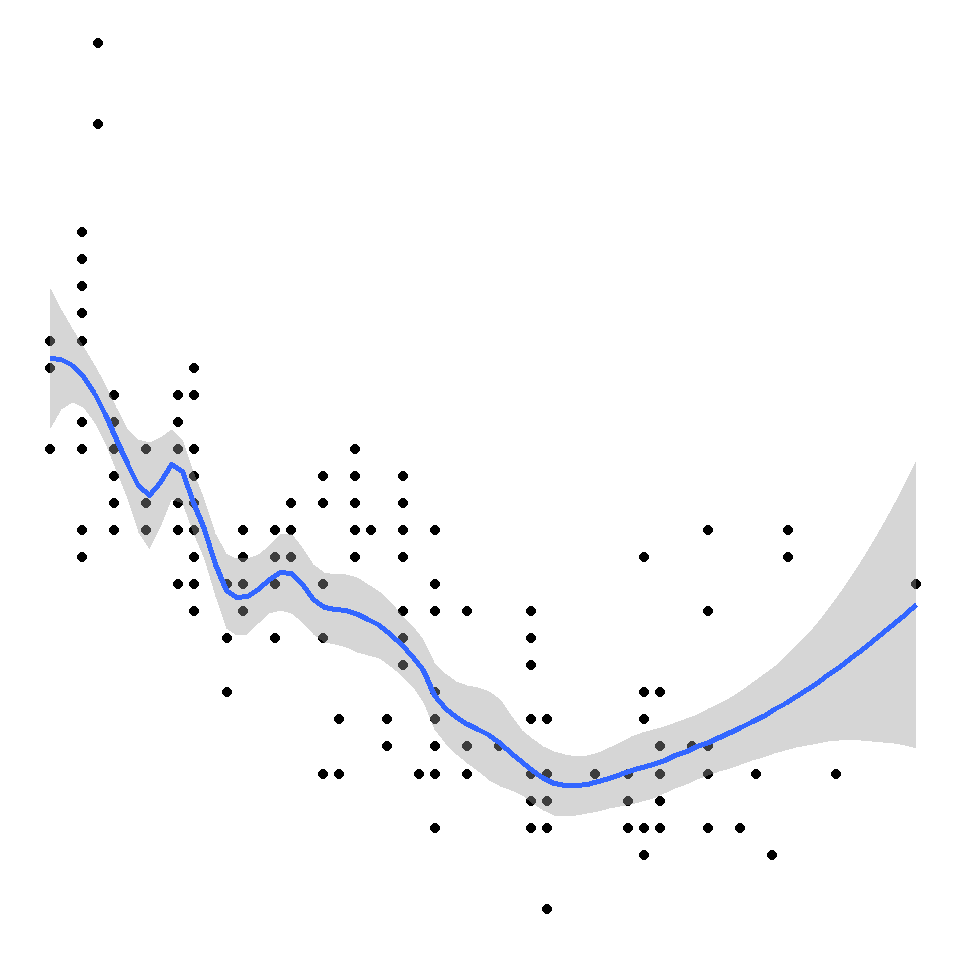
[environmentalcomputing.net](http://environmentalcomputing.net/intro-to-gams/)

**Generalised additive models (GAMs): an introduction**

13-16 minutes

Many data in the environmental sciences do not fit simple linear models and are best described by “wiggly models”, also known as Generalised Additive Models (GAMs).



Let’s start with a famous [tweet](https://twitter.com/ucfagls/status/842444686513991680) by one Gavin Simpson, which amounts to:  
1. GAMs are just GLMs  
2. GAMs fit wiggly terms  
3. use + s(x) not x in your syntax  
4. use method = "REML"  
5. always look at gam.check()

This is basically all there is too it – an extension of [generalised linear models (GLMs)](http://environmentalcomputing.net/generalised-linear-models/) with a smoothing function. Of course, there may be many sophisticated things going on when you fit a model with smooth terms, but you only need to understand the rationale and some basic theory. There are also lots of what would be apparently magic things happening when we try to understand what is under the hood of say lmer or glmer, but we use them all the time without reservation!

**GAMs in a nutshell**

Let’s start with an equation for a Gaussian [linear model](http://environmentalcomputing.net/linear-regression/):

y=β0+x1β1+ε,ε∼N(0,σ2)

What changes in a GAM is the presence of a smoothing term:

y=β0+f(x1)+ε,ε∼N(0,σ2)

This simply means that the contribution to the linear predictor is now some function f. This is not that dissimilar conceptually to using a quadratic (x12) or cubic term (x13) as your predictor.

The function f can be something more funky or kinky – here, we’re going to focus on splines. In the old days, it might have been something like piecewise linear functions.

You can have combinations of linear and smooth terms in your model, for example

y=β0+x1β1+f(x2)+ε,ε∼N(0,σ2)

or we can fit generalised distributions and random effects, for example

ln(y)=β0+f(x1)+ε,ε∼Poisson(λ)

ln(y)=β0+f(x1)+z1γ+ε,ε∼Poisson(λ),γ∼N(0,Σ)

**A simple example**

Lets try a simple example. First, let’s create a data frame and fill it with some simulated data with an obvious non-linear trend and compare how well some models fit to that data.

x <- seq(0, pi \* 2, 0.1)

sin\_x <- sin(x)

y <- sin\_x + rnorm(n = length(x), mean = 0, sd = sd(sin\_x / 2))

Sample\_data <- data.frame(y,x)

library(ggplot2)

ggplot(Sample\_data, aes(x, y)) + geom\_point()

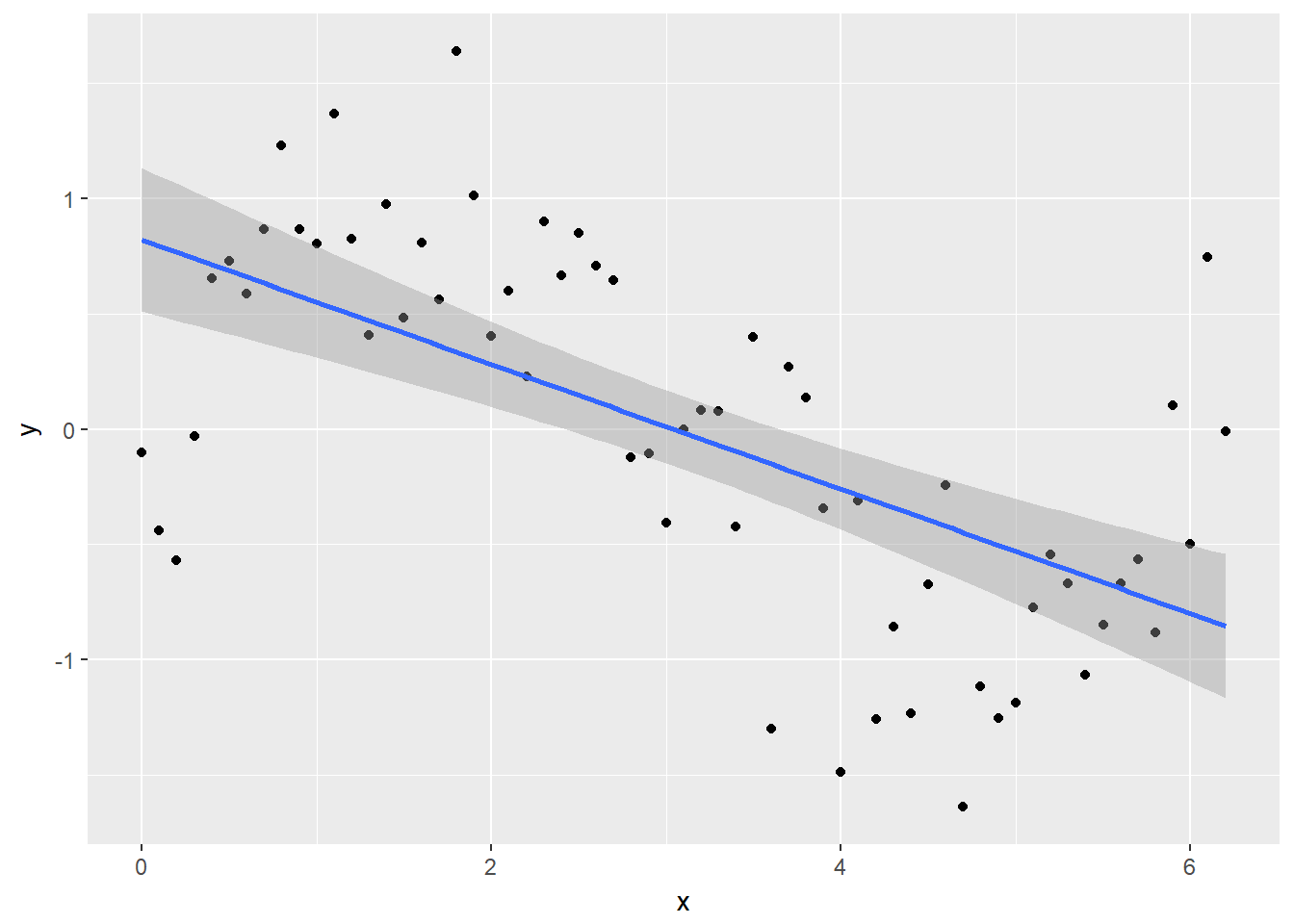


Try fitting a normal linear model:

lm\_y <- lm(y ~ x, data = Sample\_data)

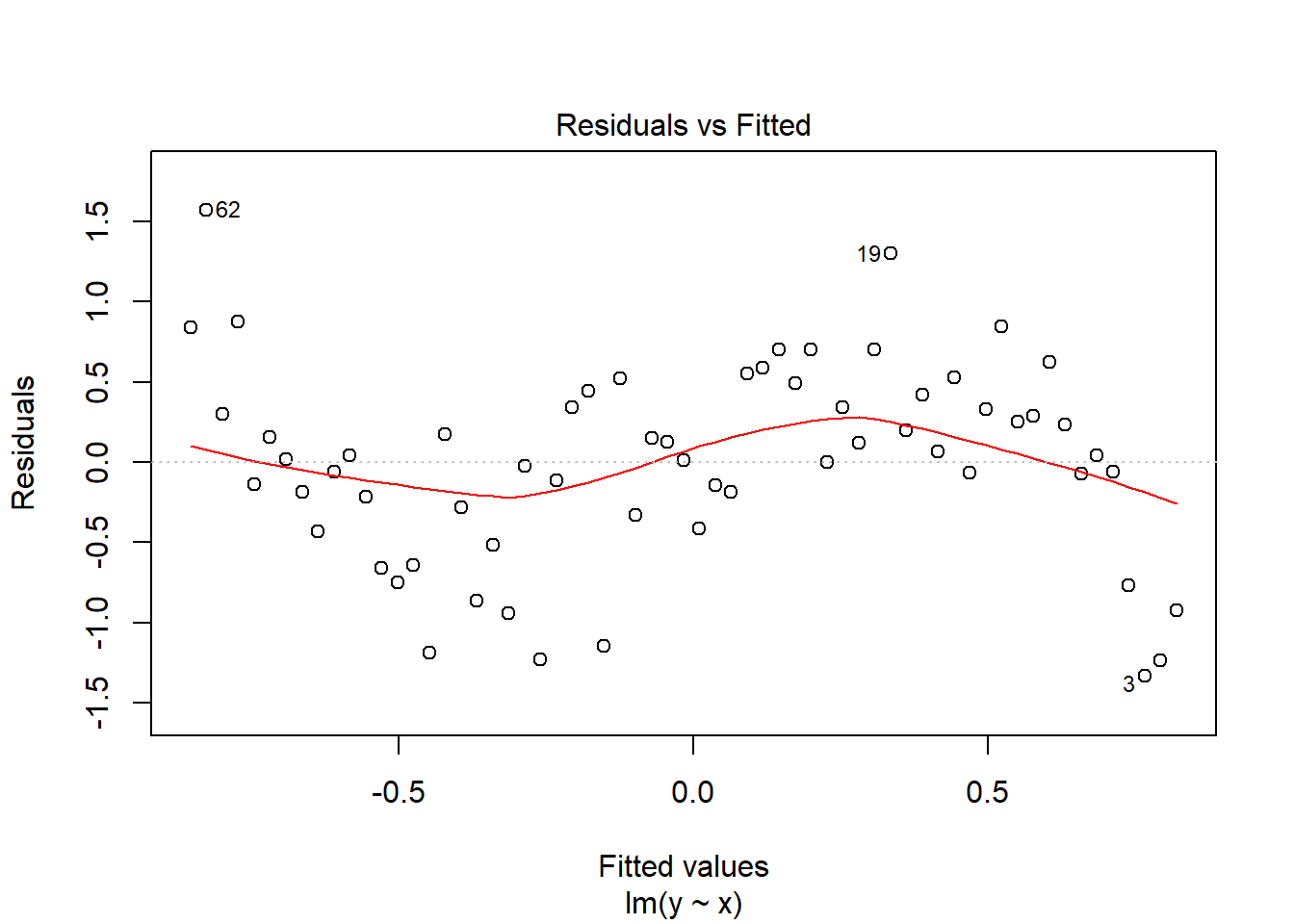
and plotting the fitted line with data using geom\_smooth in ggplot

ggplot(Sample\_data, aes(x, y)) + geom\_point() + geom\_smooth(method = lm)



Looking at the plot or summary(lm\_y), you might think the model fits nicely, but look at the residual plot – eek!

plot(lm\_y, which = 1)



Clearly, the residuals are not evenly spread across values of x, and we need to consider a better model.

**Running the analysis**

Before we consider a GAM, we need to load the package [mgcv](https://cran.r-project.org/web/packages/mgcv/index.html) – *the* choice for running GAMs in R.

library(mgcv)

To run a GAM, we use:

gam\_y <- gam(y ~ s(x), method = "REML")

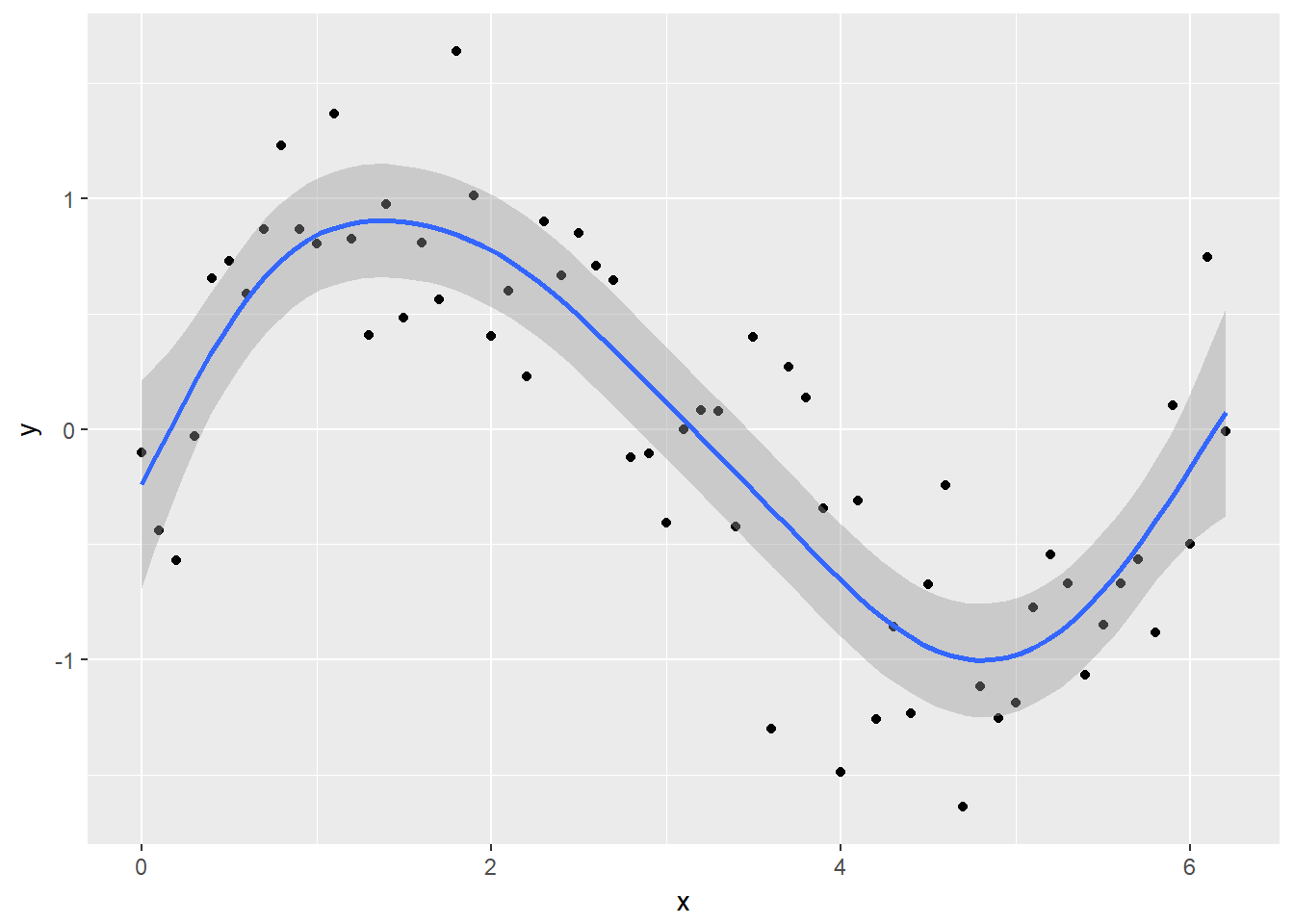
To extract the fitted values, we can use predict just like normal:

x\_new <- seq(0, max(x), length.out = 100)

y\_pred <- predict(gam\_y, data.frame(x = x\_new))

But for simple models, we can also utilise the method = argument in geom\_smooth, specifying the model formula.

ggplot(Sample\_data, aes(x, y)) + geom\_point() + geom\_smooth(method = "gam", formula = y ~s(x))

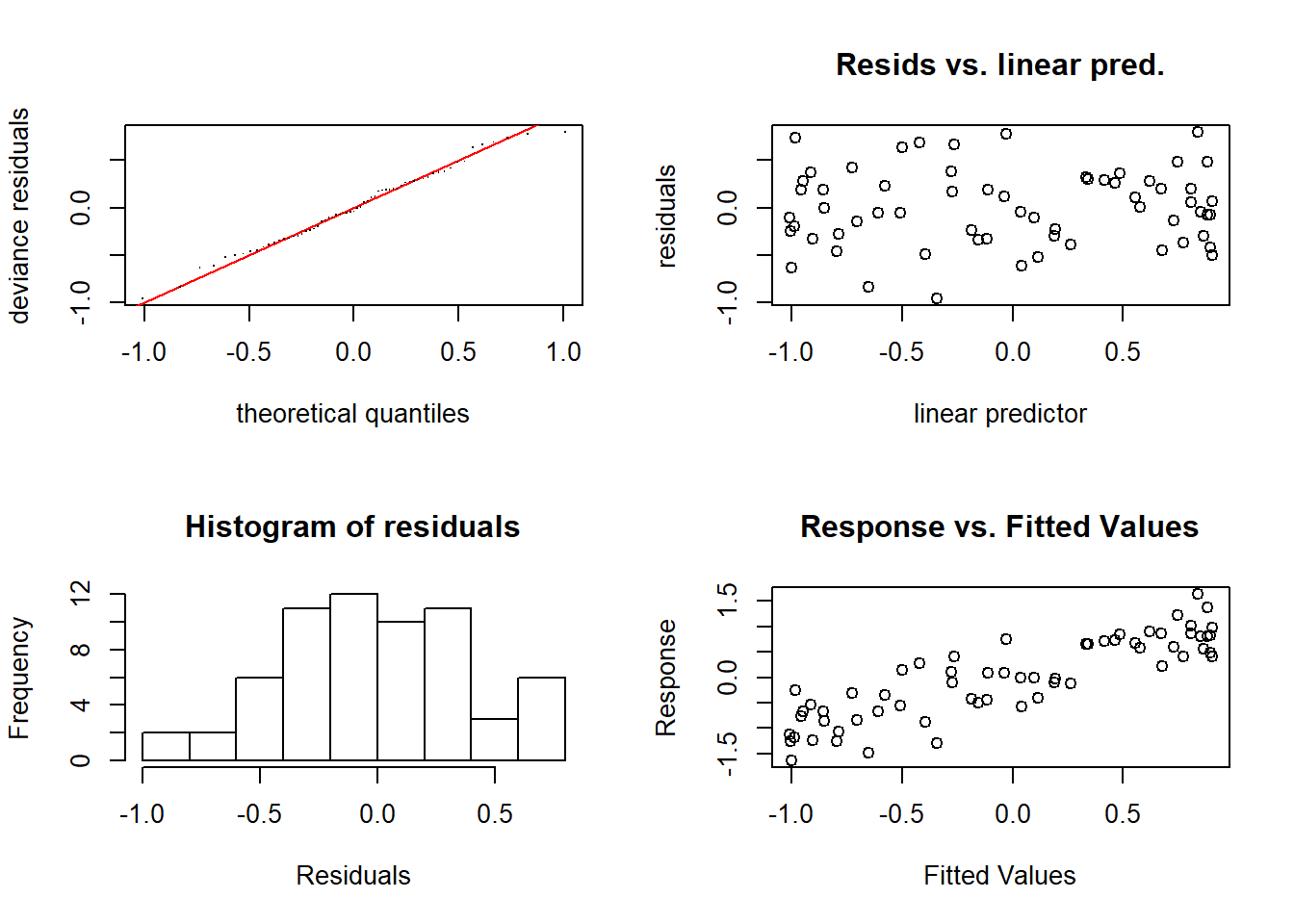


You can see the model is better fit to the data, but always check the diagnostics.

check.gam is quick and easy to view the residual plots.

par(mfrow = c(2,2))

gam.check(gam\_y)



##

## Method: REML Optimizer: outer newton

## full convergence after 6 iterations.

## Gradient range [-2.37327e-09,1.17425e-09]

## (score 44.14634 & scale 0.174973).

## Hessian positive definite, eigenvalue range [1.75327,30.69703].

## Model rank = 10 / 10

##

## 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(x) 9.00 5.76 1.19 0.9

Using summary with the model object will give you the significance of the smooth term (along with any parametric terms, if you’ve included them), along with the variance explained. In this example, a pretty decent fit. The ‘edf’ is the estimated degrees of freedom – essentially, the larger the number, the more wiggly the fitted model. Values of around 1 tend to be close to a linear term. You can read about penalisation and shrinkage for more on what the edf reflects.

summary(gam\_y)

##

## Family: gaussian

## Link function: identity

##

## Formula:

## y ~ s(x)

##

## Parametric coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) -0.01608 0.05270 -0.305 0.761

##

## Approximate significance of smooth terms:

## edf Ref.df F p-value

## s(x) 5.76 6.915 23.38 <2e-16 \*\*\*

## ---

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

##

## R-sq.(adj) = 0.722 Deviance explained = 74.8%

## -REML = 44.146 Scale est. = 0.17497 n = 63

**Smooth terms**

As mentioned above, we’ll focus on splines, as they are the smooth functions that are most commonly implemented (and are pretty quick and stable). So what was actually going on when we specified s(x)?

Well, this is where we say we want to fit y as a linear function of some set of functions of x. The default in mgcv is a thin plate regression spline – the two common ones you’ll probably see are these, and cubic regression splines. Cubic regression splines have the traditional *knots* that we think of when we talk about splines – they’re evenly spread across the covariate range in this case. We’ll just stick to thin plate regression splines, since I figure Simon made them the default for a reason,

**Basis functions** OK, so here’s where we see what the wiggle bit is really made of. We’ll start with the fitted model, then we’ll look at it from first principles (not really). Remembering that the smooth term is the sum of some number of functions (I’m not sure how well this equation really represents the smooth term, but you get the point),

f(x1)=∑j=1kbj(x1)βj

First we extract the set of *basis functions* (that is, the bj(xj) part of the smooth term). Then we can plot say the first and second basis functions.

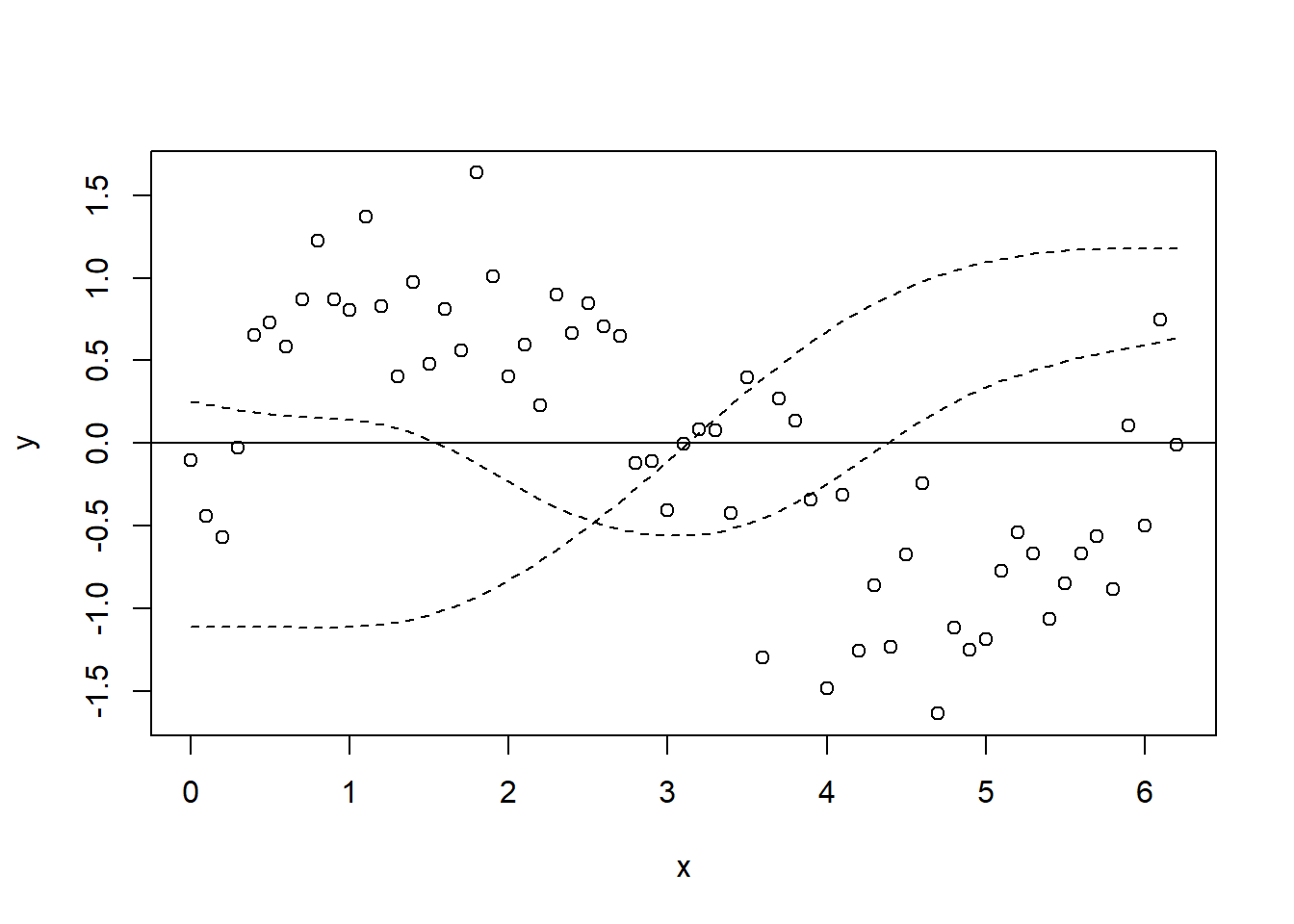
model\_matrix <- predict(gam\_y, type = "lpmatrix")

plot(y ~ x)

abline(h = 0)

lines(x, model\_matrix[, "s(x).1"], type = "l", lty = 2)

lines(x, model\_matrix[, "s(x).2"], type = "l", lty = 2)



Let’s plot all of the basis functions now, and then add that to the predictions from the GAM (y\_pred) on top again.

plot(y ~ x)

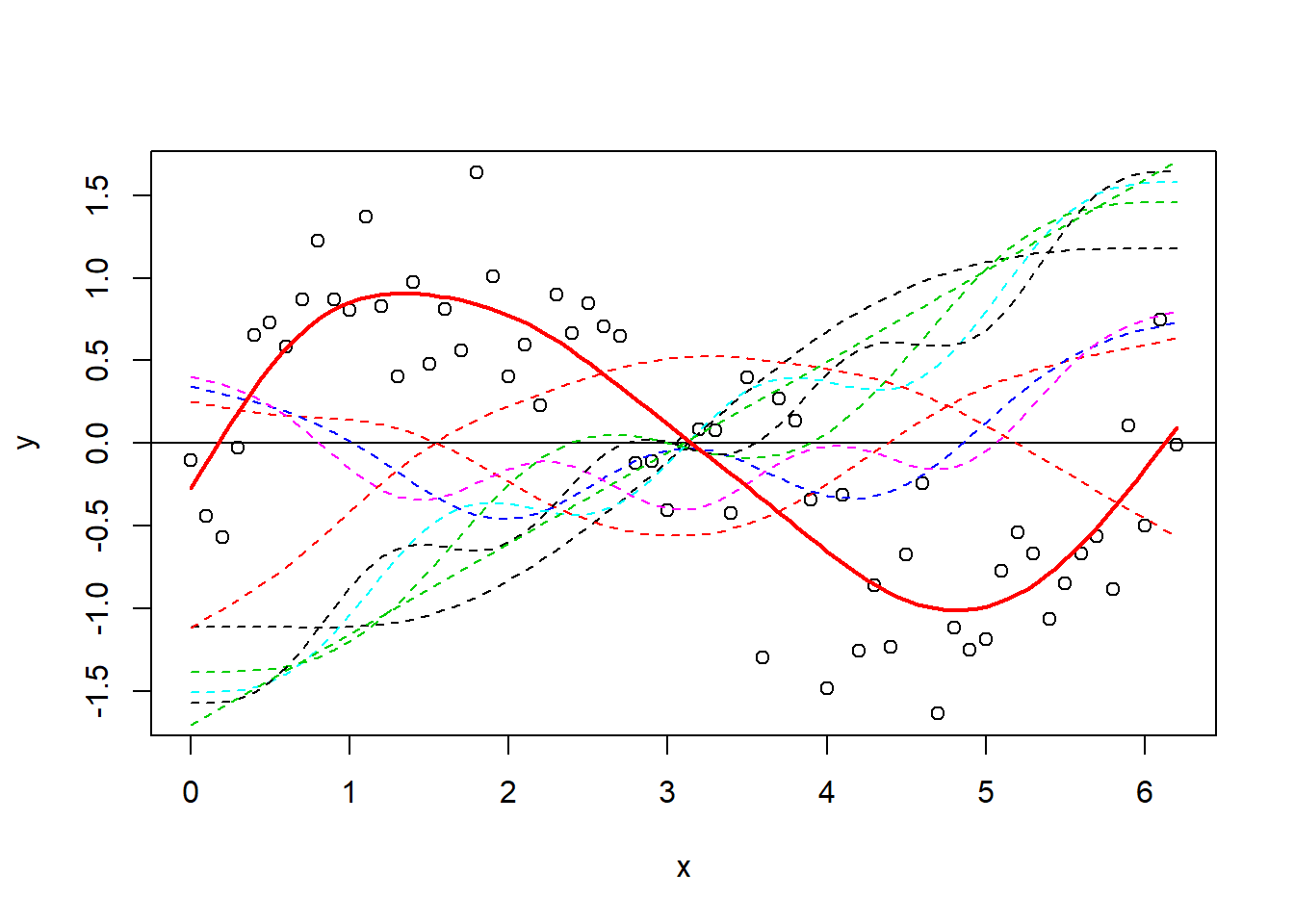
abline(h = 0)

x\_new <- seq(0, max(x), length.out = 100)

y\_pred <- predict(gam\_y, data.frame(x = x\_new))

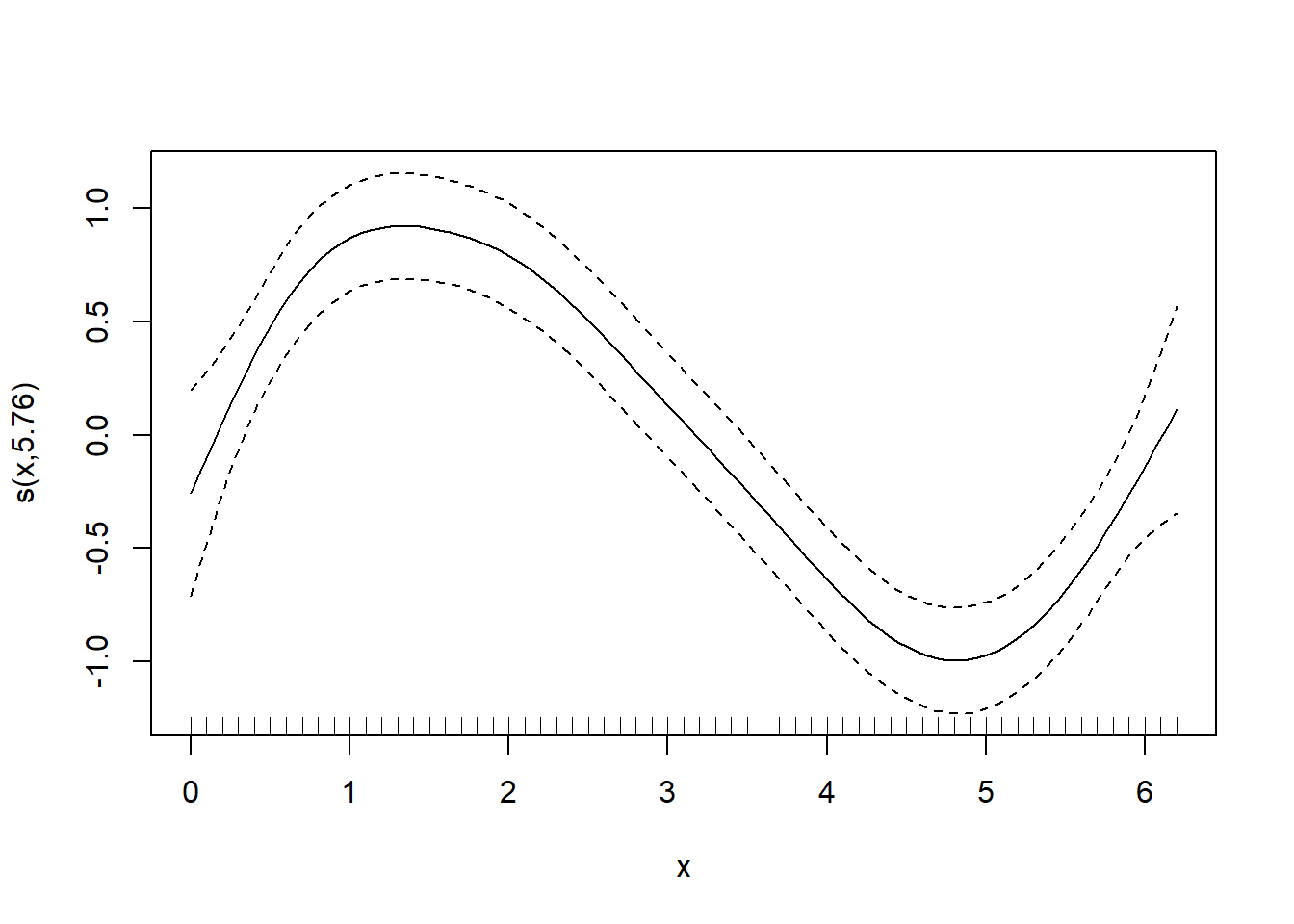
matplot(x, model\_matrix[,-1], type = "l", lty = 2, add = T)

lines(y\_pred ~ x\_new, col = "red", lwd = 2)



Now, it’s difficult at first to see what has happened, but it’s easiest to think about it like this – each of those dotted lines represents a function (bj) for which gam estimates a coefficient (βj), and when you sum them you get the contribution for the corresponding f(x) (i.e. the previous equation). It’s nice and simple for this example, because we model y only as a function of the smooth term, so it’s fairly relatable. As an aside, you can also just use plot.gam to plot the smooth terms.

plot(gam\_y)



OK, now let’s look a little closer at how the basis functions are constructed. You’ll see that the construction of the functions is separate to the response data. Just to prove it, we’ll use smoothCon.

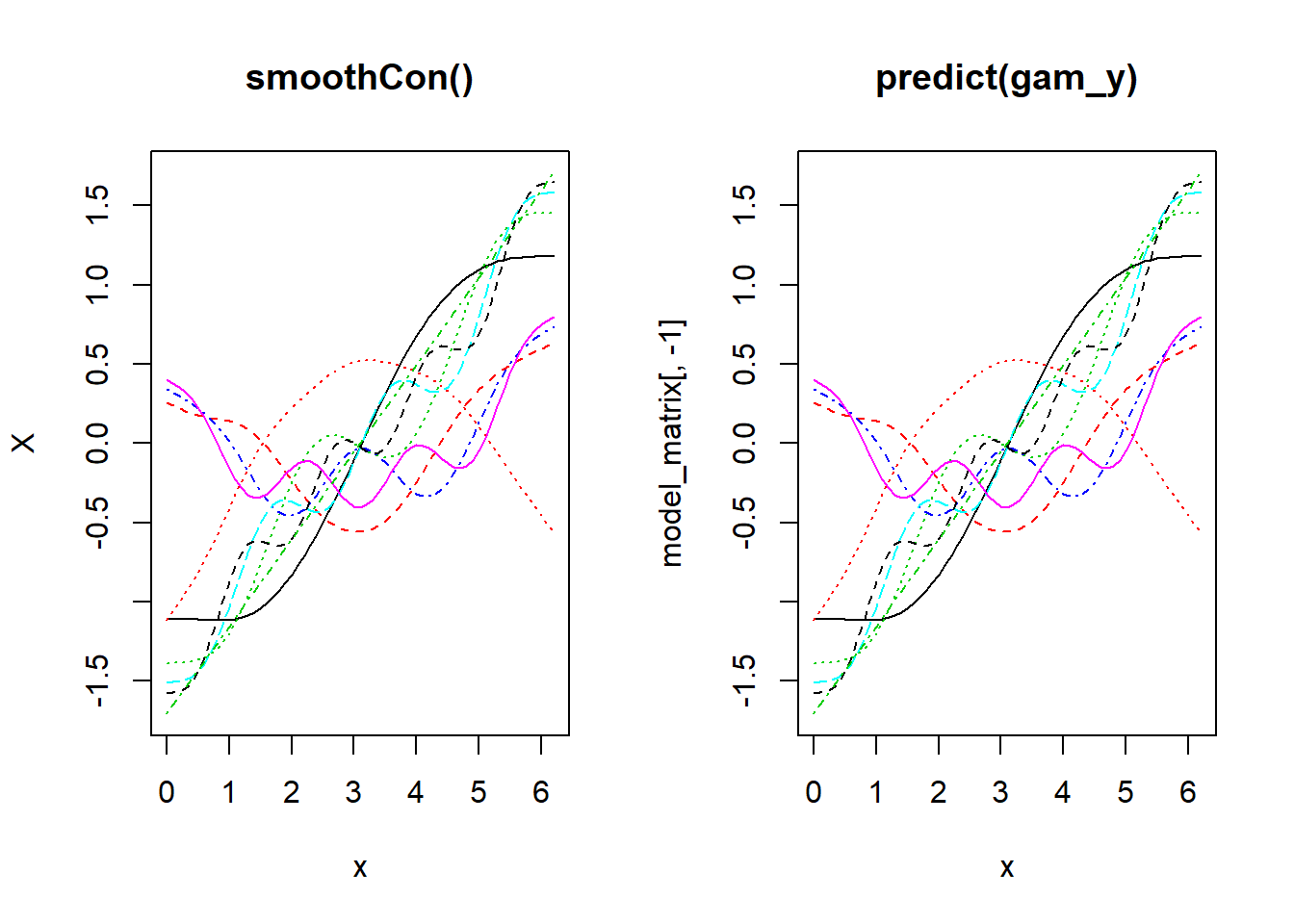
x\_sin\_smooth <- smoothCon(s(x), data = data.frame(x), absorb.cons = TRUE)

X <- x\_sin\_smooth[[1]]$X

par(mfrow = c(1,2))

matplot(x, X, type = "l", main = "smoothCon()")

matplot(x, model\_matrix[,-1], type = "l", main = "predict(gam\_y)")



And now to prove that you can go from the basis functions and the estimated coefficients to the fitted smooth term. Again note that this is simplified here because the model is just one smooth term. If you had more terms, we would need to add up all of the terms in the linear predictor.

betas <- gam\_y$coefficients

linear\_pred <- model\_matrix %\*% betas

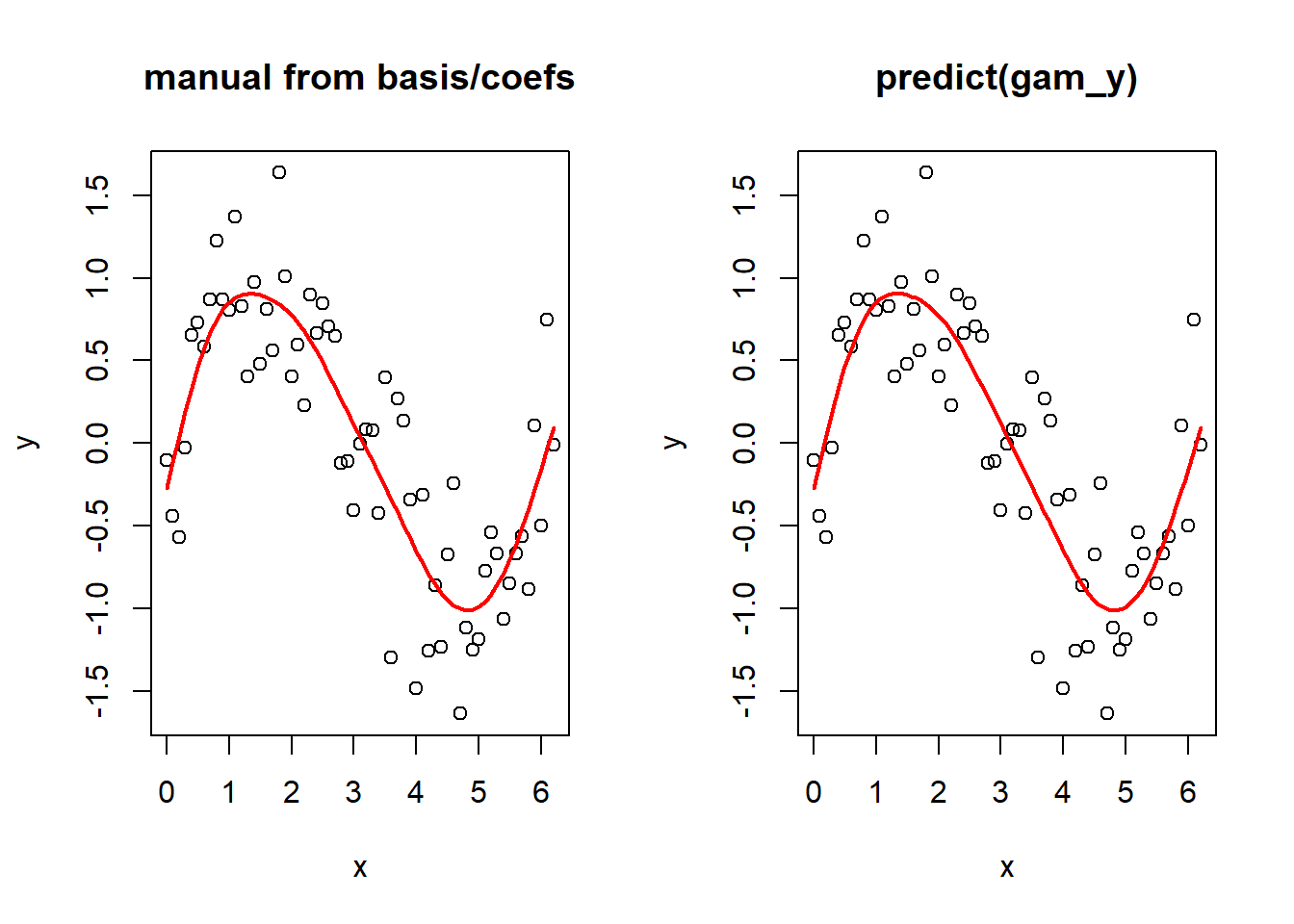
par(mfrow = c(1,2))

plot(y ~ x, main = "manual from basis/coefs")

lines(linear\_pred ~ x, col = "red", lwd = 2)

plot(y ~ x, main = "predict(gam\_y)")

lines(y\_pred ~ x\_new, col = "red", lwd = 2)

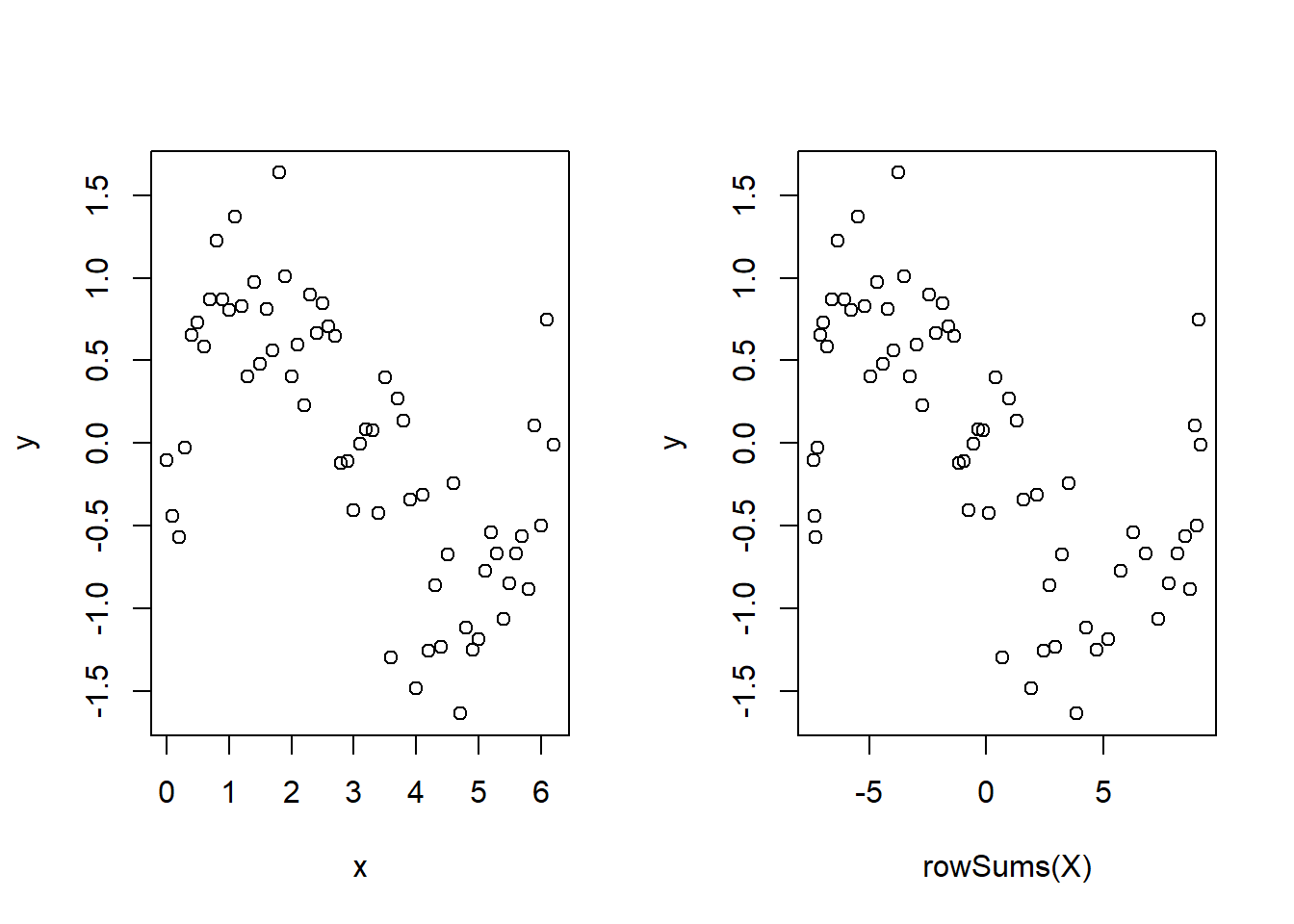


Out of interest, take a look at the following plot, remembering that X is the matrix of basis functions.

par(mfrow = c(1,2))

plot(y ~ x)

plot(y ~ rowSums(X))



Not so scary huh? Of course this is not quite the whole story – see gam.models and smooth.terms to see all of the options for types of smoothers, how the basis functions are constructed (penalisation etc.), types of models we can specify (random effects, linear functionals, interactions, penalisation) plus much more.

**A quick real example**



We’ll now look at a quick real example – we’ll just scratch the surface, and in a future tutorial we will look at it in more detail. We’re going to look at some CO2 data from Manua Loa (it’s used elsewhere in this series). We will fit a couple GAMs to the data to try and pick apart the intra- and inter-annual trends.

First load the data – you can download it [here](http://environmentalcomputing.net/wp-content/uploads/2017/10/manua_loa_co2.csv).

CO2 <- read.csv("mauna\_loa\_co2.csv")

We want to look at inter-annual trend first, so let’s convert the date into a continuous time variable (take a subset for visualisation).

CO2$time <- as.integer(as.Date(CO2$Date, format = "%d/%m/%Y"))

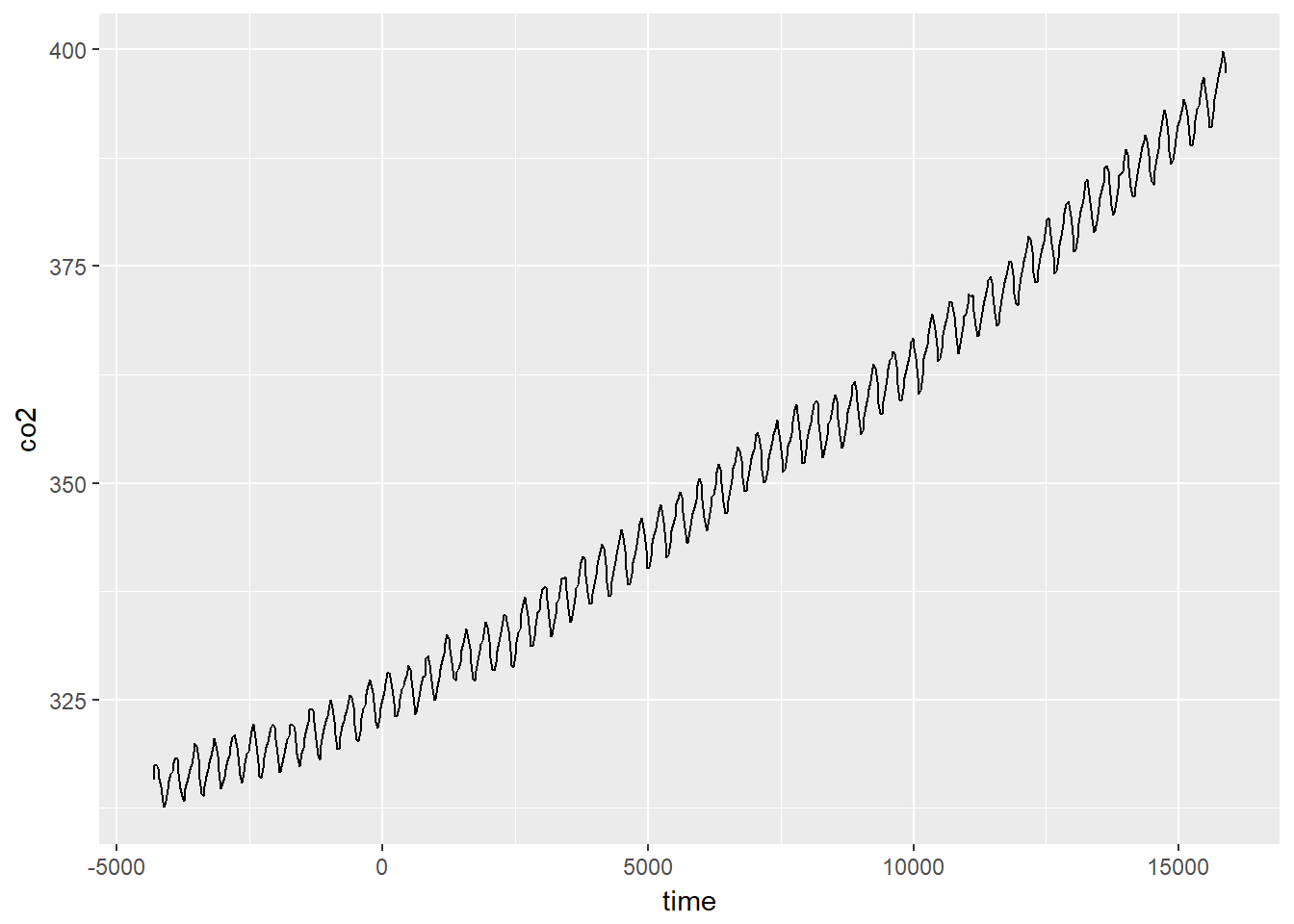
CO2\_dat <- CO2

CO2 <- CO2[CO2$year %in% (2000:2010),]

OK, so let’s plot it and look at a smooth term for time.

y=β0+ftrend(time)+ε,ε∼N(0,σ2)

ggplot(CO2\_dat, aes(time, co2)) + geom\_line()

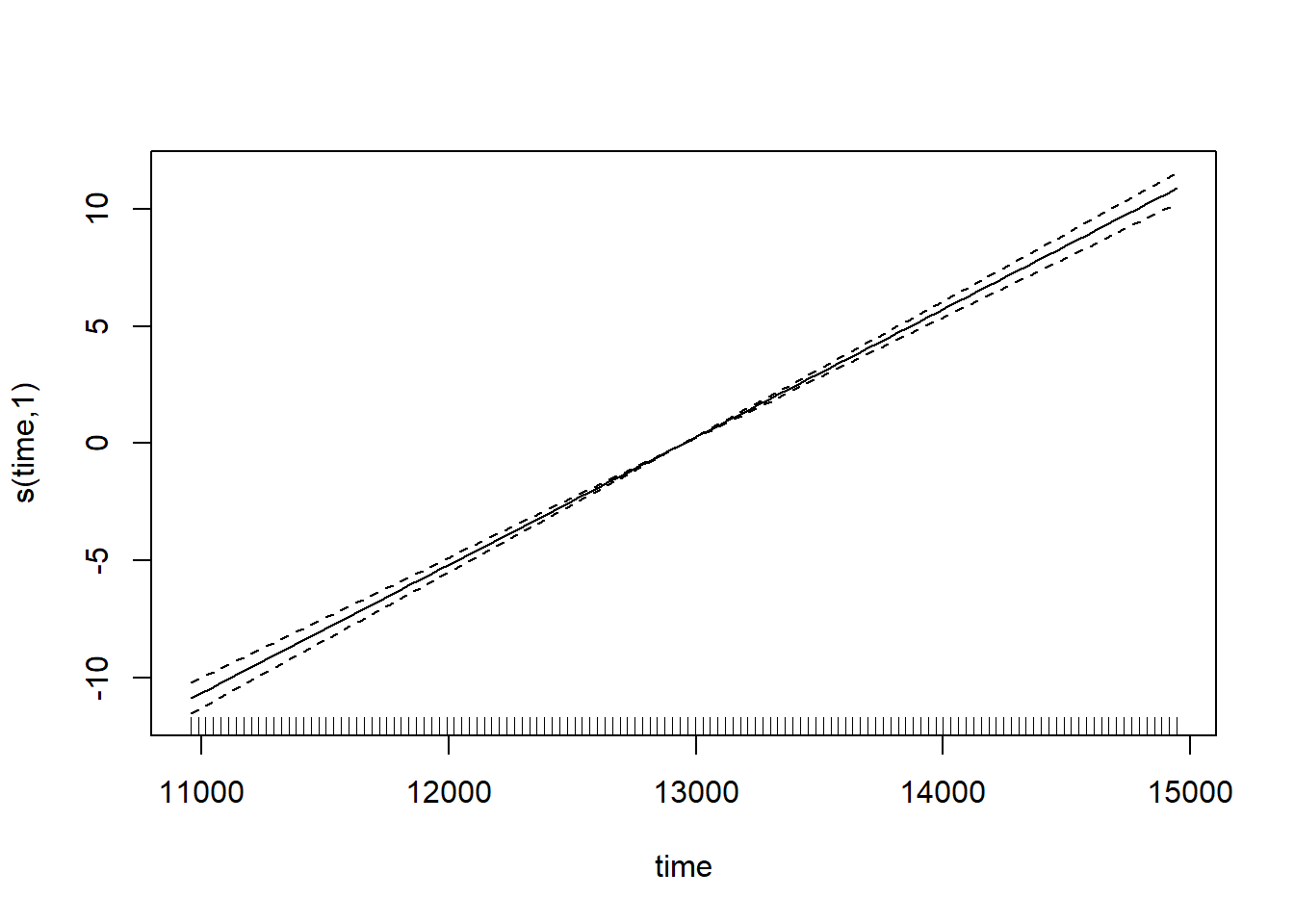


We can fit a GAM for these data using:

CO2\_time <- gam(co2 ~ s(time), data = CO2, method = "REML")

which fits a model with a single smooth term for time. We can look at the predicted values for this:

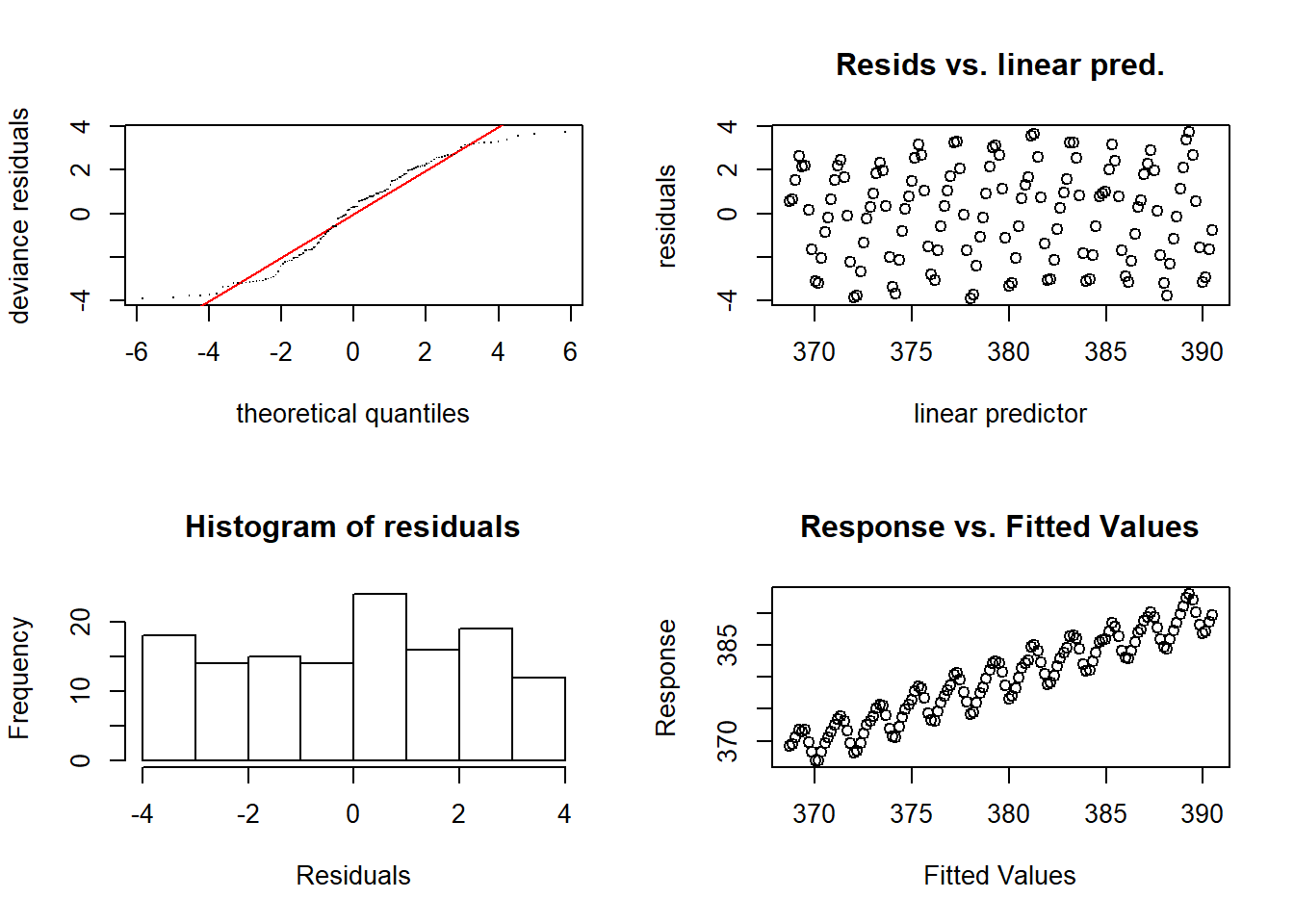
plot(CO2\_time)



Note how the smooth term actually reduces to a ‘normal’ linear term here (with an edf of 1) – that’s the nice thing about penalised regression splines. But if we check the model, then we see something is amuck.

par(mfrow = c(2,2))

gam.check(CO2\_time)



The residual plots have a very odd looking rise-and-fall pattern – clearly there is some dependence structure (and we can probably guess it has something to do with intra-annual fluctuations). Let’s try again, and introduce something called a cyclical smoother.

y=β0+fintrannual(month)+ftrend(time)+ε,ε∼N(0,σ2)

The cyclical smooth term, fintrannual(month), is comprised of basis functions just the same as we have seen already, except that the end points of the spline are constrained to be equal – which makes sense when we’re modelling a variable that is cyclical (across months/years).

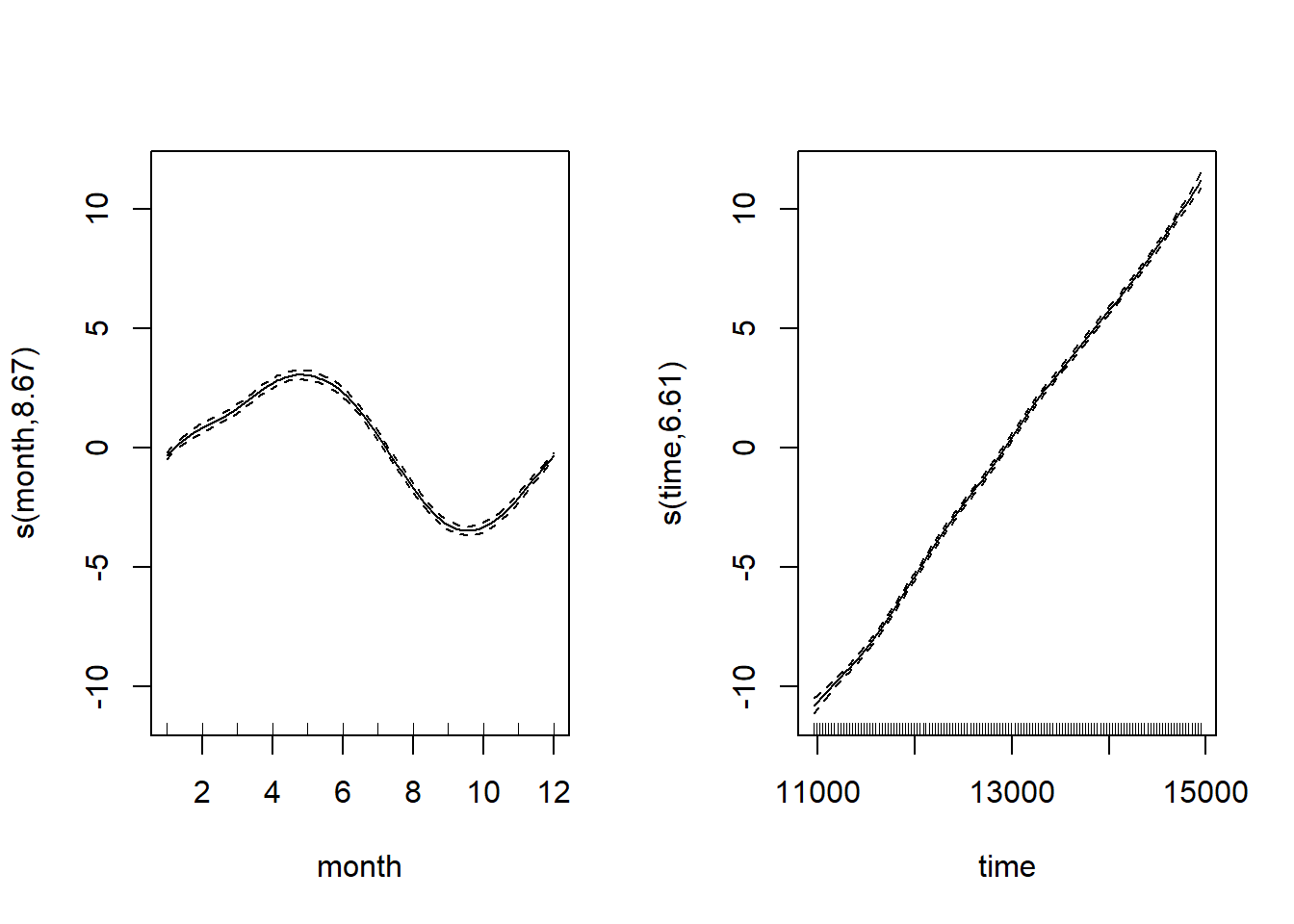
We’ll now see the bs = argument to choose the type of smoother, and the k = argument to choose the number of knots, because cubic regression splines have a set number of knots. We use 12 knots, because there are 12 months.

CO2\_season\_time <- gam(co2 ~ s(month, bs = 'cc', k = 12) + s(time), data = CO2, method = "REML")

Let’s look at the fitted smooth terms:

par(mfrow = c(1,2))

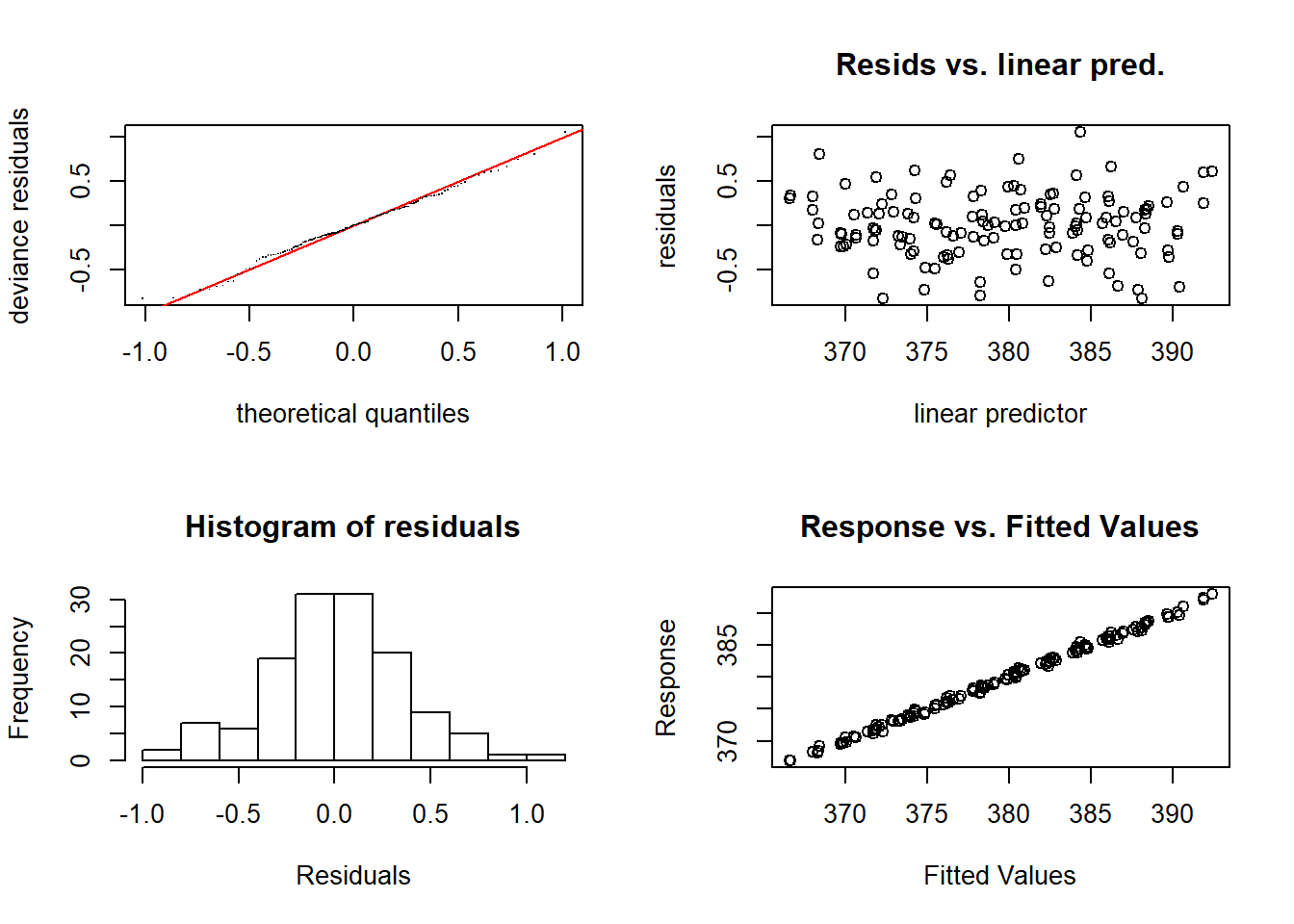
plot(CO2\_season\_time)



Looking at both smooth terms, we can see that the monthly smoother is picking up that monthly rise and fall of CO2 – looking at the relative magnitudes (i.e. monthly fluctuation vs. long-term trend), we can see how important it is to disintangle the components of the time series. Let’s see how the model diagnostics look now:

par(mfrow = c(2,2))

gam.check(CO2\_season\_time)

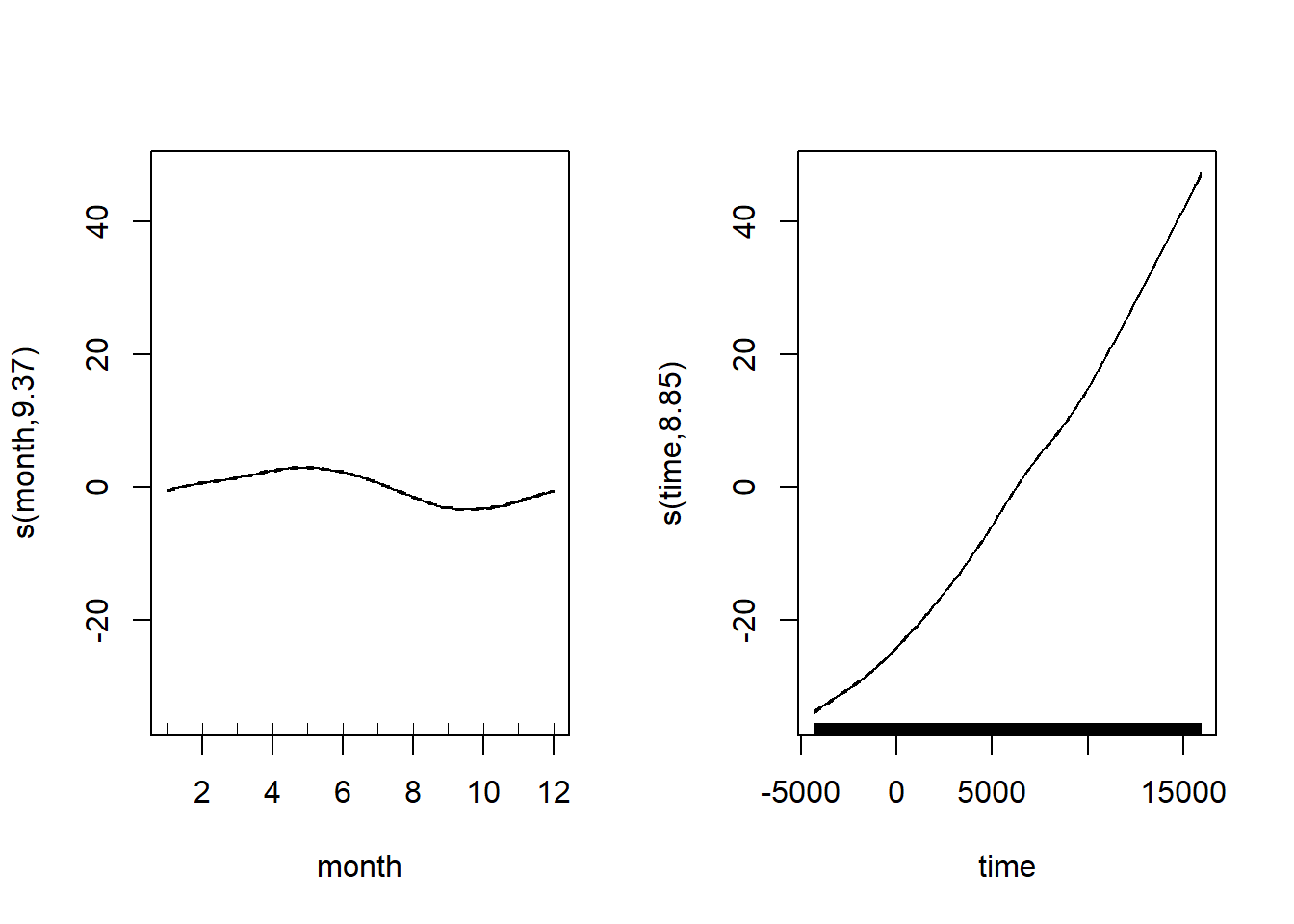


Much better. Let’s look at how the seasonal component stacks up against the full long term trend.

CO2\_season\_time <- gam(co2 ~ s(month, bs = 'cc', k = 12) + s(time), data = CO2\_dat, method = "REML")

par(mfrow = c(1,2))

plot(CO2\_season\_time)



There’s more to the story – pephaps spatial autocorrelations of some kind? gam can make use of the spatial autocorrelation structures available in the nlme package, more on that next time. Hopefully for the meantime GAMs now don’t seem qutie so scary or magical, and you can start to make use of what is really an inrecibly flexible and powerful modelling framework.

**Communicating the results**

You can essentially present model results from a GAM as if it were any other linear model, the main difference being that for the smooth terms, there is no single coefficient you can make inference from (i.e. negative, positive, effect size etc.). So you need to rely on either interpretting the parital effects of the smooth terms visually (e.g. from a call to plot(gam\_model)) or make inference from the predicted values. You can of course include normal linear terms in the model (either continuous or categorical, and in an ANOVA type framework even) and make inference from them like you normally would. Indeed, GAMs are often useful for accounting for a non-linear phenomonon that is not directly of interest, but needs to be acocunted for when making inferece about other variables.

You can plot the partial effects by calling the plot function on a fitted gam model, and you can look at the parametric terms too, possibly using the termplot function too. You can use ggplot for simple models like we did earlier in this tutorial, but for more complex models, it’s good to know how to make the data using predict. We just use the existing time-series here, but you would generate your own data for the newdata= argument.

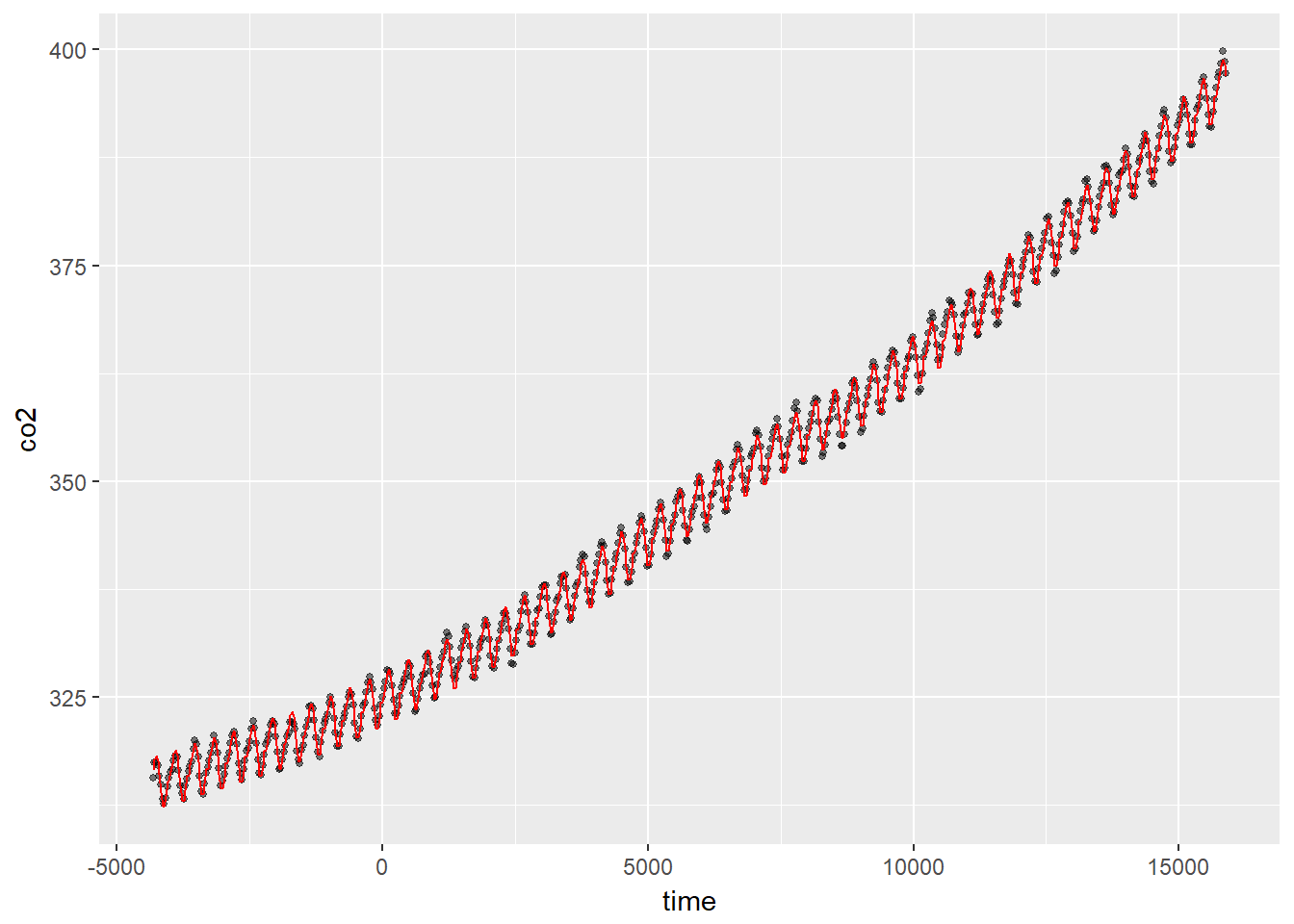
CO2\_pred <- data.frame(time = CO2\_dat$time,

co2 = CO2\_dat$co2,

predicted\_values = predict(CO2\_season\_time, newdata = CO2\_dat))

ggplot(CO2\_pred, aes(x = time)) +

geom\_point(aes(y = co2), size = 1, alpha = 0.5) + geom\_line(aes(y = predicted\_values), colour = "red")



**Further help**

* The R help ?gam is very good, and there is masses of information to read
* Check out ?gamm, ?gamm4 and ?bam
* Use citation("mgcv") for a range of papers with more tehcnical explanations – the book on GAMs with R is particularly good (there’s a 2017 version)
* A great blog with lots of stuff on GAMs: <https://www.fromthebottomoftheheap.net/>

**Author**: Mitchell Lyons  
Last updated:

## [1] "Tue Feb 06 2018"