ARE 212 Section 10:
Non-Standard Standard Errors II

Fiona Burlig

March 31, 2016

This week, we'll continue our discussion of non-standard standard errors. I've been looking all over for a comprehensive set of notes on standard errors, and couldn't find satisfactory ones - so I used this section as an excuse to make my own (sorry not sorry). We'll cover plain OLS, Eicker-Huber-White, Newey-West, Conley, and clustered standard errors. Just like calculating point estimates, it's incredibly important to get your standard errors right. You have to know what you don't know - the unknown unknowns are the dangerous ones.[1] This will be a fairly matrix-algebra-heavy section, so fasten your seatbelts, and let's get started. These notes are quite long: We'll have to gloss over some of the finer points in section, but you'll have this as a reference guide going forward.

Off we go

We're going to start with a little bit of a review. A standard error is of course an estimate of the uncertainty around an estimated parameter. It takes the general form of $se = \sqrt{\hat{\mathbb{V}}(\hat{\boldsymbol{\beta}})}$, and to estimate it, we have to make an estimate of the variance of our estimated parameter.[2] Suppose we have the following data generating process:

$$\mathbf{Y} = \mathbf{X}'\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

As you well know, we can write:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

With complete generality, we can write down the OLS variance-covariance matrix as:

$$\mathbb{V}(\hat{\boldsymbol{\beta}}) = \mathbb{V}((\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\Sigma}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

we know what $\mathbf{X}$ looks like, so that means that we know what $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ and what $\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$ look like - the only thing we don't (necessarily) know is what $\boldsymbol{\Sigma}$ looks like. But it turns out that writing this down in general terms is trivial. In the most general terms, we can write $\boldsymbol{\Sigma}$ as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \varepsilon_1\varepsilon_1 & \varepsilon_1\varepsilon_2 & \dots & \varepsilon_1\varepsilon_N \\ \varepsilon_2\varepsilon_1 & \varepsilon_2\varepsilon_2 & \dots & \varepsilon_2\varepsilon_N \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon_N\varepsilon_1 & \varepsilon_N\varepsilon_2 & \dots & \varepsilon_N\varepsilon_N \end{bmatrix}$$

[1] Thanks, Donny R.

[2] That's a lot of hats. We could have a small party. All that's saying is it's the *estimated* variance around the *estimated* parameter.

In the Beginning, [Econometricians] created the Spherical Error

When we make our standard OLS spherical errors assumption, however, this matrix simplifies to:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix}$$

To see this, remember that spherical errors means that $\mathbb{E}[\varepsilon\varepsilon'|\mathbf{X}] = \sigma^2\mathbf{I}$. This means that the variance of β, assuming spherical errors is[3]:

$$\begin{aligned} \mathbb{V}^{SE}(\hat{\boldsymbol{\beta}}) &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{\Sigma}\mathbf{X}'(\mathbf{X}'\mathbf{X})^{-1} \\ &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\sigma^2\mathbf{I}\mathbf{X}'(\mathbf{X}'\mathbf{X})^{-1} \\ &= \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} \\ &= \sigma^2(\mathbf{X}'\mathbf{X})^{-1} \end{aligned}$$

This should look familiar to you. Now let's talk about actually calculating standard errors on our beta coefficients. The good news is that $\mathbf{X}$ is data. So all we need is an estimate of σ^2. Remember that, under these assumptions, we can estimate σ^2 with $s^2 \equiv \frac{\mathbf{e}'\mathbf{e}}{n-k}$. This means that we can calculate the standard error of $\hat{\beta}_j$ as $se(\hat{\beta}_j) = \sqrt{s^2(X'X)^{-1}_{jj}}$. Let's show this using data.[4] As usual, we'll start by grabbing a few of our favorite things[5]. We'll also install four new packages, `Matrix`, `sandwich`, `HistData`, and `robustbase`:

```
install.packages('Matrix')
install.packages('sandwich')
install.packages('HistData')
install.packages('robustbase')
```

```
###### PACKAGES
library(dplyr)
library(ggplot2)
library(lfe)
library(Matrix)
library(sandwich)
library(HistData)
library(robustbase)

###### FUNCTIONS
as.tbl_df <- function(data) {
  dataset <- as.data.frame(data) %>%
    tbl_df()
```

[3] Some people call these "OLS" standard error. I think this is kind of weird terminology, because we run OLS and compute different standard errors all the time. But I am (fortunately) not in charge of the world of econometrics.

[4] I know we've calculated plain vanilla standard errors before - but for completeness' sake, we'll do it here too.

[5] Enter Maria von Trapp.

```
}

tbldfGrabber <- function(data, varnames) {
  matrixObject <- data %>%
    select_(.dots = varnames) %>%
    as.matrix()
}

miniOLS <- function(data, y, X) {
  n <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  return(betahat)
}


miniResids <- function(data, y, X) {
  n <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  e <- ydata - xdata %*% betahat
  output <- list(xdata, e)
  return(output)
}

########## RANDOMIZATION SEED
set.seed(12345)

########## GGPLOT SETUP
myThemeStuff <- theme(panel.background = element_rect(fill = NA),
        panel.border = element_rect(fill = NA, color = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.ticks = element_line(color = "gray5"),
        axis.text = element_text(color = "black", size = 10),
        axis.title = element_text(color = "black", size = 12),
        legend.key = element_blank()
        )
```

We're going to use the diamonds dataset from `ggplot2` for this exercise, because why not. Let's regress price on carats and depth. We'll want to run the regression, compute the residuals, and spit out the standard error on both coefficients, so let's update our OLS function to do this.

```r
diamonds <- mutate(diamonds, ones = 1)

olsSE <- function(data, y, X) {
  n <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  e <- ydata - xdata %*% betahat
  s2 <- (t(e) %*% e) / (n - k)
  XpXinv <- solve(t(xdata) %*% xdata)
  se <- sqrt(s2 * diag(XpXinv))
  olsOut <- list(betahat, se)
  names(olsOut) <- c("betahat", "se")
  return(olsOut)
}
```

Aaaaaand go:

```r
myReg <- olsSE(data = diamonds, "price", c("ones", "carat", "depth"))
myReg
```

```
## $betahat
##           price
## ones  4045.3332
## carat 7765.1407
## depth -102.1653
##
## $se
## [1] 286.205390  14.009367   4.635278
```

And, as usual, we'll check this against `felm()`:

```r
cannedReg <- felm(data = diamonds, price ~ carat + depth) %>%
  summary()
cannedReg
```

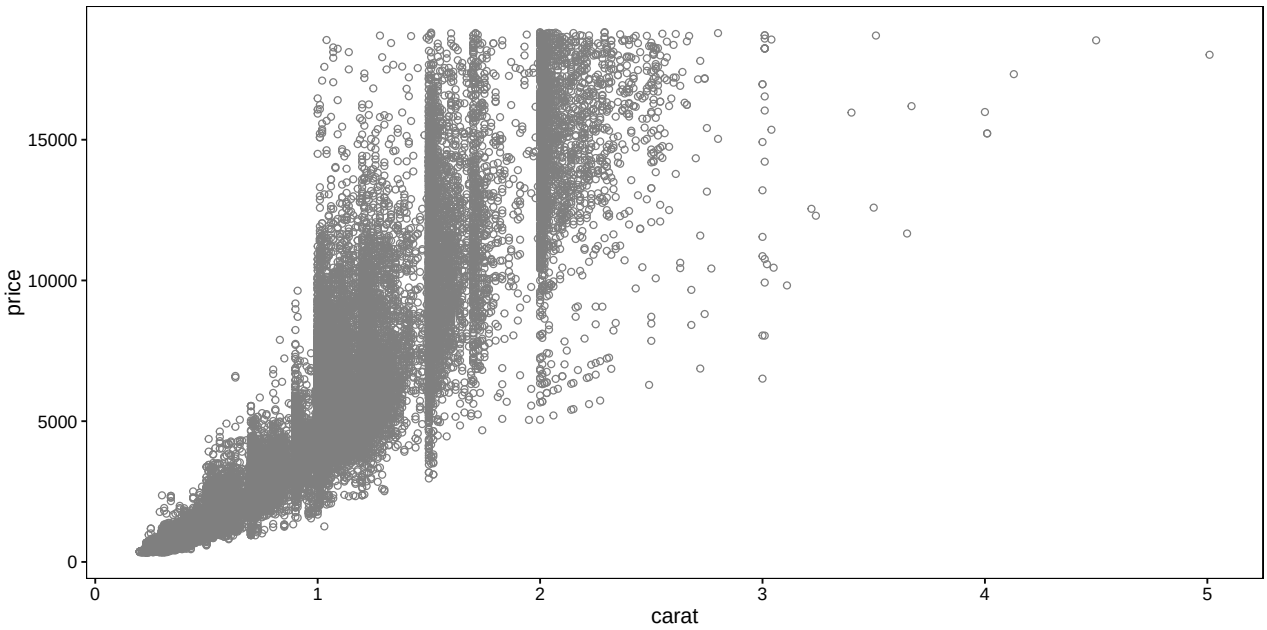
```
##
## Call:
##    felm(formula = price ~ carat + depth, data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18238.9   -801.6    -19.6    546.3  12683.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4045.333    286.205   14.13   <2e-16 ***
```

```
## carat       7765.141     14.009  554.28   <2e-16 ***
## depth       -102.165      4.635  -22.04   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1542 on 53937 degrees of freedom
## Multiple R-squared(full model): 0.8507   Adjusted R-squared: 0.8507
## Multiple R-squared(proj model): 0.8507   Adjusted R-squared: 0.8507
## F-statistic(full model):1.536e+05 on 2 and 53937 DF, p-value: < 2.2e-16
## F-statistic(proj model): 1.536e+05 on 2 and 53937 DF, p-value: < 2.2e-16
```

Looks good to me. But of course, we should make sure that our OLS assumptions make sense. One easy way to do this is to plot our data:

```
myPlot <- ggplot(data = diamonds, aes(y = price, x = carat)) +
  geom_point(color = "gray50", shape = 21) +
  myThemeStuff
```

```
myPlot
```



There are a bunch of things about this figure which should concern you, not least of which is that there seems to be severe discontinuities or bunching at different carat values.[6] From an OLS-assumption perspective, you should be very afraid that these data are definitely not homoskedastic. The higher the carat, the greater the variance in price. Our OLS standard errors are likely getting things wrong.

[6] Did I mention I was involved in buying two diamond rings in the last few years, and the market is a mess? That's a story for another day.

reg y x comma robust

Heteroskedasticity is scary - but thankfully, all is not lost![7] All we have to do is tweak our original assumptions a little bit. We'll retain the assumption that observations are *iid* draws, meaning that the off-diagonal terms of our $\mathbf{\Sigma}$ matrix will remain zero. But, importantly, we'll relax the homoskedasticity assumption.[8]

Assuming *iid* draws, we can invoke a Central Limit theorem, which gives us the following result[9]:

$$\frac{1}{\sqrt{N}} \sum_i X_i \varepsilon_i \xrightarrow{d} N(0, \mathbb{E}[X_i X_i' \varepsilon_i^2])$$

This gives us a $\mathbf{\Sigma}$ matrix that looks like this:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_N^2 \end{bmatrix}$$

This new matrix doesn't look too daunting, but instead of needing one estimate of σ^2, now we need a σ_i^2 estimate for each observation in our data. Crap!

We could deal with this in one of two ways. First, we could recognize that we could write the off-diagonal terms as $\sigma^2(X_i)$, where σ^2 is now a function of X_i (this is what heteroskedasticity means - the variance depends on the value of X_i), and then we could try to parametrically model this function. You should be very wary of anyone who says they can do that credibly.[10]

Good news, though! Thanks to Eicker, Huber, and White (since Max got his PhD from San Diego, he only thanks White), it turns out that we can actually use e_i^2 as non-parametric estimates of σ_i^2. Lucky for us, e_i^2 are consistent for the true errors, which lets us write (notice the new hats!)[11]:

$$\begin{aligned} \hat{\mathbb{V}}(\hat{\boldsymbol{\beta}}) &= (\mathbf{X'X})^{-1} \mathbf{X'} \hat{\mathbf{\Sigma}} \mathbf{X} (\mathbf{X'X})^{-1} \\ &= (\mathbf{X'X})^{-1} \left(\sum_i X_i X_i' e_i^2 \right) (\mathbf{X'X})^{-1} \end{aligned}$$

This is called a "sandwich" estimator, because we "sandwich" our estimate of the "meat", $\mathbf{X'\Sigma X}$ between "bread", $(\mathbf{X'X})^{-1}$. Delicious, delicious errors.

One great thing about sandwich estimators is that they only require us to estimate the k-by-k sandwich, rather than the n-by-n $\mathbf{\Sigma}$, which we couldn't do with only n observations!

---

[7] If it were, we'd have to close down a lot of economics departments. It's incredibly rare in actual data to have homoskedasticity.

[8] If you're a Stata person, the following is equivalent to typing ", robust" in your regression command - with non-panel data. If you're interested in running a regression with panel data in Stata, and have heteroskedasticity, check this out first. Shameless self-promotion, I know.

[9] Notice that this relies on an asymptotic result - so don't just go applying what we're about to do in small samples.

[10] This is especially true because often, heteroskedasticity results from the fact that a linear regression is a misspecified model of the true DGP: if we have a non-linear DGP and try to fit a straight line to it, we'll often end up with errors that are heteroskedastic. If you're not getting your model right in the first place, why should we trust you to get your model of your errors right? That's what I thought.

[11] If you don't trust me, the original math is here. Both Max's notes (section 5.1) and Wooldridge, pp. 171-172, have treatments of this as well.)

We commonly call the resulting standard errors "robust", or "heteroskedasticity-robust".[12] Not to be confused with "cluster-robust." We'll get to that. But let's not get ahead of ourselves.

Let's go ahead and implement this in R. We'll need to adjust our OLS function slightly. We'll need the `Diagonal()` function from the `Matrix` package to do this.

```
olsEHW <- function(data, y, X) {
  n <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  e <- ydata - xdata %*% betahat
  # create our estimated sigma matrix
  sigmamat <- Diagonal(n, e^2)
  XpXinv <- solve(t(xdata) %*% xdata)
  # create the full v-cov matrix
  vCov <- t(XpXinv) %*% t(xdata) %*% sigmamat %*% xdata %*% XpXinv
  se <- sqrt(diag(vCov))
  olsOut <- list(betahat, se)
  names(olsOut) <- c("betahat", "EHWse")
  return(olsOut)
}
```

And let's test this out:

```
myEHW <- olsEHW(data = diamonds, "price", c("ones", "carat", "depth"))
myEHW

## $betahat
##           price
## ones  4045.3332
## carat 7765.1407
## depth -102.1653
##
## $EHWse
## [1] 369.166140  25.104229   5.945381
```

Let's also compare this to the canned version in R:

```
# note we tell felm to get robust SE's when we look at the summary
cannedReg <- felm(data = diamonds, price ~ carat + depth) %>%
  summary(robust = TRUE)
cannedReg

##
```

[12] Or "White", or "Huber-White", or "Eicker-Huber-White", or ...

```
## Call:
##    felm(formula = price ~ carat + depth, data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18238.9   -801.6    -19.6    546.3  12683.7
##
## Coefficients:
##             Estimate Robust s.e t value Pr(>|t|)
## (Intercept) 4045.333    369.176   10.96   <2e-16 ***
## carat       7765.141     25.105  309.31   <2e-16 ***
## depth       -102.165      5.946  -17.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1542 on 53937 degrees of freedom
## Multiple R-squared(full model): 0.8507   Adjusted R-squared: 0.8507
## Multiple R-squared(proj model): 0.8507   Adjusted R-squared: 0.8507
## F-statistic(full model, *iid*):1.536e+05 on 2 and 53937 DF, p-value: < 2.2e-16
## F-statistic(proj model): 4.878e+04 on 2 and 53937 DF, p-value: < 2.2e-16
```

Boom! Spot on.[13] It's notable that the Eicker-Huber-White standard errors are larger than the regular standard errors - since you should be suspicious of everything, you should be happier with the EHW version. Another thing to be suspicious about: as we mentioned before, the EHW standard errors rely on asymptopia. They've been shown to be biased in finite samples, and unfortunately for us, biased towards zero.

Pop quiz: what would happen if you tried to Eicker-Huber-White, not only the diagonal elements, but the entire $\mathbf{\Sigma}$ matrix, by which I mean not assuming *anything* about the off-diagonal terms, and just estimating all of the $e_i' e_j$ terms? This would be bad news bears. The EHW-equivalent estimator of the general $\mathbf{X'\Sigma X}$ matrix would be $\sum_i \sum_j X_i X_j e_i e_j$ - but of course, by construction, $\sum_i X_i e_i = 0$, so that won't work. We'll have to turn to a different set of improvements over EHW to deal with the off-diagonals.

All about e

EHW standard errors are definitely an improvement over plain vanilla standard errors. But they obviously don't capture every situation. In particular, we might think that individual observations are correlated with one another. The classic example of this is in time series data. It's pretty common to think that the observation in time t is correlated with the observation in $t+1$ or in $t-1$. This means that the off-diagonal elements in our lovely $\mathbf{\Sigma}$ matrix will no longer be zero - or in other words, that our draws are no longer *iid*. We now need to estimate these elements too. But as you might have guessed, we're not going to have to come up with this estimator ourselves. This time, we have Newey and West to thank. Let's imagine a situation in which we have time series observations. We're going to introduce a new dataset for this, `Wheat` from the `HistData` package. Playfair used this dataset to make a classic figure in 1821, showing two time series: wheat, and typical weekly wages.[14]

[13] Note that if you were to run this in Stata, you'd get a slightly different answer, because Stata applies a degrees-of-freedom correction to its estimate. To recover the Stata robust standard errors, simply multiply your standard errors by $\sqrt{\frac{n}{n-k}}$.

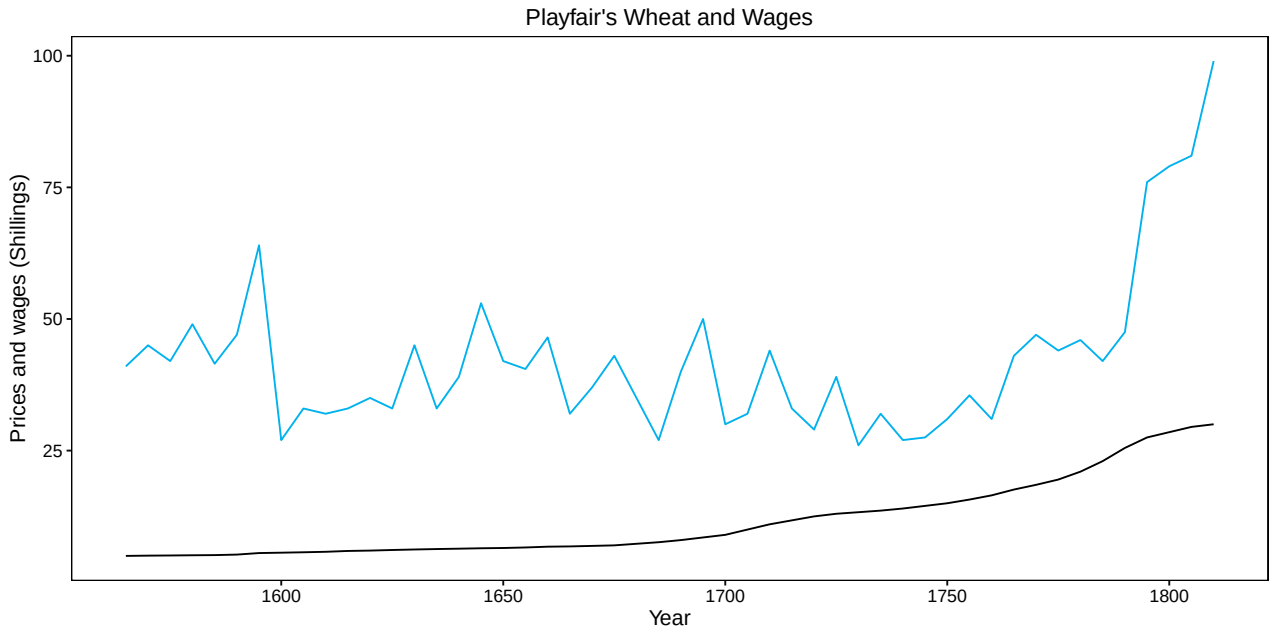
[14] Interested in the original? Check it out! Hashtag historical data visualization.

As with any dataset, we should take a look at it:

```
wheatData <- as.tbl_df(Wheat) %>%
  mutate(ones = 1) %>%
  na.omit()

wheatPlot <- ggplot(data = wheatData, aes(x = Year)) +
  geom_line(aes(y = Wheat), color = "deepskyblue2") +
  geom_line(aes(y = Wages)) +
  ggtitle("Playfair's Wheat and Wages") +
  xlab("Year") + ylab("Prices and wages (Shillings)") +
  myThemeStuff
```

```
wheatPlot
```



There's clear serial correlation here, suggesting that this dataset is a good candiate for using standard errors that are robust this correlation.

So: how do we estimate this thing? As always, we're trying to get an estimate of $\mathbf{\Sigma}$. Suppose we have T observations. In the presence of serial correlation between observations in period t and period s, our assumed Σ matrix looks like this (we're going to switch over to summation notation because it's much easier to deal with):

$$\mathbf{\Sigma} = \frac{1}{T}\sum_{t=1}^{T}\sum_{s=1}^{T}\rho_{|t-s|}X_t X_s'$$

where $\rho_{|t-s|}$ is the serial correlation term between two observations that are $t-s$ periods apart. To estimate this, we'll follow Newey and West and actually estimate the "meat" of our sandwich ($\mathbf{X'\Sigma X}$) all in one go. They write

the estimator for the "meat" as[15]:

$$\mathbf{X}'\hat{\mathbf{\Sigma}}\mathbf{X} = \frac{1}{T}\sum_t e_t^2 X_t X_t' + \frac{1}{T}\sum_{l=1}^{L}\sum_{t=l+1}^{T}\left(1 - \frac{j}{L+1}\right) e_t e_{t-l}[X_t X_{t-l}' + X_{t-l}X_t']$$

Gross. But let's break this down - it isn't actually quite as bad as it looks. Notice that the first component, $\frac{1}{T}\sum_t e_t^2 X_t X_t'$, is just the EHW estimate. We use that when $t = s$ - no lags (this would be $l = 0$). The observations with no lags appear on the diagonal of our $\mathbf{\Sigma}$ matrix. On the off-diagonals, now, though, we're allowing time period t's error term to be correlated with error terms that are up to l periods away. Rather than weighting all of the terms between t and $t \pm m$ equally, though, we'll assign higher weight to the terms closer to t. This is what the $\left(1 - \frac{l}{L+1}\right)$ term does. Notice that we need one new choice variable to estimate this matrix: L, the maximum distance between observations that we want to take into account in our error estimates. Usually, we use something like $L = T^{1/4}$. We call this estimator a HAC: Heteroskedasticity-Autocorrelation-Robust, because it both accounts for autocorrelation, and nests the heteroskedasticity-robust White estimator. Cool. Let's give coding it up a go:

```
olsNW <- function(data, y, X, L) {
  # usual setup (note now we're calling n t instead)
  #   hashtag timeseries mindset
  t <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  # note that we now need our residuals to be a vector not a mat
  #  so that we can scalar-multiply them by our xmatrix
  e <- as.vector(ydata - xdata %*% betahat)
  # emat is the scalar product of our residuals with the x mat
  emat <- e * xdata
  # sets up our Newey-West weights -
    # they will start at 1, and max out at  1 - l/(L + 1)
  weights <- seq(1, 0, by = -(1/(L + 1)))
  # initialize our eMat'eMat matrix - note NOT  = e'e
  #    scale by 1/2 bc we're only estimating half of the matrix
  eMatpeMat <- 0.5 * crossprod(emat) * weights[1]
  # create the lag-weighted terms we're going to add to our main mat
  #   this spits out a list (note: lapply not sapply) of matrices
  weightStep <- lapply(2:length(weights), function(l) {
    # crossprod(x,y) is just like t(x) %*% y, but faster
    weights[l] * crossprod(emat[1:(t - l + 1), ], emat[l:t,])
  })
  # now actually add up all of the weighted summation terms
  #   Reduce() consecutively applies a fxn to each element of a list
  eMatpeMat <- eMatpeMat + Reduce("+", weightStep)

  # fully populate the matrix - so far we've just estimated the lower diagonal
  eMatpeMat <- eMatpeMat + t(eMatpeMat)
```

[15]The original paper can be found here. There's also a discussion of this estimator in Max's notes. I use his notation here.

```
  # create the "meat" of the sandwich by multiplying this big matrix by 1/t
  meat <- eMatpeMat/t
  # create the bread (as usual)
  bread <- solve(t(xdata) %*% xdata)
  # finally, the full sandwich: multiply by t to get everything to work out
  sandwich <- t * (t(bread)  %*% meat  %*% bread)
  # as usual, se's are the sqrt of the diagonal of the sandwich
  se <- sqrt(diag(sandwich))
  # finally done!
  output <- list(betahat, sandwich, se)
  names(output) <- c("betahat", "sandwich", "se")
  return(output)
}
```

This code is a little ugly - you might have to stare at it for a while (read: quite a while) to figure out exactly what's going on. Thankfully, for your future life, `R` has a canned routine, helpfully named `NeweyWest()`, and belonging to the `sandwich` package, that will do this for you. Let's do our due dilligence and test our estimates against the canned ones. We'll use the $T/4$ rule of thumb to compute the optimal lags. Note that the canned routine only takes `lm()` objects, not `felm()` objects - but you shouldn't worry too much about this, since you'll use Newey-West standard errors in time series land, and likely not have that many fixed effects to worry about anyway.

We'll start by grabbing our own estimates.[16]

```
optimalLags <- (nrow(wheatData))^(1/4)
myNW <- olsNW(wheatData, "Wheat", c("ones", "Wages"), optimalLags)
```

A couple of things to note about the canned routine: it doesn't actually directly spit out standard errors; instead, it returns either the meat or the full sandwich matrix. Luckily, we're pretty good at dealing with that at this point.

```
wheatModel <- lm(data = wheatData, Wheat ~ Wages)
cannedNW <- NeweyWest(wheatModel, lag = optimalLags,
                      prewhite = FALSE, sandwich = TRUE)
cannedNWSE <- sqrt(diag(cannedNW))
```

Before we look at this, let's actually take a quick look at the regular standard error estimates:

```
# canned - regular SE
summary(wheatModel)

##
## Call:
## lm(formula = Wheat ~ Wages, data = wheatData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

[16] Note that for this to work, your data needs to be ordered along the time dimension.

```
## -18.163  -8.895  -2.351   7.369  35.176
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  28.5047     3.2587   8.747 1.67e-11 ***
## Wages         1.1773     0.2384   4.939 9.92e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.24 on 48 degrees of freedom
## Multiple R-squared:  0.337,Adjusted R-squared:  0.3231
## F-statistic: 24.39 on 1 and 48 DF,  p-value: 9.922e-06

# our SE estimates
myNW$se

##      ones     Wages
## 4.9733139 0.4908693

#canned SE estimates
cannedNWSE

## (Intercept)       Wages
##   4.9733139   0.4908693
```

As we might expect, the Newey-West standard errors are larger than the ordinary standard errors in the presence of serial correlation.

"It's like we finish each other's..." "Sandwiches!"

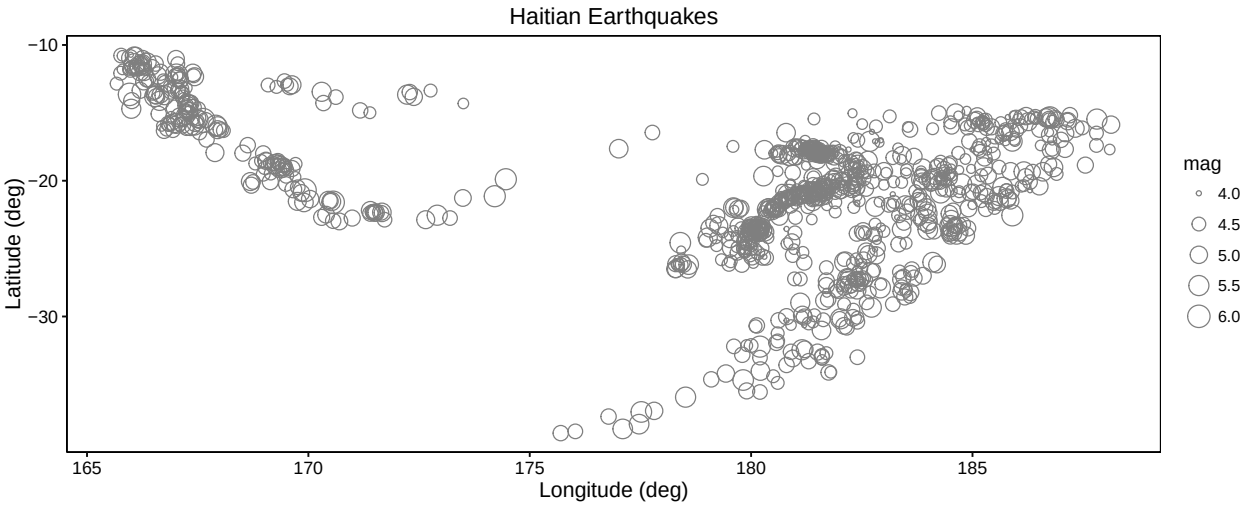
Newey-West standard errors are an important econometric advance - but it's spatial data, not time series data, that're all the rage these days.[17] What if instead of having a time series of wheat prices, we have a cross-sectional dataset on, say, the locations of earthquakes?[18] The `datasets` package contains just such a dataset, of the locations of earthquakes (as well as their depth, and Richter scale magnitude) in the vicinity of Haiti, `quakes`. Of course, before we do anything with this dataset, we should go ahead and plot it. This will be our first foray into plotting spatial data using `ggplot2` - more on that later in the semester.

```
quakePlot <- ggplot(data = quakes, aes(x = long, y = lat, size = mag)) +
  geom_point(shape = 21, color = "gray50") + myThemeStuff +
  ggtitle("Haitian Earthquakes") + xlab("Longitude (deg)") + ylab("Latitude (deg)")
```

[17] You in the back asking about panel data - hold your darn horses.

[18] Obviously, these earthquakes didn't happen all at once. But suppose for some reason we don't observe the time dimension. Or that we had a dataset of something that did happen all at once. Humor me and suspend your econometric disbelief.

```
quakePlot
```



There's clear spatial dependence in the magnitudes of these earthquakes. So - we should again be worried that our spherical errors assumption is violated.

Just like we can use Newey-West standard errors when we have a time series with serially correlated error terms, there exists an analagous estimator for spatially correlated error terms, thanks to Conley.[19] Conley standard errors are also a form of HAC - except rather than being robust to *temporal* autocorrelation, they're robust to *spatial* autocorrelation. The good news is that the intuition carries over nearly identically from Newey-West. The original Conley paper is written as a GMM estimator, but we can write up something that should look familiar in our notation:

$$\mathbf{X}'\hat{\mathbf{\Sigma}}\mathbf{X} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} X_i X_j' e_i e_j K(d_{ij})$$

where $K()$ is a kernel function, and d_{ij} represents the physical distance between unit i and unit j. Just like in the Newey-West setup, we have that our "meat" depends on how far apart observations are. Rather than specifying lags, in this context, we specify a kernel to use - Conley himself recommends uniform, but other people like Bartlett (triangular) or Epanechnikov. A uniform kernel, for example, will weight all observations that are within distance v of observation i equally, and all observations further than v as zero. Rather than chat too much more about it, let's go ahead and code it up.[20] This code is modified from Sol's Stata code[21] :

```
olsConley <- function(data, y, X, lat, lon, cutoff) {
  # usual setup
```

[19] The original paper is here, and a 2009 review article is here. Conley HAC standard errors have become popular in empirical microeconomics recently in large part due to Sol Hsiang's Stata and Matlab code, which he posted on his blog here. Sol has gotten a ton of citations on his 2010 PNAS paper because of this code. This is a solid (Sol-id?) strategy. Thiemo Fetzer has since contributed an update of this code which run more quickly, and added an R function as well. Both of these guys' code allows for both cross-sectional *and* time series dependence, but let's not get ahead of ourselves.

[20] You should feel excited about this code because *ooh, spatial correlation!*, but also because, as far as I can tell, this code doesn't exist on the internet (and I'm a pretty good Googler). Sol only has code for Stata and Matlab, and Thiemo's code only works for panel data. So there you are: an ARE 212 section world premiere.

[21] His code is actually written in Mata - you're welcome.

```
n <- nrow(data)
k <- length(X)
ydata <- tbldfGrabber(data, y)
xdata <- tbldfGrabber(data, X)
betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
e <- ydata - xdata %*% betahat

# grab latitude & longitude
latdata <- tbldfGrabber(data, lat)
londata <- tbldfGrabber(data, lon)

# loop over all of the spatial units (aka observations)
meatWeight <- lapply(1:n, function(i) {
  #  turn longitude & latitude into KMs. 1 deg lat = 111 km, 1 deg lon = 111 km* cos(lat)
        lonscale <- cos(latdata[i]*pi / 180) * 111
        latscale <- 111

        # distance --> use pythagorean theorem! who knew that was useful?
        dist <- as.numeric(sqrt((latscale*(latdata[i] - latdata))^2
                 + (lonscale*(londata[i] - londata))^2))

        # set a window var = 1 iff observation j is within cutoff dist of obs i
        window <- as.numeric(dist <= cutoff)
        # this next part is where the magic happens. this thing makes:
        #    sum_j(X_iX_j'e_ie_j K(d_{ij})), and we make n of them - one for each i.


        # double transpose here is because R is bad at dealing with 1 x something stuff.
        #   we want x_i'; this is an easy way to get it. Now for some dimensions
        #   (we want k x k at the end):

        XeeXh <-  ((t(t(xdata[i, ]))  %*% matrix(1, 1, n) * e[i,]) *
            #       k x 1               1 x n         1 x 1
                (matrix(1, k, 1) %*% (t(e) * t(window)))) %*% xdata
            #         k x 1              1 x n                n x k
        return(XeeXh)

})

# phew! Now let's make our sandwich. First, the meat = sum_i what we just made
meat <- (Reduce("+", meatWeight)) / n
# and the usual bread
bread <- solve(t(xdata) %*% xdata)
# mmmm, delicious sandwich
sandwich  <- n* (t(bread) %*% meat %*% bread)
# se as per usual
se <- sqrt(diag(sandwich))
```

```
  output <- list(betahat, se)
  names(output) <- c("betahat", "conleySE")
  return(output)
}
```

You're seeing the nice pretty version of this - but it's still a little complicated, so let's take a minute to breathe it in.[22] Aaaahhh. You should notice that this thing looks a lot like the Newey-West estimator we wrote up above. Really, the only differences are in the "weights." First, Newey-West uses distances between observations in time as its measure of distance; Conley uses distances between observations in space, which makes the "distance" thing a little more complicated. Second, Newey-West weights decay away from observation t, and eventually become zero. Conley weights (with a uniform kernel) are equal within some radius of observation i, and then immediately become zero outside that radius.[23] Unlike with Newey-West standard errors, there isn't a rule of thumb yet (that I know of) on the optimal distance cutoff - you'll just have to play around with this a little bit.

Let's go ahead and apply these new standard errors to our earthquake data. We'll regress earthquake depth on magnitude. Recall that our `olsConley()` function takes the regular OLS stuff, plus a latitude variable and a longitude variable, and finally a cutoff as arguments[24]:

```
quakeData <- as.tbl_df(quakes) %>%
  mutate(ones = 1)

# conley SEs
myConley <- olsConley(quakeData, "depth", c("ones", "mag"), "lat", "long", 100)
```

We can't actually compare these to standard errors from a canned routine, because there isn't one(!) But I can promise you that these are correct, since I painstakingly checked them against Sol's code. We can, however, compare them to regular (canned) SEs:

```
cannedQuakes <- summary(felm(data = quakeData, depth ~ mag))$coefficients[,2]

myConley

## $betahat
##          depth
## ones  881.6250
## mag  -123.4209
##
## $conleySE
##      ones       mag
## 109.04809  19.27074

cannedQuakes
```

[22] Next time you see Matt Woerman, you should thank him profusely - he and I stared at the matrix multiplication going on here for quite some time.

[23] You could change this and instead use a Bartlett kernel with Conley, which would give decaying weights inside our radius and zero weight outside. Left as an exercise for the (intrepid) reader, but not actually that hard to do. Just modify the `window` term so that it's the current `window` term times a weight term that depends on distance.

[24] Be careful about how big you make your cutoffs. What do you expect to happen if you make your cutoff too big?

```
## (Intercept)         mag
##    76.44439    16.48253
```

Again, larger. Are you sensing a theme? If anybody ever tells you they estimated standard errors under the spherical assumption, you should be highly skeptical.[25]

Charlie Foxtrot?

What we've all been waiting for! Clustered standard errors have become standard practice among applied microeconometricians.[26] The idea is pretty simple: suppose that every observation belongs to (only) one of G groups. The assumption we make when we cluster is that there is no correlation *across* groups - but we will allow for arbitrary within-group correlation. My favorite cross-sectional example to use to think about these groups is to considers individuals within villages. In many cases, it's pretty reasonable to think that individuals' error terms are correlated within a village, but that individuals' errors aren't correlated across villages.[27] If you don't like development, imagine instead having multiple observations of an individual over time. In math, we can write down the following DGP:

$$y_{ig} = \mathbf{x'}_{ig}\boldsymbol{\beta} + \varepsilon_{ig}$$

where $i = 1, \ldots, N$ indexes individuals and $g = 1, \ldots, G$ indexes groups. We're still going to invoke our usual assumption that $\mathbb{E}[\varepsilon_{ig}|\mathbf{x}_{ig}] = 0$, but now we'll invoke our new assumption: $\mathbb{E}[\varepsilon_{ig}\varepsilon'_{jg'}|\mathbf{x}_{ig}, \mathbf{x}_{jg}] = 0 \; \forall g \neq g'$. If we now stack all of the observations from the gth cluster, we can write our DGP as:

$$\mathbf{y}_g = \mathbf{X}_g\boldsymbol{\beta} + \boldsymbol{\varepsilon}_g$$

where $\mathbf{y}_g$ and $\boldsymbol{\varepsilon}_g$ are $N_g \times 1$, $\mathbf{X_g}$ is $N_g \times k$, and there are N_g individuals in cluster g. Stack one more time, and we get the DGP we're used to:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

With this DGP in mind, we can write:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X'X})^{-1}\mathbf{X'y} = \left(\sum_{g=1}^{G} \mathbf{X'_g X_g}\right)^{-1} \sum_{g=1}^{G} \mathbf{X'_g y_g}$$

This assumption generates a block-diagonal $\boldsymbol{\Sigma}$ matrix[28]:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Omega}_1 & 0 & \ldots & 0 \\ 0 & \boldsymbol{\Omega}_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \boldsymbol{\Omega}_G \end{bmatrix}$$

[25] Pro tip, though: don't be that guy (woman) in a seminar who asks about standard errors.

[26] This is thanks in large part to a seminal paper by Bertrand, Duflo, and Mullainathan in the QJE, which shows that we dramatically over-reject the null hypothesis in difference-in-differences designs when we fail to account for error dependence. Matt Woerman, Louis Preonas, and I are working on two papers (Open Science Framework page here), thanks to generous funding from BITSS, that also address these issues - look out for them at the end of 2016.

[27] It's also potentially reasonable to think of situations in which this isn't true - again, see Burlig, Preonas, Woerman (not yet existant) for a discussion, but let's go with it for now.

[28] Notice that we don't need to make the assumption that each cluster is the same size - and our code will be flexible in this regard.

where

$$\mathbf{\Omega}_g = \begin{bmatrix} \varepsilon_1^g \varepsilon_1^g & \varepsilon_1^g \varepsilon_2^g & \dots & \varepsilon_1^g \varepsilon_N^g \\ \varepsilon_2^g \varepsilon_1^g & \varepsilon_2^g \varepsilon_2^g & \dots & \varepsilon_2^g \varepsilon_N^g \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon_N^g \varepsilon_1^g & \varepsilon_N^g \varepsilon_2^g & \dots & \varepsilon_N^g \varepsilon_N^g \end{bmatrix}$$

We can express this in "meat" notation, too:

$$\mathbf{X'\Sigma X} = \sum_{g=1}^{G} \mathbf{X'_g} \boldsymbol{\varepsilon_g} \boldsymbol{\varepsilon_g}' \mathbf{X_g}$$

As usual, we're looking for an estimator for the "meat". Turns out that White came up with one in 1984 (and a number of econometricians have since improved our understanding of it).[29] We can express the 'cluster-robust" estimate of the "meat" as:

$$\mathbf{X'\hat{\Sigma}X} = \sum_{g=1}^{G} \mathbf{X'_g e_g e'_g X_g}$$

As with all of these sandwich estimators, clustered standard errors rely on asymptopia. General rules of thumb are that you shouldn't use this estimator with fewer than 40 clusters - but even more than that is better.[30]

Why do people like clustered standard errors so much? In my mind, the feature that people like the most is that they allow for arbitrary dependence between observations within a cluster - this is really non-parametric, which is great. Also, we like to think about groups - it's easy to cluster by, say, state, and we don't have to be as restrictive in choosing weights as we did with Conley and Newey-West. Another advantage is that clusters play nicely with panel data: if you cluster at the individual level in a individual-by-date panel, you're allowing for arbitrary dependence between *any* two observations in that individual's time series, regardless of when they appear.[31] Even better, if you have data on individuals in multiple counties by date, you could cluster at the *county* level and allow for arbitrary dependence between any two individual-dates within that county, which is very flexible.

What are the downsides of clustering? First, you need a lot of clusters for the estimate to be credible. Second, when you cluster, you're still making a parametric assumption - you're restricting the off-block-diagonal terms of your $\mathbf{\Sigma}$ matrix to be zero, which is pretty strong. This is particularly concerning when you think, for example, that people living close to state borders look more alike than people living far away from the borders. Clustering at the state level will do a bad job of capturing this dependence.

At the end of the day, the right answer with standard errors is always to think carefully about what you think the true DGP is, and then estimate a bunch of different types of standard errors that make sense. Of course, you also always want to be conservative. So: clustered standard errors are often great, but they're not the be-all and end-all (just like, for example, RCTs are great but aren't the right tool for every job).[32] Okay. End rant.

Given all of our experience with Newey-West and Conley, which are actually much more complicated, coding the clustered standard error estimator up should be a sinch.

[30] If you really have very few clusters, you can use Cameron, Gelbach, and Miller's Wild Cluster Bootstrap to improve your standard error estimates.

[31] This is nicer than Newey-West in instances where we think, for example, that my electricity consumption on Wednesdays is super highly correlated, but less so than my Wednesday to Thursday consumption, which would receive higher weight in Newey-West than Wednesday to Wednesday a week later.

[32] Hashtag antagonizingtherandomistas. Hashtag wannaberandomista.

```r
olsCluster <- function(data, y, X, clustervar) {
  # usual setup
  n <- nrow(data)
  k <- length(X)
  ydata <- tbldfGrabber(data, y)
  xdata <- tbldfGrabber(data, X)
  # grab the cluster column too
  clusterdata <- tbldfGrabber(data, clustervar)

  # grab each cluster identifier
  clusters <- unique(clusterdata)
  # number of clusters
  G <- length(clusters)

  betahat <- solve(t(xdata) %*% xdata) %*% t(xdata) %*% ydata
  e <- ydata - xdata %*% betahat


  # loop over all of the clusters
  clusterMeat <- lapply(1:G, function(g) {
    # which obs belong to cluster g?
    gindex <- which(clusterdata == clusters[g])
    # x data for cluster g only
    Xg <- xdata[gindex, ]
    # resids for cluster g only
    eg <-  matrix(e[gindex, ])

    # calculate each cluster's meat
    XeeXg <- t(Xg) %*% eg %*% t(eg) %*% Xg
     #Dims:  k x ng   ng x 1   1 x ng   ng x k
    return(XeeXg)
  })
  # sandwich as per usual
  meat <- (Reduce("+", clusterMeat)) / n
  bread <- solve(t(xdata) %*% xdata)
  # apply a DoF correction
  dfc <- (G/(G-1))*((n-1)/(n-k))
  sandwich <- dfc * n * t(bread) %*% meat %*% bread

  # se as per usual
  se <- sqrt(diag(sandwich))
  output <- list(betahat, se)
  names(output) <- c("betahat", "ClusteredSE")
  return(output)
}
```

That wasn't so bad. Let's test it. We'll use the `NOxEmissions` dataset from `robustbase` for this. This is a dataset

of hourly NO_x readings, including NO_x concentration, automobile emissions, and windspeed. We're going to use the observation data as our cluster variable. This allows for arbitrary dependence between observations in the same day, and zero correlation across days. Is this reasonable? Maybe. But we'll go with it for now:

```
nox <- as.tbl_df(NOxEmissions) %>%
  mutate(ones = 1)

myNoClusters <- felm(data = nox, LNOx ~ sqrtWS)

myClusters <- olsCluster(nox, "LNOx", c("ones", "sqrtWS"), "julday")
# using more of the felm() syntax: the 0's say no FE, no IV
#   for now. the last entry is the cluster variable
cannedClusters <- felm(data = nox, LNOx ~ sqrtWS |0|0| julday)

# regular SE
(summary(myNoClusters))$coefficients

##               Estimate Std. Error   t value Pr(>|t|)
## (Intercept)  5.5588538 0.02911941 190.89856        0
## sqrtWS      -0.8644279 0.02018434 -42.82666        0

# our clustered SE
myClusters

## $betahat
##              LNOx
## ones    5.5588538
## sqrtWS -0.8644279
##
## $ClusteredSE
##       ones     sqrtWS
## 0.06475863 0.04775083

# canned clustered SE
(summary(cannedClusters))$coefficients[,2]

## (Intercept)      sqrtWS
##  0.06475863  0.04775083
```

Spot. On. I'm as surprised as you are! As we've seen over and over again, the regular standard errors are smaller than the clustered standard errors. Be aware, though, that this need not necessarily be the case - if you have negative correlation between observations within a cluster, clustered standard errors will often be *smaller* than regular standard errors, so don't panic.

Bacon, lettuce, *and* tomato!

In case you weren't already at the point where you never want to see a standard error ever again, I want to bring your attention to two more things that you can do. We won't code them up, nor will we spend a long time

discussing them, because they're both panel data techniques, which is mostly beyond the scope of this class, but you should know that they exist.

If you have panel data, and think that there might be *both* serial correlation and temporal correlation, you can use extensions to the methods we've shown today. If you have a large panel, you could take (one of two) clustering approaches: either cluster at a higher-level unit, thereby allowing for even arbitrary-ier arbitrary dependence, or *two-way* cluster. This sounds daunting, but is essentially just calculating two sets of meat: a spatial meat (just like we did above), and a temporal meat (use time periods instead of cluster identifiers and you're right on track), and then adding them up before you calculate your sandwich. Good news: `felm()` will do this for you - and in all likelihood, by the time you're doing this, you'll be done with ARE 212 and no longer subject to the canned routine ban.

The alternative to clustering is to combine the Conley approach with the Newey-West approach to get standard errors that are robust to both spatial correlation and temporal correlation. To do this, all you have to do is to compute the "meat" part of the sandwich using the Conley formula within each time unit, and using the Newey-West formula within each panel unit. Add all of these guys up to create the mega meat, and then do the usual sandwich transformation to get the standard errors. `felm()` won't do this for you (yet?), but this is what Sol's code does, and what Thiemo's does as well, so you have it at your disposal should you be interested.

Next week, we'll move out of standard error land (I know you're not *not* sad), and talk about instrumental variables.