Checking, Selecting & Predicting with GAMs

Simon Wood
Mathematical Sciences, University of Bath, U.K.

Model checking

- Since a GAM is just a penalized GLM, residual plots should be checked, exactly as for a GLM.
- The distribution of scaled residuals should be examined, marginally, and plotted against covariates and fitted values. `residuals(model)` extracts residuals.
- `gam.check(model)` produces simple residual plots, and summary λ estimation convergence information.
- `plot(model,residuals=TRUE)` plots smooth terms with partial residuals overlaid.
- The basis dimension choices should be checked, especially if the EDF for a term is close to the basis dimension, or partial residuals seem to show lack of fit. An informal check smooths the deviance residuals w.r.t. the covariate of the smooth in question *using an increased dimension*. See `?choose.k` for more information.

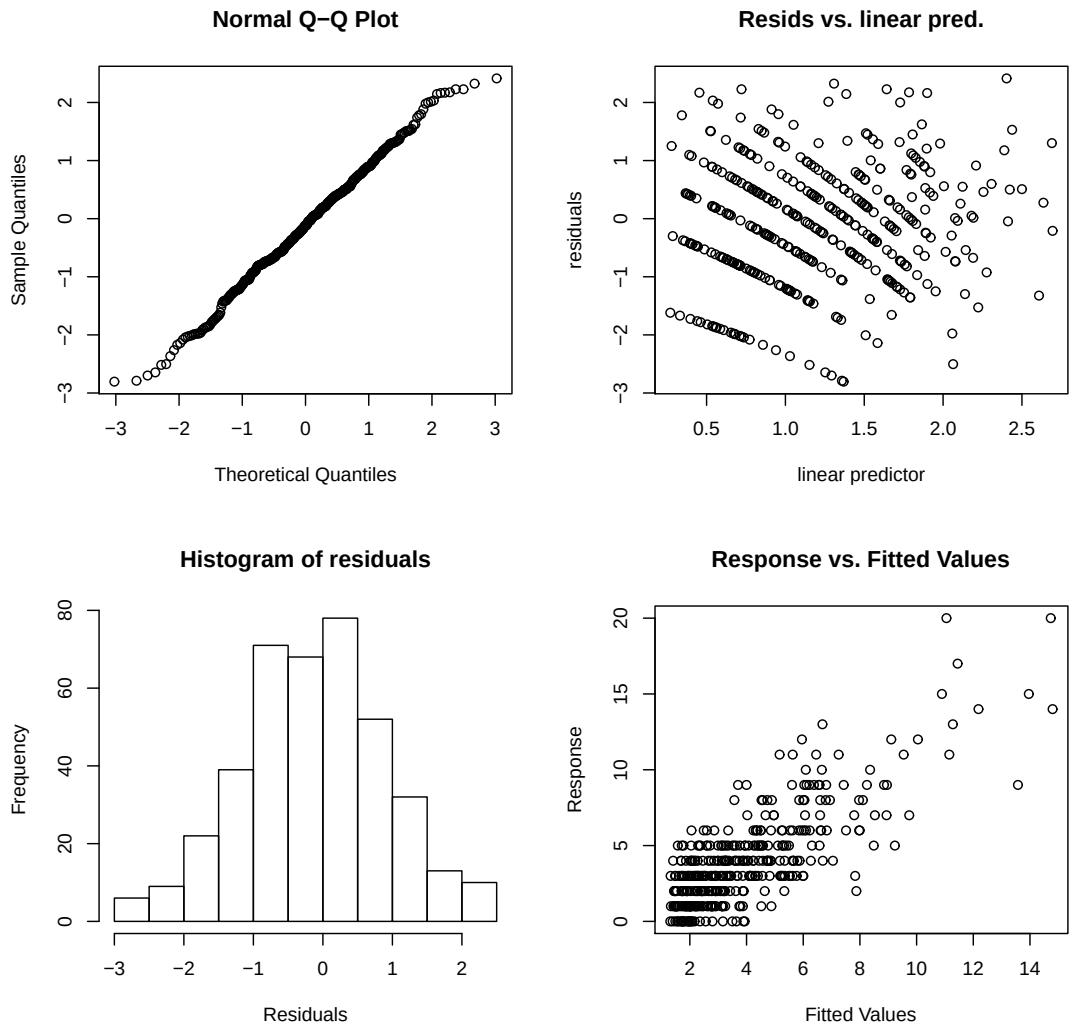
Visualization

- `plot.gam` (invoked by `plot(model)`) plots 1 and 2 dimensional smooths against predictor variables, with Bayesian confidence intervals.
- `vis.gam` (invoked with `vis.gam(model)`) plots the linear predictor or response against any two predictors, while holding the others fixed at user supplied values.
- Other plots have to be produced using `predict.gam` (invoked with `predict(model)`) and R graphics functions.

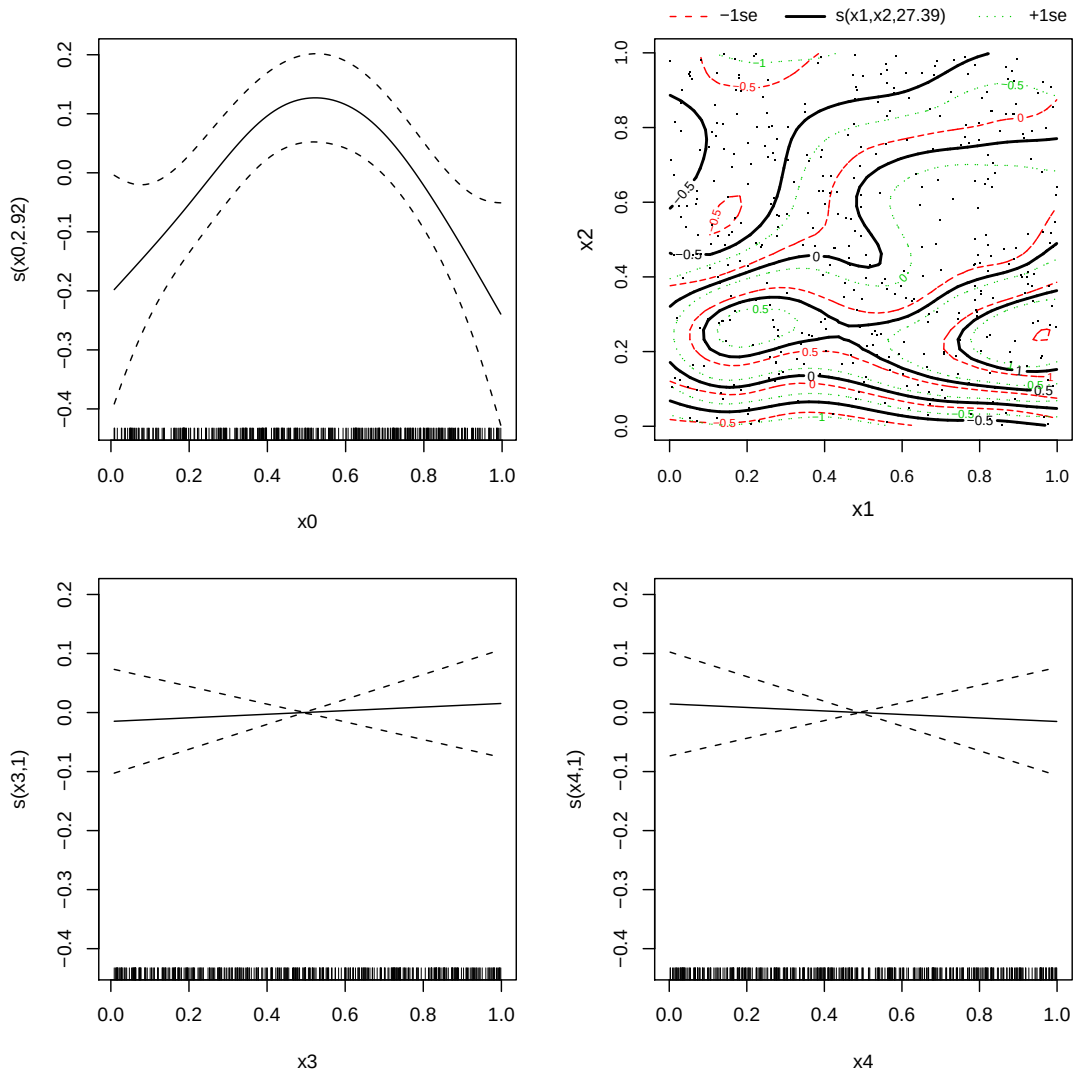
Simple checking example

```
> b<-gam(y~s(x0)+s(x1,x2,k=40)+s(x3)+s(x4),
  family=poisson,data=dat,method="REML")

> gam.check(b)

Method: REML   Optimizer: outer newton
full convergence after 8 iterations.
Gradient range [-0.0001167555,3.321004e-05]
(score 855.362 & scale 1).
Hessian positive definite, eigenvalue range
[9.66288e-05,10.52249].
```

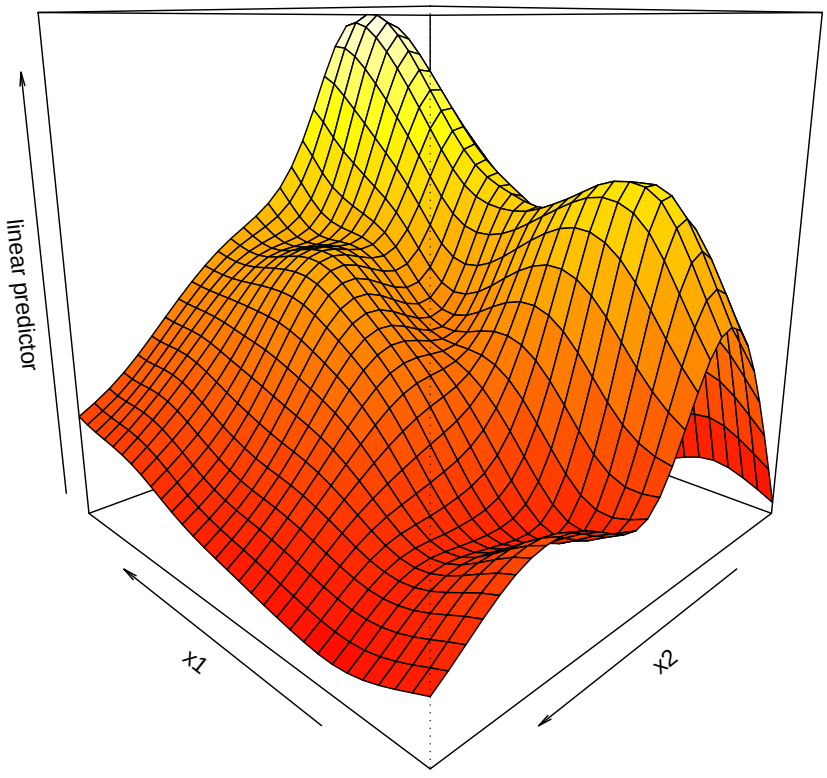
gam.check(b) plot



plot(b)



```
vis.gam(b,view=c("x1","x2"))
```



Model selection

- The greater part of model selection is performed by the $\boldsymbol{\lambda}$ estimation method.
- But $\lambda_j \to \infty$ does not generally imply $f_j \to 0$, so term inclusion/exclusion decisions are still left.
- There are a couple of obvious strategies ...
  1. Give each smooth an extra penalty, penalizing its 'fixed effect' component. Then if all the λ_j for a term $\to \infty$, the terms goes to zero.
  2. Use backward or forward selection as with a GLM, based on AIC of GCV scores, or approximate p-values for terms.
- `gam(...,select=TRUE)` implements 1. `summary` or `AIC` can be used to obtain p-values, or AIC values for 2.
- As always try to start with a reasonable model that doesn't simply 'include everything'.

Simple selection example

Continuing on from previous example, backwards selection could be based on. . .

```
> summary(b)
...
Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.20892    0.02893   41.78   <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(x0)      2.922  2.922  5.396 0.00135 **
s(x1,x2) 27.386 27.386 10.461 < 2e-16 ***
s(x3)      1.000  1.000  0.113 0.73698
s(x4)      1.000  1.000  0.109 0.74122
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

R-sq.(adj) =  0.591   Deviance explained = 55.3%
REML score = 855.36  Scale est. = 1         n = 400
```

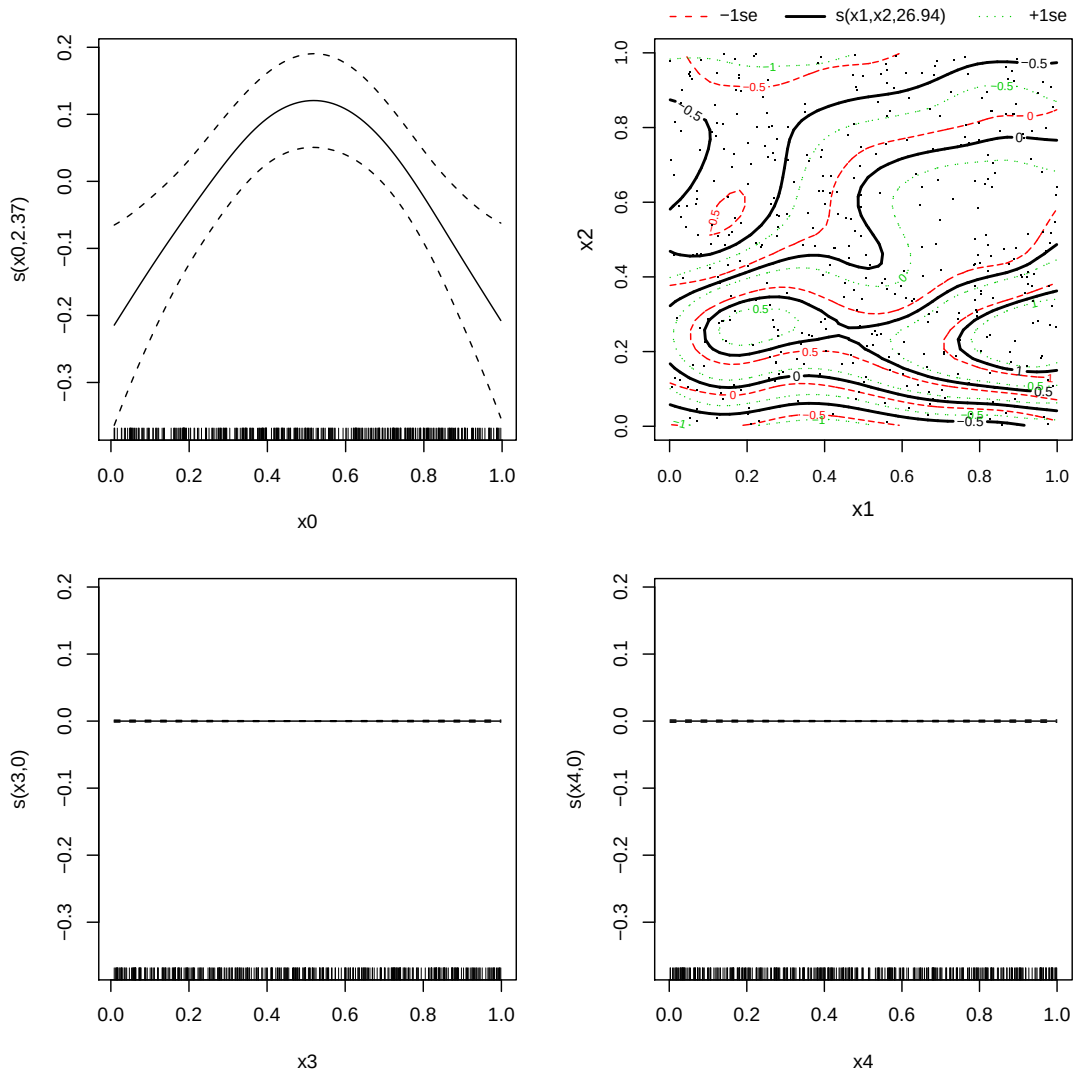
Selection via extra penalties

Giving each smooth an extra penalty on its fixed effect component (penalty null space) . . .

```
> b<-gam(y~s(x0)+s(x1,x2,k=40)+s(x3)+s(x4),
      family=poisson,data=dat,
      method="ML",select=TRUE)

> plot(b,pages=1)
```

. . . results in . . .

Model with full selection



Prediction

- Suppose we want to predict the expected response for new predictor values.
- Produce a *Prediction Matrix*, $\mathbf{X}^p$ based on the new predictor values ...
  1. ... use the new data to produce $\mathbf{X}^p$ exactly as the model fitting data were used to produce original model matrix $\mathbf{X}$
  2. ... except, that anything about the *shape* of basis functions that is data dependent, is determined from the original fit data, not the new data.
- The vector of predictions is then $\hat{\boldsymbol{\mu}}^p = \mathbf{X}^p\hat{\boldsymbol{\beta}}$, and

$$\boldsymbol{\mu}^p \sim N(\mathbf{X}^p\hat{\boldsymbol{\beta}}, \mathbf{X}^p(\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X} + \sum_j \lambda_j \mathbf{S}_j)^{-1}\mathbf{X}^{p\mathrm{T}}\phi).$$

predict.gam

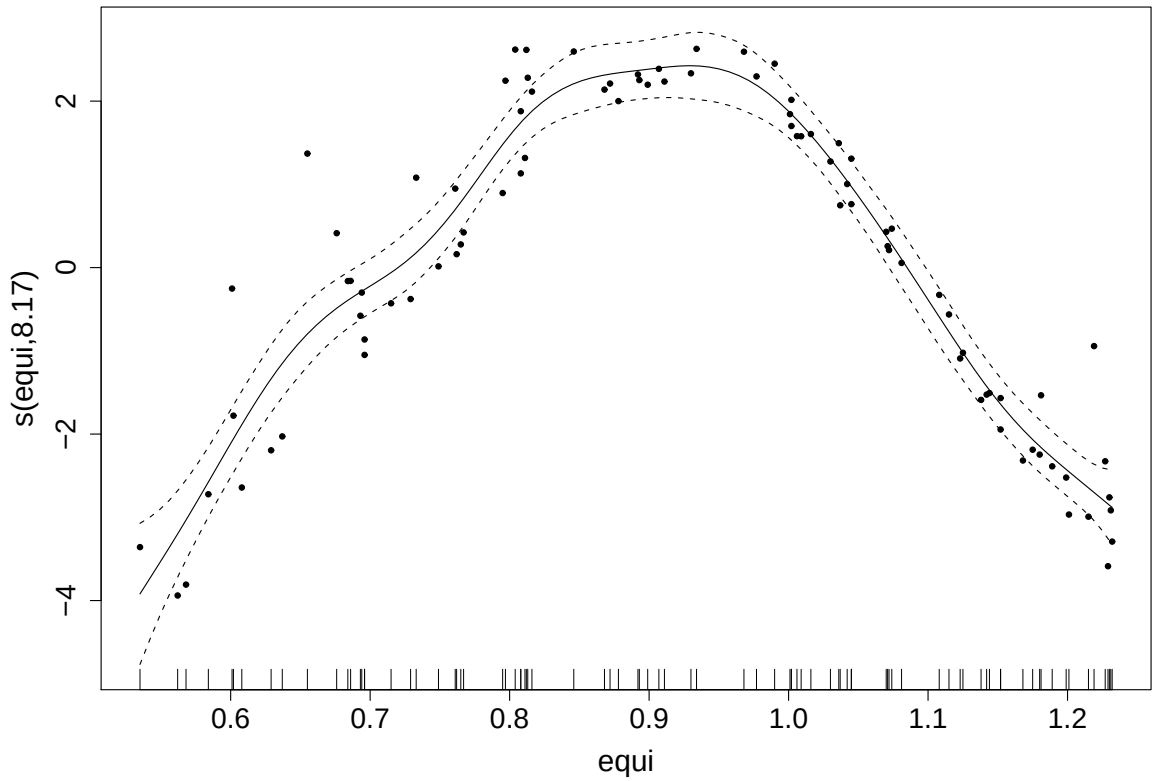
- `predict.gam(x,newdata,type,se)` is the function used for predicting from an estimated `gam` model. Main arguments are:

x	a fitted model object of class `"gam"`.
newdata	a dataframe or list containing the values of the covariates for which model predictions are required. If omitted, predictions are produced for covariate values used in fitting.
type	one of
`"response"`	return predictions (and s.e.s) on the response variable scale.
`"link"`	return predictions (and s.e.s) on the linear predictor scale.
`"terms"`	return linear predictor scale predictions (and s.e.s) split up by term.
`"lpmatrix"`	return the matrix mapping the model coefficients to the predicted linear predictor.
se	should standard errors be returned? (`TRUE/FALSE`)

NO_x prediction example

- Consider a simple smooth model for prediction of NO_X emissions from 'equivalence ratios' in an engine.

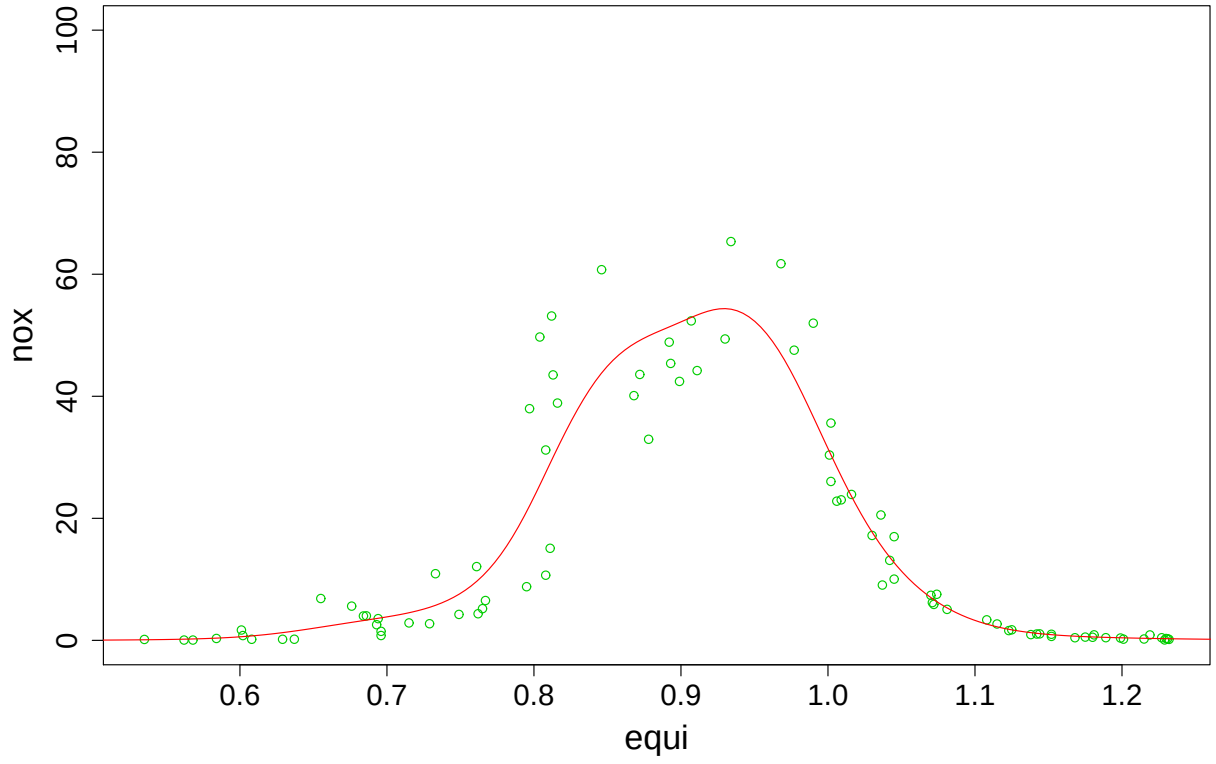
```
b <- gam(nox~s(equi,k=20),Gamma(link=log),NOX)
plot(b,residuals=TRUE,pch=19,cex=.5)}
```



NO_x response scale prediction

- Suppose we want to plot the smooth on the response scale. The following uses `predict.gam` to do this.

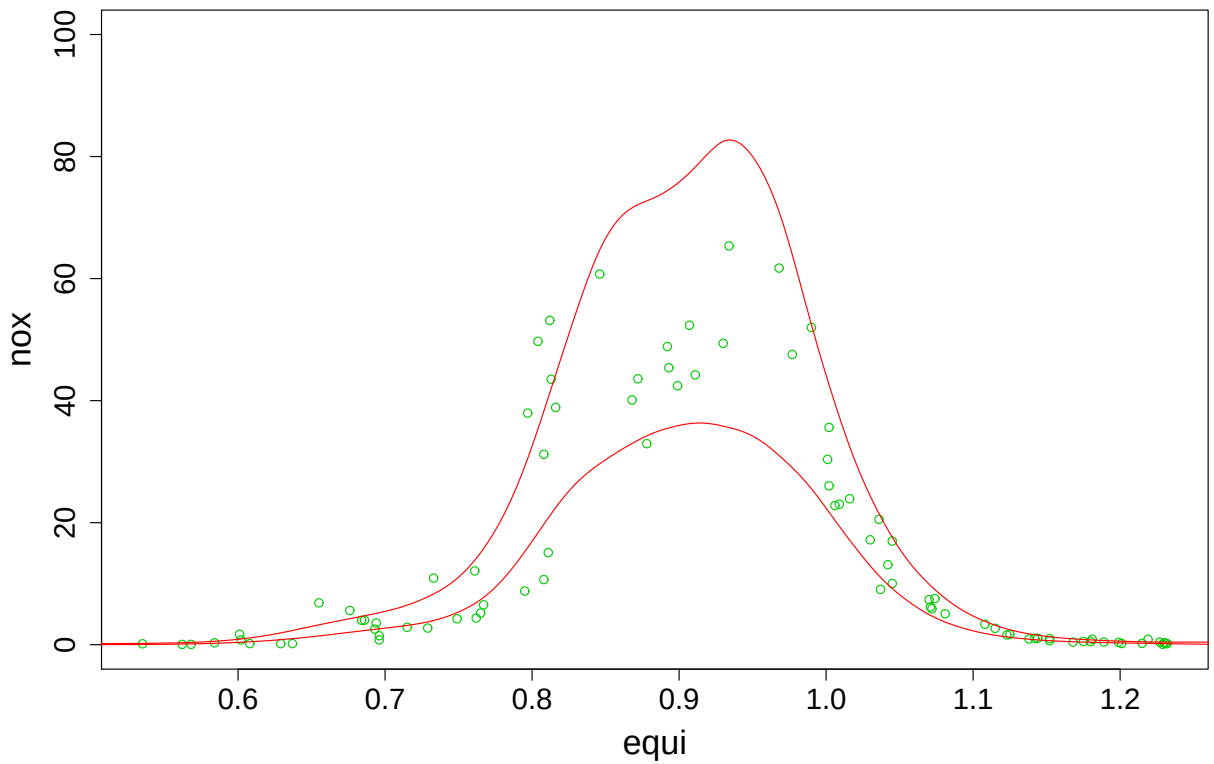
```
pd <- data.frame(equi=seq(.5,1.3,length=200))
pv <- predict(b,newdata=pd,type="response")
with(NOX,plot(equi,nox,ylim=c(0,100),col=3))
lines(pd$equi,pv,col=2)
```



NO_x response scale CI

- Normality tends to hold best on the linear predictor scale. So rather than use `se=TRUE` and `type="response"` it is usually better to do something like.

```
pv <- predict(b,newdata=pd,type="link",se=TRUE)
with(NOX,plot(equi,nox,ylim=c(0,100),col=3))
lines(pd$equi,exp(pv$fit+2*pv$se.fit),col=2)
lines(pd$equi,exp(pv$fit-2*pv$se.fit),col=2)
```



Locating the peak NO_x

- Suppose we want a CI for the `equi` value giving peak `nox`.
- We could do something crude, by finding the gradient of the smooth as a function of `equi`, and looking at where its 95% CI cuts zero.
- This is quite easy to do using `predict.gam(...,type="lpmatrix")`, but simulating from the distribution of $\boldsymbol{\beta}|\mathbf{y}$ is more direct, and more accurate in this case.

Posterior simulation

- Recall the Bayesian result that

$$\boldsymbol{\beta}|\mathbf{y} \dot{\sim} N(\hat{\boldsymbol{\beta}}, (\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X} + \sum_j \lambda_j \mathbf{S}_j)^{-1}\phi)$$

- If we plug in the estimates $\hat{\phi}$ and $\hat{\boldsymbol{\lambda}}$, then it is straightforward (and very quick) to simulate from this posterior.
- If we have a sample from the posterior, then we can obtain a sample from the posterior of any quantity that the model can predict.
- This includes the location of peak NO_x

Locating peak NO_x?

- The following R code finds the peak location to 3 significant figures

```
> eq <- seq(.6,1.2,length=1000)
> pd <- data.frame(equi=eq)
> fv <- predict(b,pd)
> eq[fv==max(fv)]
[1] 0.9291291
```

- Different model coefficients would give different answers.
- If we simulate replicate coefficient vectors from the posterior, then the peak location can be obtained for each.
- For computational efficiency first form

```
Xp <- predict(b,pd,type="lpmatrix")
```

`Xp` is the matrix mapping the model coefficients to the model predictions at the `equi` values supplied in `pd`.

Simulate from $\beta|\mathbf{y}$ and evaluate the CI

- Next simulate 1000 coefficient vectors from the posterior for β, using `mvrnorm` from the `MASS` library.

```
library(MASS)
br <- mvrnorm(1000,coef(b),vcov(b))
```

- Now we can use these draws from the posterior of β to generate draws from the posterior of the peak location.

```
> max.eq <- rep(NA,1000)
> for (i in 1:1000)
+ { fv <- Xp%*%br[i,]
+   max.eq[i] <- eq[fv==max(fv)]
+ }
```

- From which a CI is easily obtained

```
> ci <- quantile(max.eq,c(.025,.975))
> ci
     2.5%     97.5%
0.8552553 0.9561562
```

Remarks

- Notice how this is much faster than bootstrapping, to get CIs for non-linear functionals of the model.
- For linear functionals the `lpmatrix` and model covariance matrix can be used to find the posterior directly, without simulation.
- Everything has been presented conditional on the smoothing parameters... this is not always satisfactory but can be avoided — see Wood (2006) *Generalized Additive Models: An introduction with R* (order now for Christmas).