood (2011) show that the double penalty and the shrinkage smooth approaches

- performed significantly better than alternatives in terms of *predictive ability*, and
- performed as well as alternatives in terms of variable selection

## Example

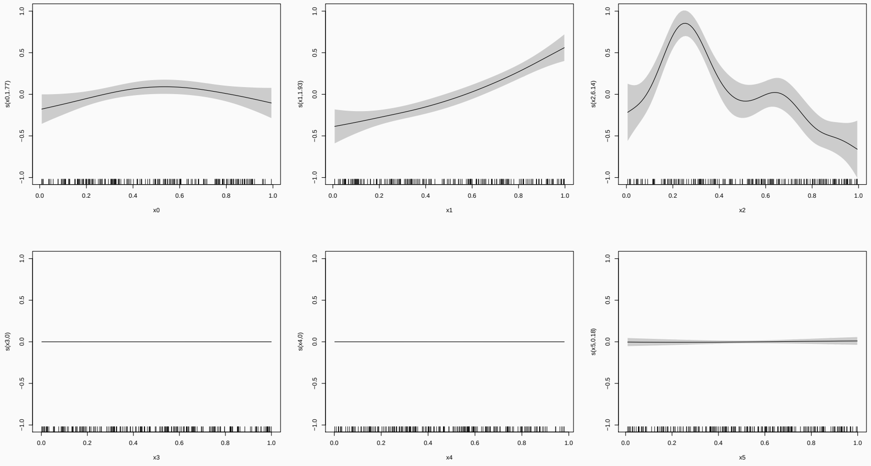


- Simulate Poisson counts
- 4 known functions
- 2 spurious covariates

## Example

```
Family: poisson
Link function: log

Formula:
y ~ s(x0) + s(x1) + s(x2) + s(x3) + s(x4) + s(x5)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.21758    0.04082   29.83   <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(x0) 1.7655082      9   5.264  0.0397 *
s(x1) 1.9271042      9  65.356  <2e-16 ***
s(x2) 6.1351238      9 156.204  <2e-16 ***
s(x3) 0.0003538      9   0.000  0.3836
s(x4) 0.0001553      9   0.000  1.0000
s(x5) 0.1756882      9   0.195  0.2963
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.545   Deviance explained = 51.6%
-REML = 430.78  Scale est. = 1         n = 200
```

## Example



## Confidence intervals for smooths

## Confidence intervals for smooths

`plot.gam()` produces approximate 95% intervals (at +/- 2 SEs)

What do these intervals represent?

Nychka (1988) showed that standard Wahba/Silverman type Bayesian confidence intervals on smooths had good **across-the-function** frequentist coverage properties.

## Confidence intervals for smooths

Marra & Wood (2012) extended this theory to the generalized case and explain where the coverage properties failed:

*Must not over-smooth too much, which happens when $\lambda_j$ are over-estimated*

Two situations where this might occur

1. where true effect is almost in the penalty null space, $\hat{\lambda}_j \to \infty$
2. where $\hat{\lambda}_j$ difficult to estimate due to highly correlated covariates
    - if 2 correlated covariates have different amounts of wiggliness, estimated effects can have degree of smoothness *reversed*

## Don’t over-smooth

*In summary, we have shown that Bayesian component-wise variable width intervals… for the smooth components of an additive model* ***should achieve close to nominal across-the-function coverage probability****, provided only that we do not over-smooth so heavily… Beyond this requirement not to over-smooth too heavily, the results appear to have rather weak dependence on smoothing parameter values, suggesting that the neglect of smoothing parameter variability should not significantly degrade interval performance.*

## Confidence intervals for smooths

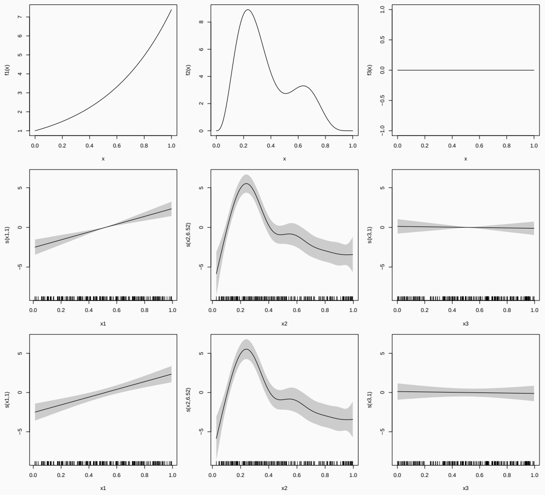
Marra & Wood (2012) suggested a solution to situation 1., namely true functions close to the penalty null space.

Smooths are normally subject to *identifiability* constraints (centred), which leads to zero variance where the estimated function crosses the zero line.

Instead, compute intervals for *j* th smooth as if it alone had the intercept; identifiability constraints go on the other smooth terms.

Use `seWithMean = TRUE` in call to `plot.gam()`

## Example



## p values for smooths

## p values for smooths

…are approximate:

1. they don't really account for the estimation of $\lambda_j$ — treated as known
2. rely on asymptotic behaviour — they tend towards being right as sample size tends to $\infty$

Also, *p* values in `summary.gam()` have changed a lot over time — all options except current default are deprecated as of `v1.18-13`.

The approach described in Wood (2006) is "*no longer recommended*"!

## p values for smooths

…are a test of **zero-effect** of a smooth term

Default *p* values rely on theory of Nychka (1988) and Marra & Wood (2012) for confidence interval coverage.

If the Bayesian CI have good across-the-function properties, Wood (2013a) showed that the *p* values have

- almost the correct null distribution
- reasonable power

Test statistic is a form of $\chi^2$ statistic, but with complicated degrees of freedom.

## p values for unpenalized smooths

The results of Nychka (1988) and Marra & Wood (2012) break down if smooth terms are unpenalized.

This include i.i.d. Gaussian random effects, (e.g. `bs = "re"`)

Wood (2013b) proposed instead a test based on a likelihood ratio statistic:

- the reference distribution used is appropriate for testing a $\mathrm{H}_0$ on the boundary of the allowed parameter space…
- …in other words, it corrects for a $\mathrm{H}_0$ that a variance term is zero.

## p values for smooths

Have the best behaviour when smoothness selection is done using `ML`, then `REML`.

Neither of these are the default, so remember to use `method = "ML"` or `method = "REML"` as appropriate

## p values for parametric terms

…are based on Wald statistics using the Bayesian covariance matrix for the coefficients.

This is the "right thing to do" when there are random effects terms present and doesn't really affect performance if there aren't.

Hence in most instances you won't need to change the default `freq = FALSE` in `summary.gam()`

## anova()

## anova()

mgcv provides an `anova()` method for `"gam"` objects:

1. Single model form: `anova(m1)`
2. Multiple model form: `anova(m1, m2, m3)`

## anova() — single model form

This differs from `anova()` methods for `"lm"` or `"glm"` objects:

- the tests are Wald-like tests as described for `summary.gam()` of a $\mathrm{H}_0$ of zero-effect of a smooth term
- these are not *sequential* tests!

## anova()

```
> b1 <- gam(y ~ x0 + s(x1) + s(x2) + s(x3), method = "REML")
> anova(b1)
```

```
Family: gaussian
Link function: identity

Formula:
y ~ x0 + s(x1) + s(x2) + s(x3)

Parametric Terms:
   df     F  p-value
x0  3 26.94 1.57e-14

Approximate significance of smooth terms:
        edf Ref.df      F  p-value
s(x1) 1.000  1.001 26.682 5.83e-07
s(x2) 6.694  7.807 18.755  < 2e-16
s(x3) 1.000  1.000  0.068    0.795
```

## anova() — multi model form

The multi-model form should really be used with care — the $p$ values are really *approximate*

For *general smooths* deviance is replaced by $-2\mathcal{L}(\hat{\beta})$

```
> b1 <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3) + s(x4) + s(x5), data = dat,
+           family=poisson, method = "ML")
> b2 <- update(b1, . ~ . - s(x3) - s(x4) - s(x5))
> anova(b2, b1, test = "LRT")
```

```
Analysis of Deviance Table

Model 1: y ~ s(x0) + s(x1) + s(x2)
Model 2: y ~ s(x0) + s(x1) + s(x2) + s(x3) + s(x4) + s(x5)
  Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
1    186.23     248.97
2    183.34     248.01 2.8959  0.96184    0.795
```

## AIC for GAMs

## AIC for GAMs

- Comparison of GAMs by a form of AIC is an alternative frequentist approach to model selection
- Rather than using the marginal likelihood, the likelihood of the $\beta_j$ *conditional* upon $\lambda_j$ is used, with the EDF replacing *k*, the number of model parameters
- This *conditional* AIC tends to select complex models, especially those with random effects, as the EDF ignores that $\lambda_j$ are estimated
- Wood et al (2015) suggests a correction that accounts for uncertainty in $\lambda_j$

$$AIC = -2l(\hat{\beta}) + 2\mathrm{tr}(\widehat{\mathcal{I}}V'_{\beta})$$

## AIC

In this example, $x_3$, $x_4$, and $x_5$ have no effects on $y$

```
> AIC(b1, b2)
```

```
         df      AIC
b1 15.03493 847.7961
b2 12.12435 842.9368
```

## References

- Marra & Wood (2011) *Computational Statistics and Data Analysis* **55** 2372–2387.
- Marra & Wood (2012) *Scandinavian journal of statistics, theory and applications* **39**(1), 53–74.
- Nychka (1988) *Journal of the American Statistical Association* **83**(404) 1134–1143.
- Wood (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC.
- Wood (2013a) *Biometrika* **100**(1) 221–228.
- Wood (2013b) *Biometrika* **100**(4) 1005–1010.

## Re-use