Chapter 10

DJM

10 March 2020

Quick review of OLS

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

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

- The hat matrix is

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

- 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. $\mathrm{Cov}[\epsilon_i,\ \epsilon_j] = 0$

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

What is WLS and why use it?

- Weighted least-squares (WLS) is simply

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

- 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 $\mathbb{V}[\epsilon_i] = \sigma_i^2$.
  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 algorithms for the new estimators (often called GLS for generalized least squares) use 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

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

  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 $\mathbb{E}\left[\widehat{\beta}\right] = \beta$. That is OLS **is** still unbiased.
  2. We still have that OLS minimizes the sum of squared residuals: among all lines, OLS makes $\sum_{i=1}^{n}(x_i^\top \widehat{\beta} - y_i)^2$ 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

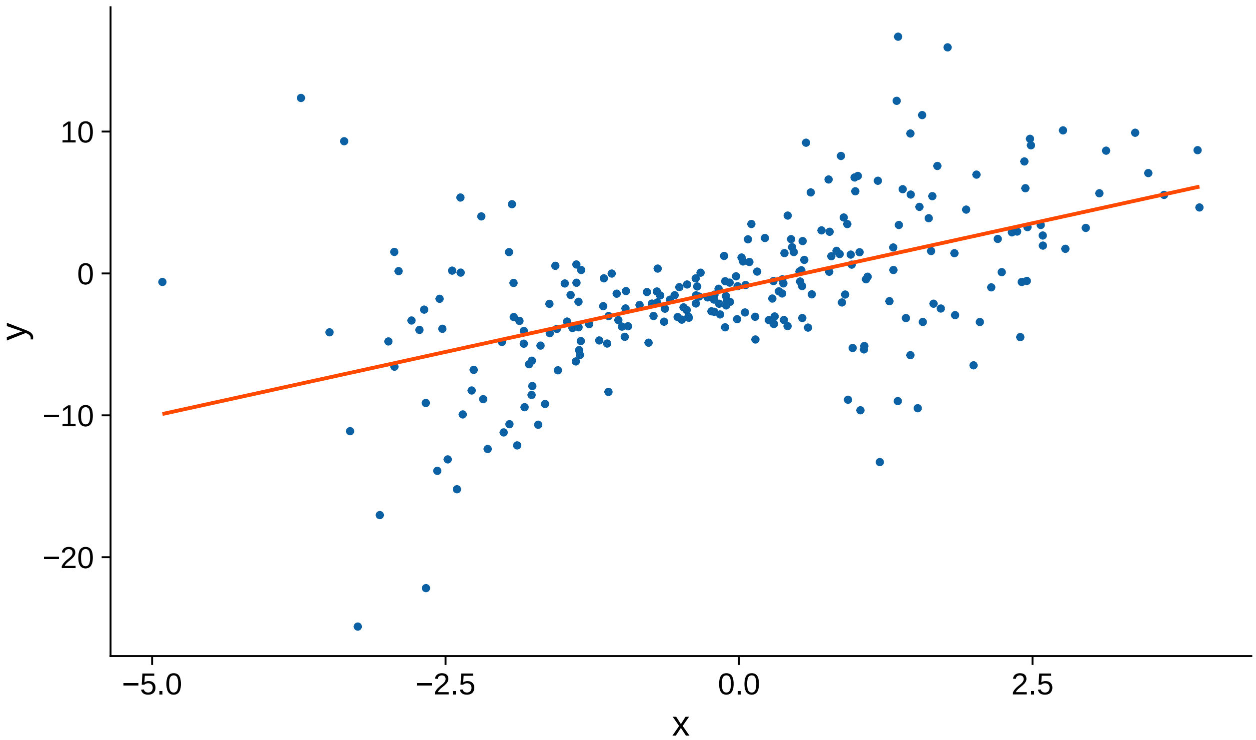
- The dataset `jobInt` contains data from a simple linear model with heteroskedastic noise.
- In other words, for $i = 1, \ldots, 250$,

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

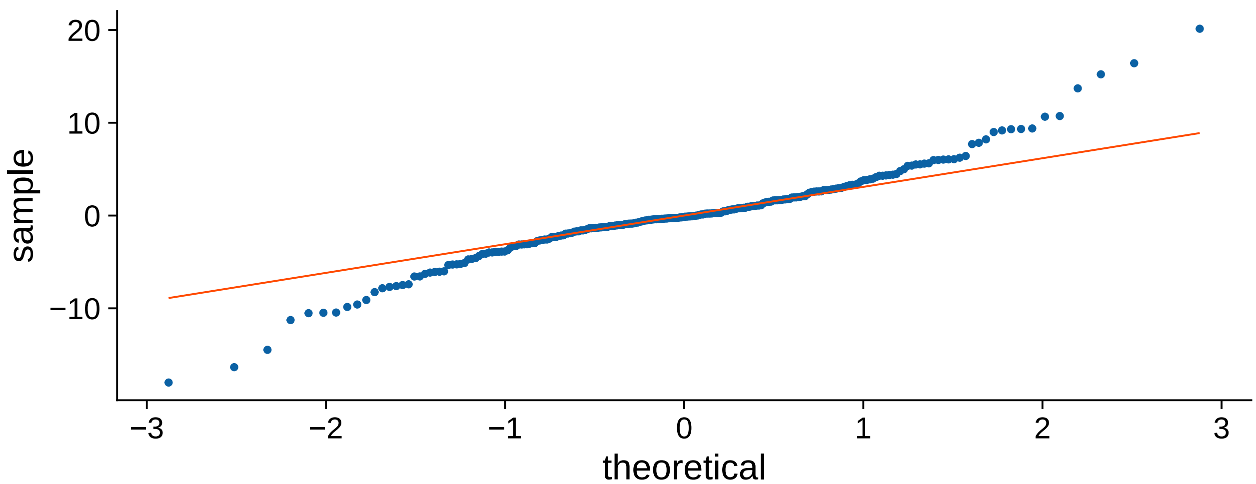
- You know nothing about (the function) $\sigma(\cdot)$.
- Your goal is to estimate $(\beta_0,\ \beta_1)$ as well as possible, and provide a CI.

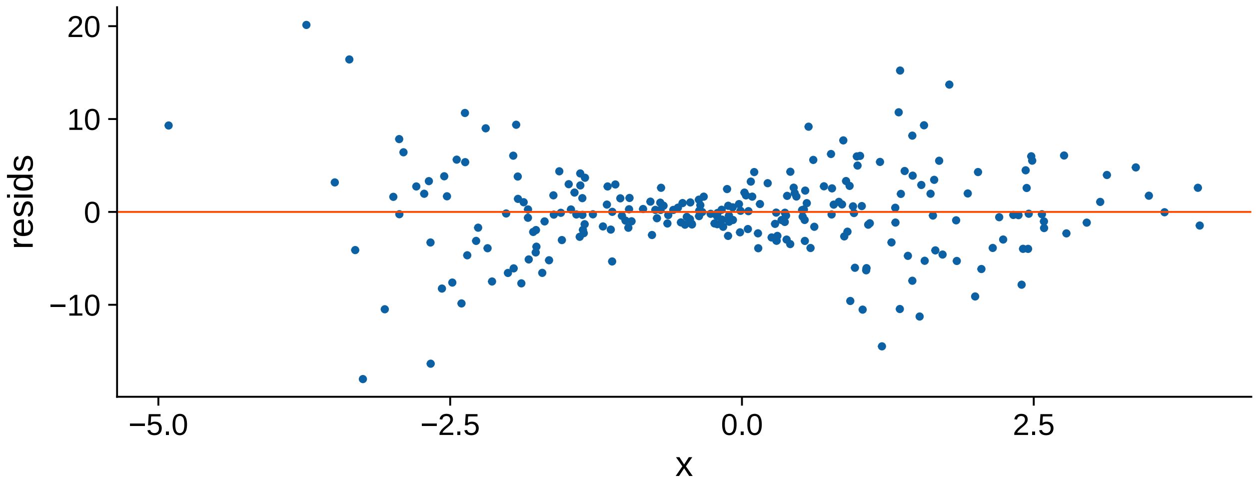
How do I do this?

- First things first, EDA.

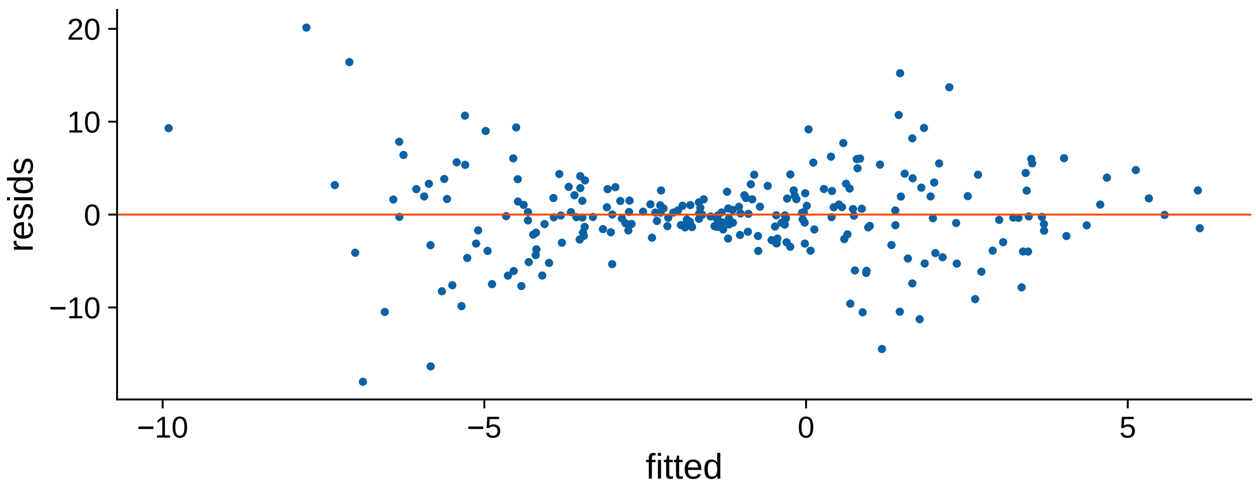


qq-plot and residuals against x





residuals vs. fitted values



So now what?

- We **know** that $\mathbb{E}[Y \mid X = x] = \beta_0 + \beta_1 x$.
- We **know** that $\mathbb{E}[\widehat{e} \mid X = x] = 0$.
- We **know** that $\mathbb{E}\left[\widehat{e}^2 \mid X = x\right] = \sigma^2(x)$.
- 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 $\log(\widehat{e}^2)$ 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.

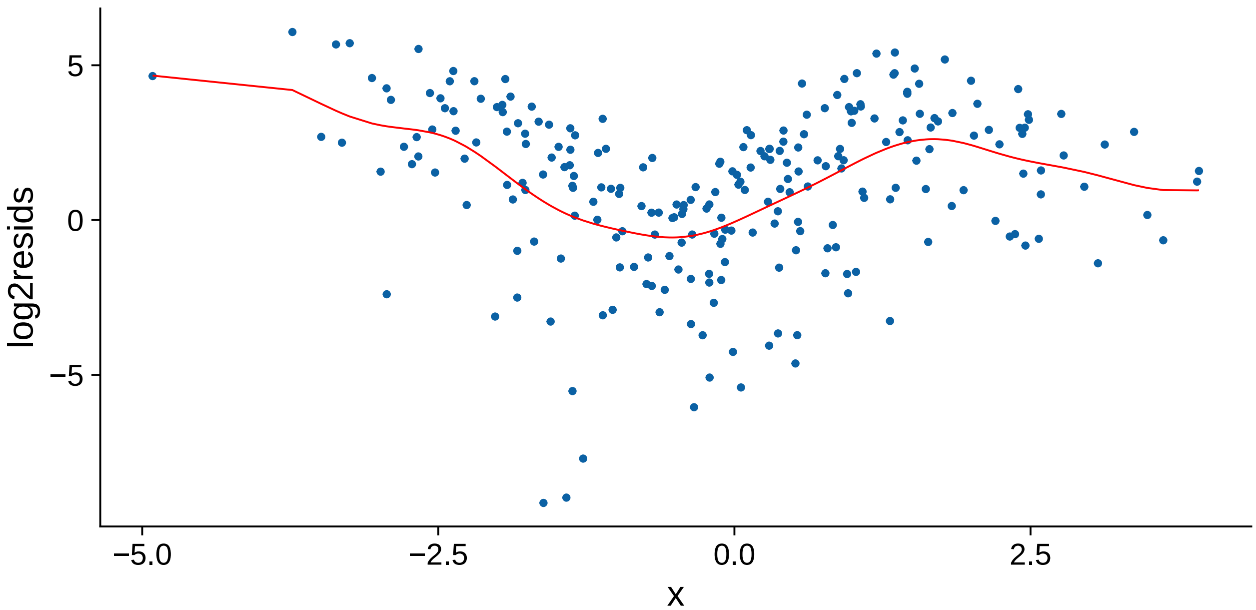
```
heteroWLS <- function(dataFrame, tol = 1e-4, maxit = 100, track=FALSE){
  # inputs: a data object, optional: tolerance, max.iterations, and progress tracker (prints)
  # outputs: estimated betas and weights
  require(np)
  ols = lm(y~x, data=dataFrame)
  b = coefficients(ols)
  conv = FALSE
  for(iter 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,],...)$betas) # passed things on if desired
}
B = 100 #
alp = .05
origBetas = heteroWLS(jobInt)
system.time(bootBetas <- replicate(B, resampWLS(jobInt, maxit=20)))
qq = apply(bootBetas, 1, quantile, probs=c(1-alp/2, alp/2))
CI = cbind(origBetas$betas, 2*origBetas$betas - t(qq))
colnames(CI) = c('coef', rev(colnames(CI)[2:3]))
```

```
##                   coef      2.5%      97.5%
## (Intercept) -0.9938074 -1.406956 -0.4306921
## x            2.0023640  1.607402  2.6352379
```

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 knows nothing about y_i)
- This takes a bit more work to code up (Try it!)

Local linear vs. Kernels

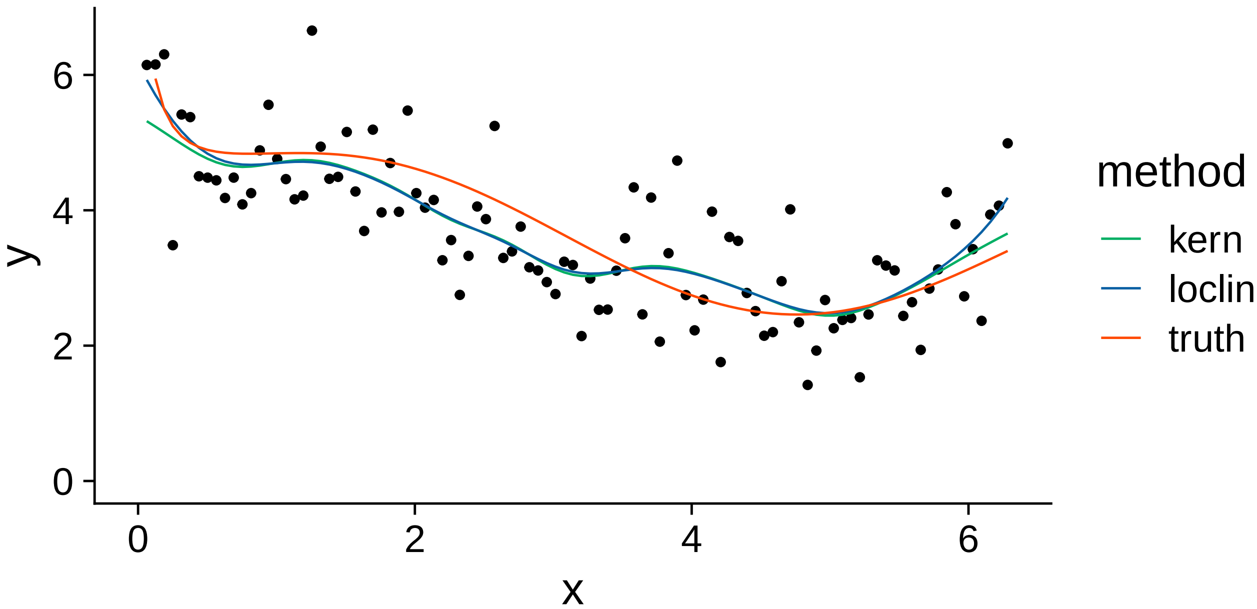
- People often wonder whether to use local linear regression or Kernels.
- Like with many things, there isn't really a cut-and-dried answer.
- Some practitioners prefer local linear regression.
- It's main benefit is to correct "boundary bias".
- Otherwise, not much different.

Repeat of Ch. 4

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

```
kern = npreg(y~x, data = df)
loclin = npreg(y~x, data=df, regtype='ll')
df$kern = fitted(kern)
df$loclin = fitted(loclin)
df$truth = trueFunction(x)
dflines = pivot_longer(select(df,-y),-x, names_to = 'method')
ggplot(df, aes(x, y)) + geom_point() + xlim(0,2*pi) + ylim(0,max(df$y)) +
```

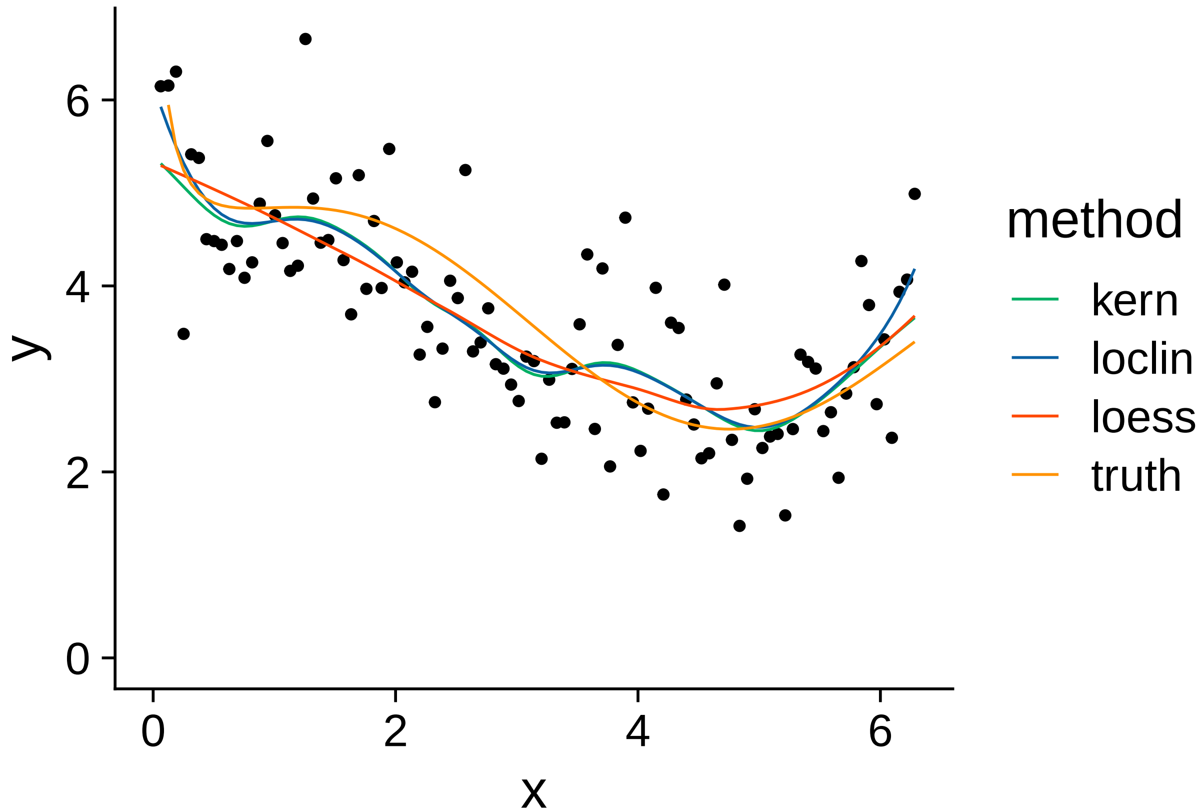
```
  geom_line(data=dflines, aes(x=x,y=value,color=method)) +
  scale_color_manual(values=c(green,blue,red))
```



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, data=df)
df$loess = fitted(ls)
dflines = pivot_longer(select(df,-y),-x,names_to='method')
ggplot(df, aes(x, y)) + geom_point() + xlim(0,2*pi) + ylim(0,max(df$y)) +
  geom_line(data=dflines, aes(x=x,y=value,color=method)) +
  scale_color_manual(values=c(green,blue,red,orange))
```



6
4
2
0
y
0
2
4
6
x
method
kern
loclin
loess
truth