Chapter 7

DJM

28 February 2017

Quick review of OLS

- OLS stands for "ordinary least-squares".
- Essentially, it means "solve the least-squares problem"

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (x_i^\top \beta - y_i)^2 = (X^\top X)^{-1} X^\top Y$$

- The hat matrix is

$$\hat{Y} = X\hat{\beta} = X(X^\top X)^{-1} X^\top Y = HY$$

- The Gauss-Markov theorem says if:
  1. $Y_i = x_i^\top \beta + \epsilon_i$
  2. $\mathbb{E}[\epsilon_i] = 0$
  3. $\mathbb{V}[\epsilon_i] = \sigma^2 < \infty$
  4. $\operatorname{Cov}[\epsilon_i,\ \epsilon_j] = 0$

Then $\hat{\beta}$ has the smallest variance of all possible unbiased estimators for β.

What is WLS and why use it?

- Weighted least-squares (WLS) is simply

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (x_i^\top \beta - y_i)^2 = (X^\top W X)^{-1} X^\top W Y$$

- If some of those assumptions for G-M are violated, in particular, if $\mathbb{V}[\epsilon_i]$ depends on x_i (notated like $\sigma^2(x_i)$), then we lose the optimality.
- Aside: Gauss-Markov is a commonly used justification for OLS in applied work. The logic goes like this: (1) unbiased is good, (2) G-M says OLS is the best linear model which is unbiased. The problem is that (1) is wrong. Unbiased may be good, but often a little bias is better.
- So what does WLS do?
  1. It **is** optimal, in the sense of G-M, if $\mathbb{V}[\epsilon_i] = \sigma^2(x_i)$.
  2. You've already used it (see next slide).
  3. What if you want to predict Y_i which have other structures like $y_i \in \{0, 1\}$? The new estimators (often called GLS for generalized least squares) are special cases of WLS (logistic regression is one example we are building to).

You already used WLS

- I said you already did this. It is convenient that Kernel regression **is** WLS.
- In particular Kernel regression looks like

$$\hat{c} = \underset{c}{\operatorname{argmin}} \sum_{j=1}^{n} \sum_{i=1}^{n} w_{ij}(c_j - y_i)^2 \quad w_{ij} = \frac{K((x_i - x_j)/h)}{\sum_{i=1}^{n} K((x_i - x_j)/h)}$$

  This is locally constant regression.
- You don't need to understand this formula, but it can be useful, and it provides some justification for WLS based on previous ideas.

What goes wrong with heteroskedasticity?

- So suppose $\mathbb{V}[\epsilon_i] = \sigma^2(x_i)$. That is our "homoskedasticity" assumption is violated. Should we care?
- What if we just use OLS (that is `lm`) anyway?
- Some things don't change.
  1. We still have that $\mathbb{E}\left[\hat{\beta}\right] = \beta$. That is OLS **is** still unbiased.
  2. We still have that OLS minimizes the sum of squared residuals: among all lines, OLS makes $\sum_{i=1}^{n}(x_i^\top \hat{\beta} - y_i)^2$ as small as possible.
- Some things **do** change.
  1. OLS no longer has the best variance of all unbiased estimators (WLS does).
  2. The standard errors that `R` produces are wrong. They make it seem "more certain" than is correct (could use the bootstrap to fix it though).
  3. So are the F-tests and p-values (again, the bootstrap).

Log squared residuals

- So WLS is fairly general. But for now, let's focus on how to use it for heteroskedasticity.
- Suppose you **know** the following:
  1. $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$.
  2. You know $\beta_0 = 3$ and $\beta_1 = 2$.
  3. $\mathbb{E}[\epsilon_i] = 0$
  4. $\mathbb{V}[\epsilon_i] = \sigma^2(x_i)$ ($\sigma^2(\cdot)$ is a function).
- You don't know $\sigma^2(\cdot)$
- How would you estimate $\sigma^2(x)$?
- You can use nonparametric regression of course!
  - Just look at $e_i = y_i - 3 + 2x_i$.
  - We already know that e_i has mean zero, so no use estimating it's mean
  - Therefore $\mathbb{E}\left[e_i^2\right] = \sigma^2(x_i)$.
  - Now this is easy: rewrite the model as $e_i^2 = \sigma^2(x_i) + \eta_i$
  - It's just nonparametric regression (since you know e_i^2 and x_i).

So why not?

- There's one problem with the above line of reasoning: $e_i^2 > 0$ so there are some constraints on η_i.
- Imagine if, say, at $x = 1$, $\sigma^2(1) = .001$, then it's pretty likely that η is large and positive there, so there's heteroskedasticity in our model for heteroskedasticity...
- If the original ϵ_i were $N(0, \sigma^2(x_i))$, then the new η_i are distributed as χ^2_1, so these are right skewed, everywhere.
- A remedy is to look at $\log e_i^2 = \log \sigma^2(x_i) + \tau_i$.
- You could try both and look at qq-plots of the residuals. I tend to prefer the second set-up. It's just more satisfying.
- This is just a transformation: just like when you looked at qq-plots and decided to model $\log Y$ rather than Y.

What's the Oracle?

- So back to WLS.
- I had that example where I wanted to estimate the variance function
- In that case **I knew the mean:** $3 + 2x$
- I knew because the Oracle told me. The Oracle is a wise woman who lives at Delphi according to the ancient Greeks, and speaks the thoughts of Apollo.
- In other words, she tells me things no one could possibly know, like the mean function.
- In statistics, we talk about Oracles a lot, usually as a way of comparing a procedure we cook up for estimating something to the answers the Oracle would have told us (the best possible, but unobtainable estimator).

A big example

- This is a (slightly modified) portion of a real job interview.
- It is a very simple application of heteroskedasticity.
- Heteroskedasticity appears frequently with financial data, so those companies like to see if you can handle it.

The set up

- The dataset `jobInt` contains data from a simple linear model with heteroskedastic noise.
- In other words, for $i = 1, \ldots, 250$,

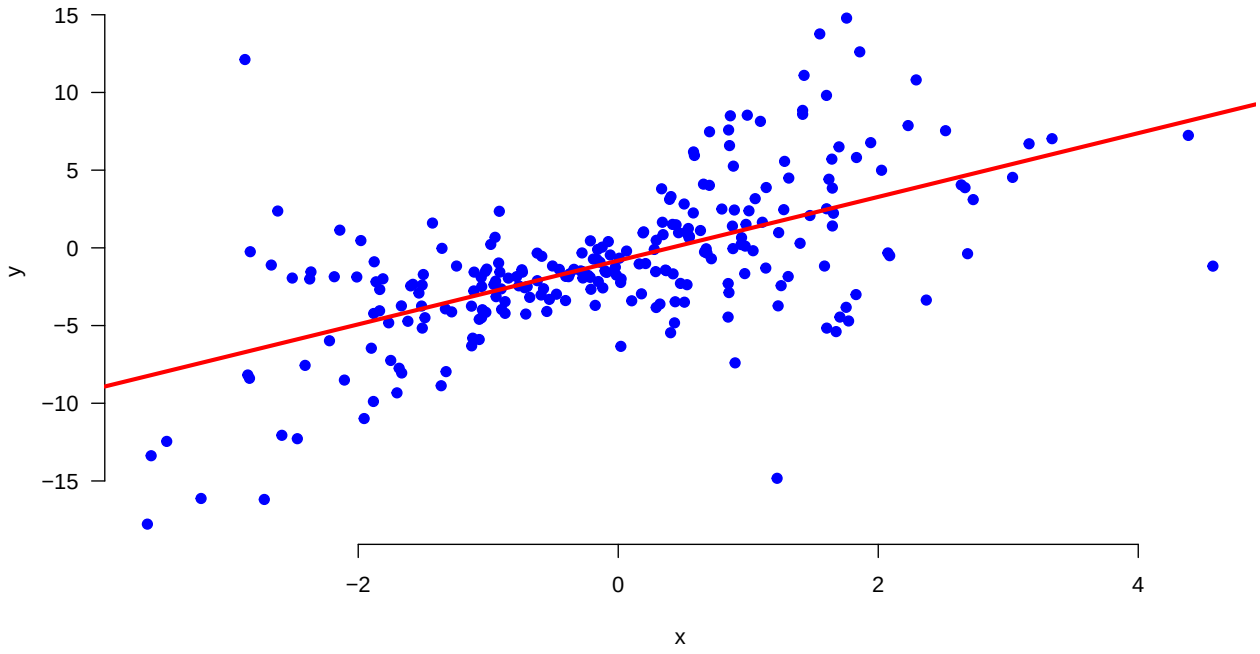
$$y_i = \beta_0 + \beta_1 x_i + \sigma(x_i)\epsilon_i \qquad \epsilon_i \sim \mathrm{N}(0, 1).$$

- You know nothing about (the function) $\sigma(\cdot)$.
- Your goal is to estimate $(\beta_0,\ \beta_1)$ as well as possible, and provide a CI.

How do I do this?

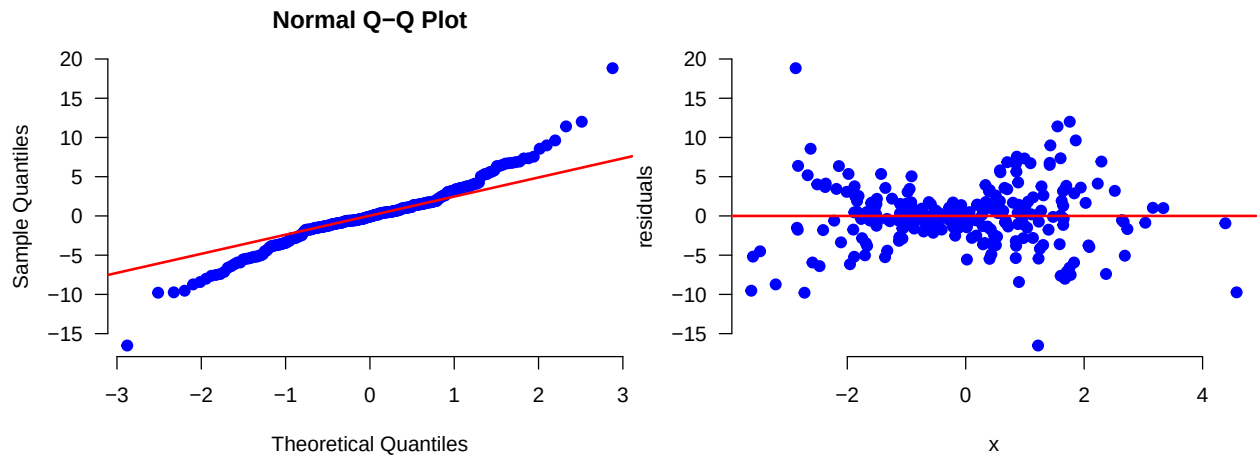
- First things first, EDA.

```
plot(jobInt, pch=19, col=4, bty='n', las=1)
basicMod = lm(y~x)
abline(basicMod, col=2, lwd=3)
```



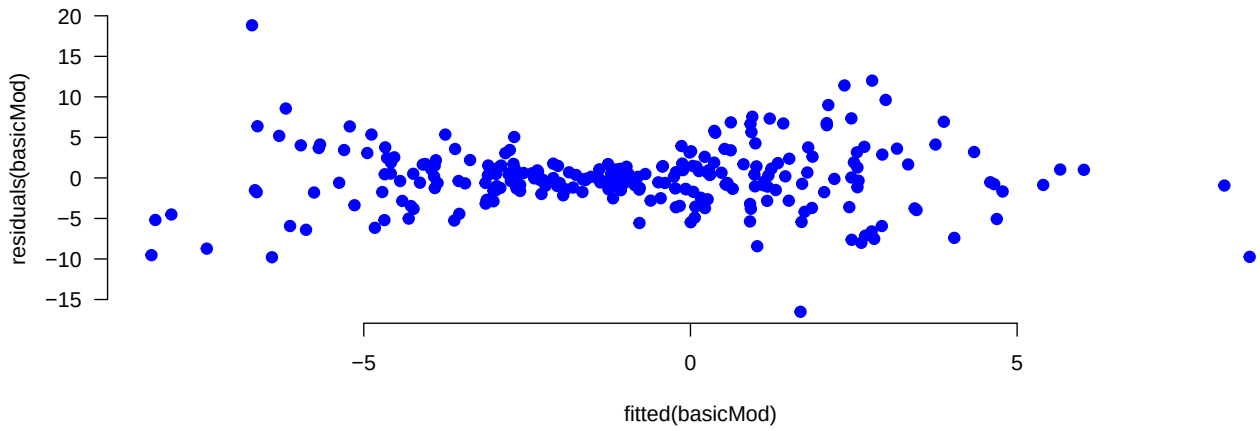
qq-plot and residuals against x

```
par(mfrow=c(1,2), las=1, bty='n', mar=c(5,4,3,0), lwd=2, pch=19)
qqnorm(residuals(basicMod), col=4)
qqline(residuals(basicMod), col=2)
plot(jobInt$x, residuals(basicMod), col=4, xlab='x', ylab='residuals')
abline(h=0, col=2)
```



residuals vs. fitted values

```
par(las=1, bty='n', mar=c(5,4,3,0), lwd=2, pch=19)
plot(fitted(basicMod), residuals(basicMod), col=4)
```



So now what?

- We **know** that $\mathbb{E}[Y \mid X = x] = \beta_0 + \beta_1 x$.
- We **know** that $\mathbb{E}[\hat{e} \mid X = x] = 0$.
- We **know** that $\mathbb{E}\left[\hat{e}^2 \mid X = x\right] = \sigma^2(x)$.
- So we want to try to estimate $\sigma^2(x)$ and β_0 and β_1 all at the same time.

Oracle information

- If we knew β_0 and β_1, then we could use `npreg` to estimate $\sigma^2(x)$.
- If we **knew** $\sigma^2(x)$, then we could use WLS to estimate β_0 and β_1 (and all the SEs would be right!)
- But we don't know either.

Procedure

1. Use `lm` to estimate β_0 and β_1.
2. Now pretend that you "know" them, calculate $\log(\hat{e}^2)$ and use `npreg` to estimate $\log \sigma^2(x)$.
3. Now pretend that you "know" $\sigma^2(x)$ (take exp of your estimate from 2.) and use WLS (with `lm(y~x, weights=1/sig2)`)
4. You could stop here. But since you now have "better" estimates of β_1 and β_0, it's better to iterate 2 and 3 until some convergence.
5. Ok. Something converged, so you return the last estimates of β_0 and β_1. But the SEs are not right (because you "know" $\sigma^2(x)$ but you don't **know** it).
6. To get SEs, use the bootstrap:
   a. Non-parametric: repeat 1-5 B times on resampled data.
   b. Model-based: this is actually pretty hard here, better not to do it.

Some code

- This code takes in data and does steps 1-5. It is **not** optimized for speed, but for readability, so run with care.

```
heteroWLS <- function(dataFrame, tol = 1e-4, maxit = 100, track=FALSE){
  # inputs: a data object, optional: tolerance, max.iterations, and progress tracker (prints)
  # outputs: estimated betas and weights
  require(np)
  ols = lm(y~x, data=dataFrame)
  b = coefficients(ols)
  conv = FALSE
  for(iter in 1:maxit){ # don't let this run forever
    if(conv) break # if the b's stop moving, get out of the loop
    logSqResids = log(residuals(ols)^2)
    winv = exp(predict(npreg(logSqResids~x, data=dataFrame, tol=1e-2, ftol=1e-2)))
    winv[winv < tol] = tol # zero inverse weights are bad, make them small
    ols = lm(y~x, weights = 1/winv, data=dataFrame) #weights are 1 / estim.variance
    newb = coefficients(ols)
    conv.crit = sum((b-newb)^2) # calculate how much b moved
    if(track) cat('\n', iter, '/', maxit, ' conv.crit = ', conv.crit) # print progress
    conv = (conv.crit < tol) # check if the b's changed much
    b = newb # update the coefficient estimates
  }
  return(list(betas=b, weights = winv, log2resids = log(residuals(ols)^2)))
}
```

Do it! (takes a little while...)

```
resampWLS <- function(dataFrame,...){ # ... means options passed on
  rowSamp = sample(1:nrow(dataFrame), size=nrow(dataFrame), replace=TRUE)
  return(heteroWLS(dataFrame[rowSamp,],...)$betas) # passed things on if desired
}
B = 100 #
alp = .05
origBetas = heteroWLS(jobInt)
ptm = proc.time()
bootBetas = replicate(B, resampWLS(jobInt, maxit=20))
ptm = proc.time()-ptm
qq = apply(bootBetas, 1, quantile, probs=c(1-alp/2, alp/2))
CI = cbind(origBetas$betas, 2*origBetas$betas - t(qq))
colnames(CI) = c('coef', rev(colnames(CI)[2:3]))
```

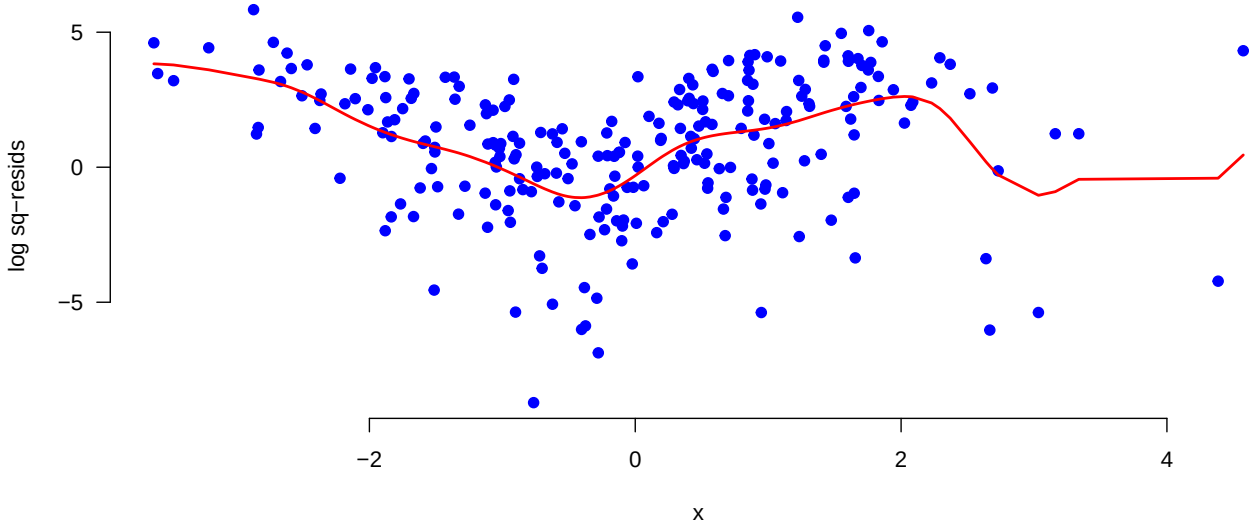
```
CI
```

```
##                  coef      2.5%      97.5%
## (Intercept) -1.035868 -1.645894 -0.7582354
## x            1.858586  1.218625  2.3484705
```

```
ptm
```

```
##    user  system elapsed
##  59.490   0.561  61.450
```

Some plots



Some caveats

- If we **care** about estimating $\sigma(\cdot)$, then what we did is ok.
- But we don't.
- We only care about estimating β_0 and β_1.
- So better to use CV with leaving out (y_i, x_i), instead of using CV to estimate $\sigma(\cdot)$ (which is what `npreg` is doing; it know's nothing about y_i)
- This takes a bit more work to code up (Try it!)

Local linear vs. Kernels

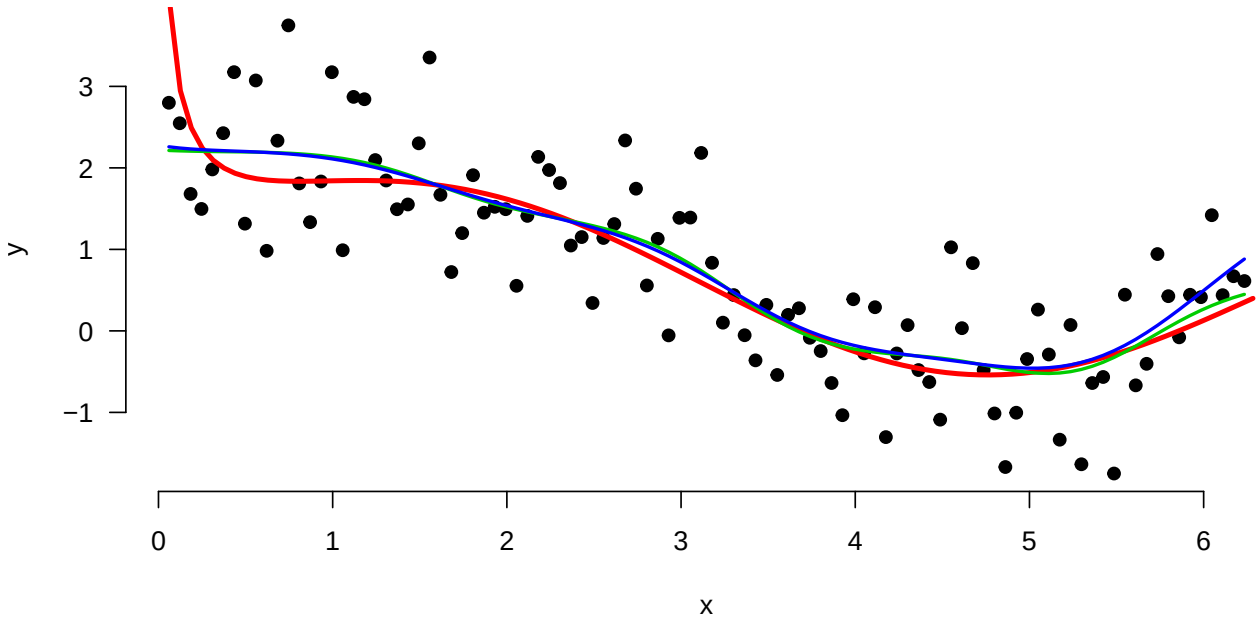
- People often wonder whether to use local linear regression or Kernels.
- Like with many things, there isn't really a cut-and-dried answer.
- Some practitioners prefer local linear regression.
- It's main benefit is to correct "boundary bias".
- Otherwise, not much different.

Repeat of Ch. 4

- We can estimate this easily with both a kernel and local linear regression

```
set.seed(1234)
trueFunction <- function(x) sin(x) + 1/sqrt(x)
x = runif(100, 0, 2*pi)
x = seq(min(x), max(x), length.out=100)
y = trueFunction(x) + rnorm(100, 0, .75)
par(mar=c(5,4,0,0))
```

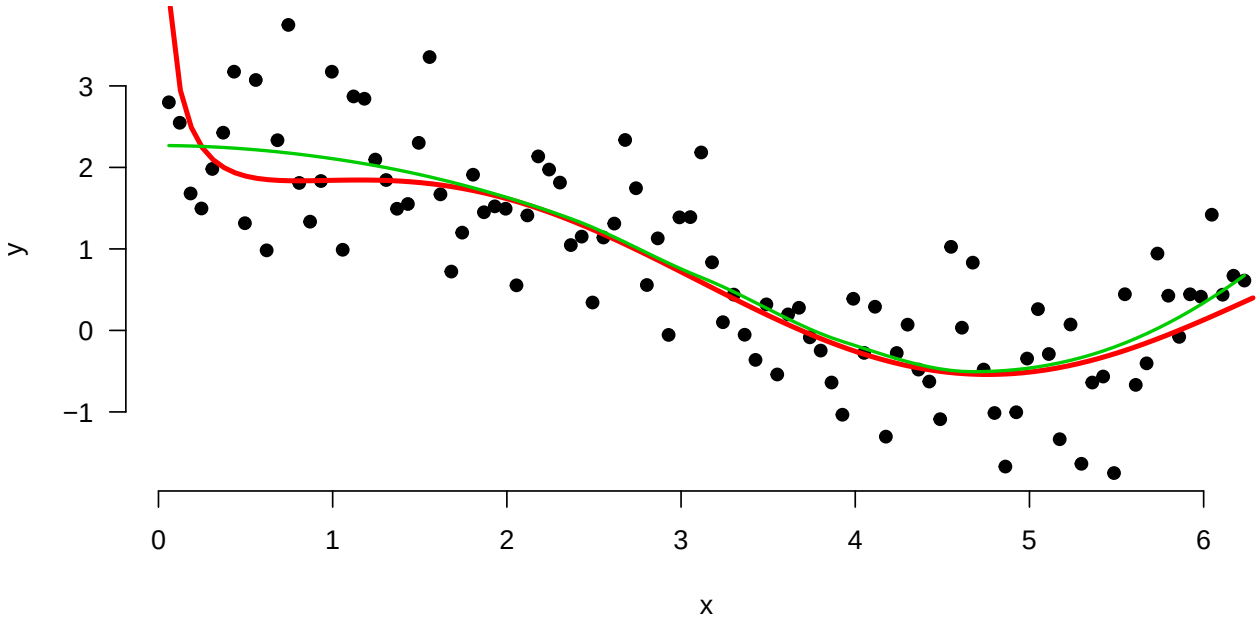
```
plot(x, y, pch=19, bty='n', las=1, cex.lab=1, cex.axis=1)
curve(trueFunction(x), 0, 2*pi, col=2, lwd=3, add=TRUE)
library(np)
kern = npreg(y~x)
loclin = npreg(y~x, regtype='ll')
lines(x, fitted(kern), col=3, lwd=2)
lines(x, fitted(loclin), col=4, lwd=2)
```



What is Loess?

- So kernel regressions are local constants, we then saw local linear regression which is quite similar.
- Why stop there? The next term is squared things, then cubics, then...
- Loess uses local polynomials (of some order) in a particular way (combining k-nearest neighbor regression with subsampling).
- It is actually quite cool, but it's complexity makes it hard to deal with.
- Theoretically, one can show that Kernels are optimal, so it's not really worth worrying too much about, but it can work well, and it doesn't require installing a package.
- Here it is on my previous example

```
ls = loess(y~x)
lines(x, predict(ls), col=3, lwd=2)
```



Chapter 9 intro

- Here we introduce the concept of GAMs (**G** eneralized **A** dditive **M** odels)
- The basic idea is to imagine that the response is the sum of some functions of the predictors:

$$\mathbb{E}\left[Y_i \mid X_i = x_i\right] = \alpha + f_1(x_{i1}) + \cdots + f_p(x_{ip}).$$

- Note that OLS **is** a GAM (take $f_j(x_{ij}) = \beta_j x_{ij}$):

$$\mathbb{E}\left[Y_i \mid X_i = x_i\right] = \alpha + \beta_1 x_{i1}) + \cdots + \beta_p x_{ip}.$$

- The algorithm for fitting these things is called "backfitting":
  1. Center Y and X.
  2. Hold f_k for all $k \neq j$ fixed, and regress f_j on the partial residuals using your favorite smoother.
  3. Repeat for $1 \leq j \leq p$.
  4. Repeat steps 2 and 3 until the estimated functions "stop moving" (iterate)
  5. Return the results.

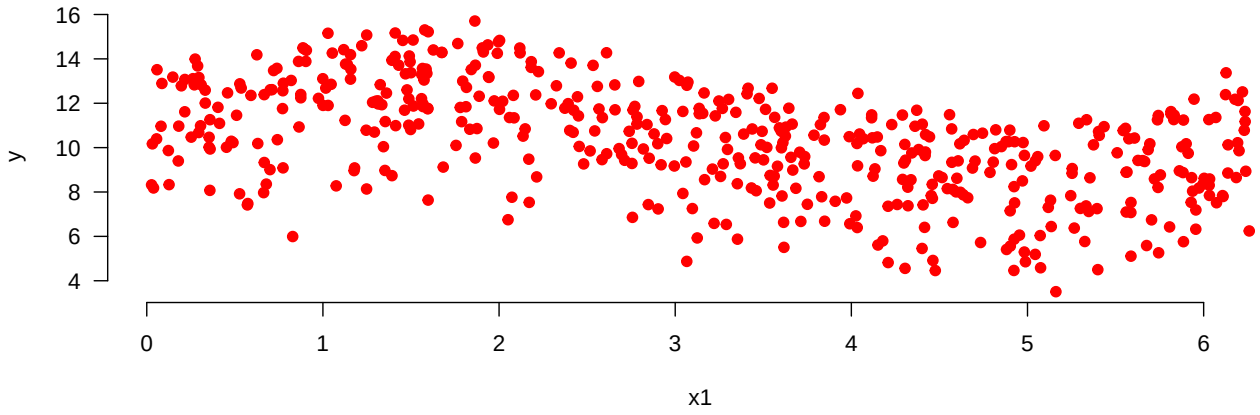
Results

- We're not going to try to code this ourselves (though it's not hard and good practice).
- There are two `R` packages that do this for us. I find `mgcv` easier.
- Let's look at a small example.

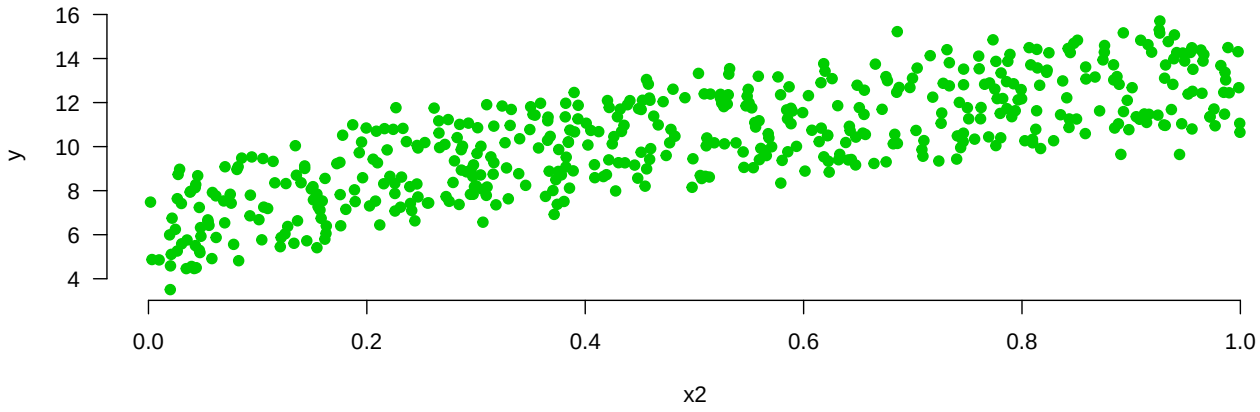
```
if (!is.element("mgcv", installed.packages()[,1]) ) install.packages("mgcv") # installs if it isn't alr
library(mgcv)
set.seed(03-07-2016)
n = 500
x1 = runif(n, 0, 2*pi)
x2 = runif(n)
y = 5 + 2*sin(x1) + 8*sqrt(x2)+rnorm(n,sd=.5)
```

More plots

```
plot(x1, y, col=2, pch=19, bty='n', las=1)
```




```
plot(x2, y, col=3, pch=19, bty='n', las=1)
```



Small example

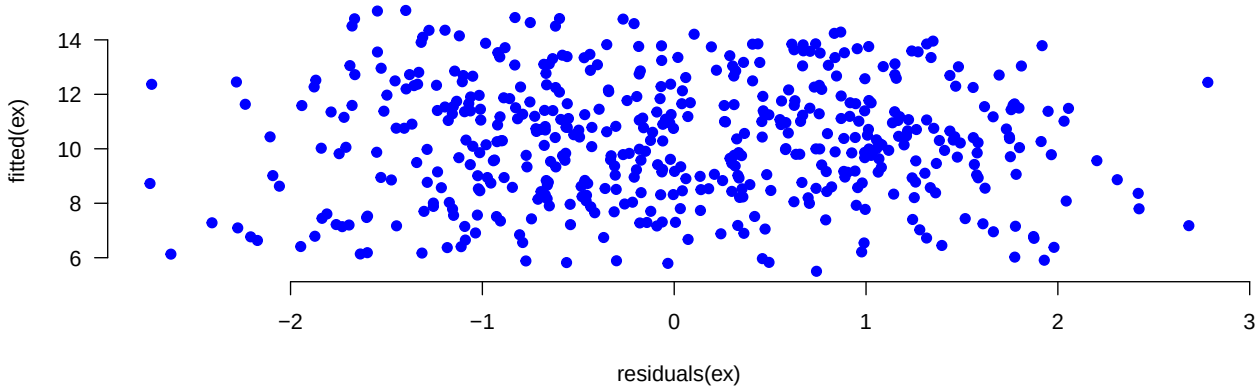
This just fits the linear model.

```
df = data.frame(y, x1, x2)
ex = gam(y~x1+x2, data=df)
summary(ex)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ x1 + x2
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.10033    0.13445   67.69   <2e-16 ***
## x1          -0.59958    0.02713  -22.10   <2e-16 ***
```

```
## x2           6.26541    0.16992   36.87   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =    0.8   Deviance explained = 80.1%
## GCV = 1.1984  Scale est. = 1.1912    n = 500
```

```
plot(residuals(ex),fitted(ex), bty='n', las=1, pch=19, col=4)
```

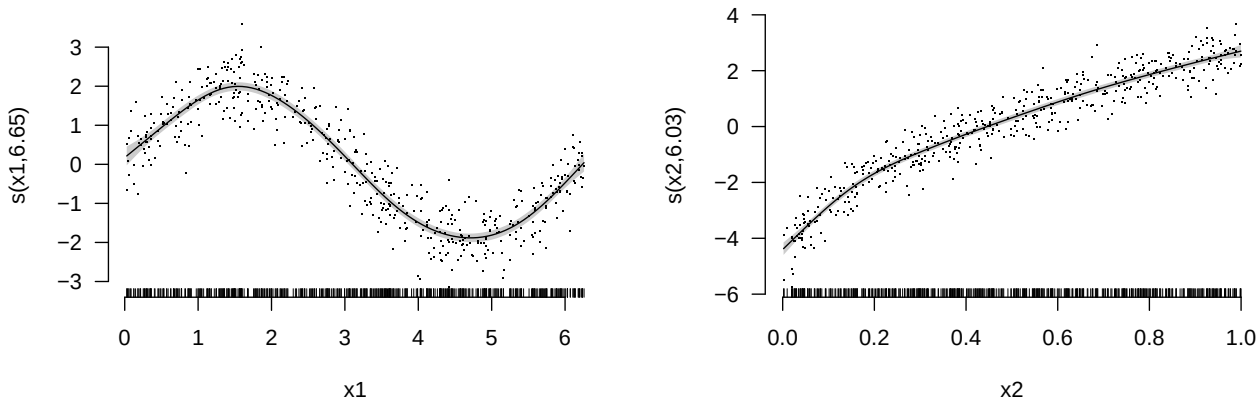


Smoothing

```
ex.smooth = gam(y~s(x1)+s(x2), data=df) # uses splines with GCV on each xi
coefficients(ex.smooth) # still produces something
```

```
##  (Intercept)      s(x1).1      s(x1).2      s(x1).3      s(x1).4
## 10.300188260 -4.390840091 -0.083453655 -0.254475813  0.128249427
##      s(x1).5      s(x1).6      s(x1).7      s(x1).8      s(x1).9
## -0.092309200  0.039765792  0.006702675 -0.413305728  3.196591297
##      s(x2).1      s(x2).2      s(x2).3      s(x2).4      s(x2).5
## -0.523050371  1.695813628 -0.053205945 -0.940567901  0.235820952
##      s(x2).6      s(x2).7      s(x2).8      s(x2).9
## -0.803738757  0.155991337  3.051497054  2.873218636
```

```
plot(ex.smooth, pages = 1, scale=0, shade=TRUE, resid=TRUE, se=2, bty='n', las=1)
```



Residuals vs. fitted

```
plot(residuals(ex.smooth), fitted(ex.smooth), bty='n', las=1, pch=19, col=4)
```