We'll start section with a review of some of the common issues on the problem set. Then, we'll go over GLS and we'll use an empirical example to try out categorical dummies. Along the way, we'll try out some more advanced tools of the trade, in particular exporting to and using the advanced graphing package `ggplot2`.

Problem set retrospective

TBD

GLS

This section will briefly outline two general concepts, GLS and `ggplot`. We will examine the characteristics of generalized least squares (GLS), and specifically the efficiency gains from a special case of GLS, weighted least squares (WLS). We will then recreate the graphs from Figures 2.6 and 2.7, roughly, in the notes using `ggplot2` a very popular graphing package in `R`. This is part is optional, especially since it is only a very brief treatment of the package — there is a lot more to learn.

Let $x \sim U(0, 2000)$ and $\epsilon \sim N(0, (x/1000)^2)$. The underlying data generating process in (2.102) is $y_i = \alpha + x_i\beta + \epsilon$, where $\alpha = 0.5$ and $\beta = 1.5$. The objective is to plot the simulated sampling distribution of the OLS estimator applied to $B = 10,000$ draws, each of size $n = 1000$. First, let's generate the sample data for one draw.

```
n <- 1000
x <- runif(n, min=0, max=2000)
eps <- rnorm(n, 0, sqrt((x/1000)^2))
y <- 0.5 + x*1.5 + eps
```

Now we can calculate the standard OLS parameter vector $[\hat{\alpha} \;\; \hat{\beta}]'$ by noting that $\mathbf{X}$ is just the x vector bound to a column of ones. We will only examine $\hat{\beta}$ for this section, rather than both parameters.

```
X <- cbind(1, x)
params <- solve(t(X) %*% X) %*% t(X) %*% y
beta <- params[2]
print(beta)
```

```
[1] 1.499942
```

Let's package this into a function, called `rnd.beta`, so that we can collect the OLS parameter for an arbitrary number of random samples, noting that n is a constant so we may as well keep it out of the function so that 1000 is not reassigned thousands of times to n.

```
rnd.beta <- function(i) {
  x <- runif(n)
  eps <- rnorm(n, 0, sqrt(x/10))
  y <- 0.5 + x * 1.5 + eps
```

```
  X <- cbind(1, x)
  params <- solve(t(X) %*% X) %*% t(X) %*% y
  beta <- params[2]
  return(beta)
}
```

Since there aren't any supplied arguments, the function will return an estimated $\hat{\beta}$ from a different random sample for each call:

```
rnd.beta()
rnd.beta()
```

```
[1] 1.471854
[1] 1.512609
```

We could do this in a `for` loop (like last time), but for pedagogical purposes and to be more `R`-ish, this time we'll use `sapply` to apply the function to a list of effective indices. Now replicating the process for B draws is straightforward:

```
B <- 100
beta.vec <- sapply(1:B, rnd.beta)
head(beta.vec)
mean(beta.vec)
```

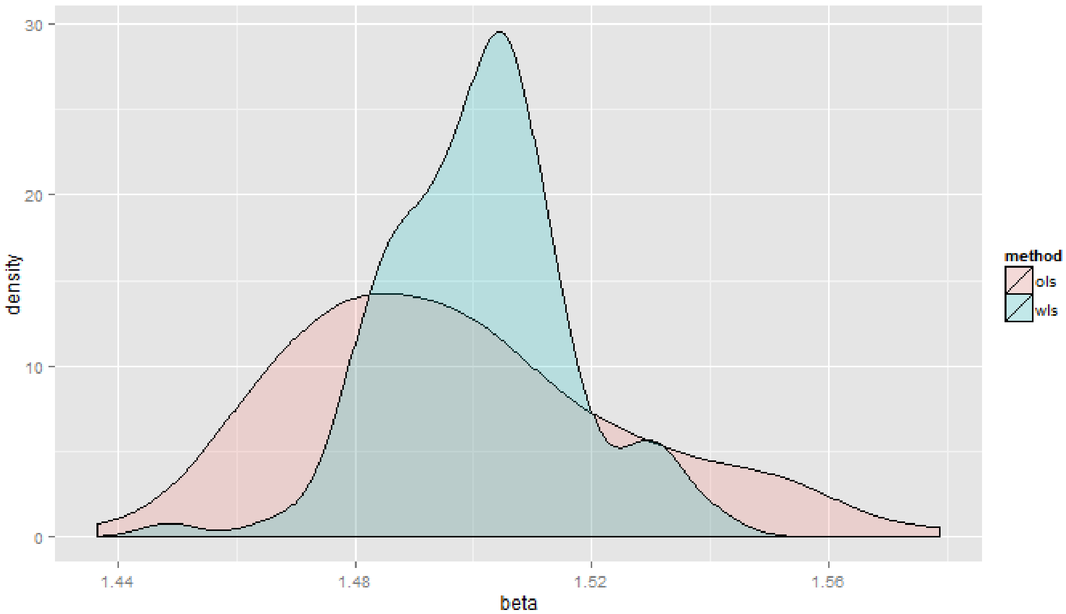
```
[1] 1.513552 1.483372 1.491940 1.536422 1.506401 1.556708
[1] 1.497148
```

All right. Looking good. The average of the simulated sample is much closer to β than any individual call of `rnd.beta`, suggesting that the distribution of the simulated parameters will be unbiased. Now, let's create another, similar function that returns the WLS estimates.

```
rnd.wls.beta <- function(i) {
  x <- runif(n)
  y <- 0.5 + x * 1.5 + rnorm(n, 0, sqrt(x / 10))
  C <- diag(1 / sqrt(x / 10))
  y.wt <- C %*% y
  X.wt <- C %*% cbind(1, x)
  param.wls <- solve(t(X.wt) %*% X.wt) %*% t(X.wt) %*% y.wt
  beta <- param.wls[2]
  return(beta)
}
wls.beta.vec <- sapply(1:B, rnd.wls.beta)
```

We now have two collections of parameter estimates, one based on OLS and another based on WLS. It is straightforward to plot two separate histograms using `R`'s core histogram plotting function `hist()`. However, we can use this to introduce a more flexible, powerful graphing package called `ggplot2`.

```
library(ggplot2)
labels <- c(rep("ols", B), rep("wls", B))
data <- data.frame(beta=c(beta.vec, wls.beta.vec), method=labels)
ggplot(data, aes(x=beta, fill=method)) + geom_density(alpha=0.2)
```



Returns to education example

The purpose of this section is to estimate the returns to education using `R`. There is nothing valid about the results found in this section; but the empirical application gives us a chance to explore categorical dummies and the `ggplot2` package. First, as always, we load the required libraries.

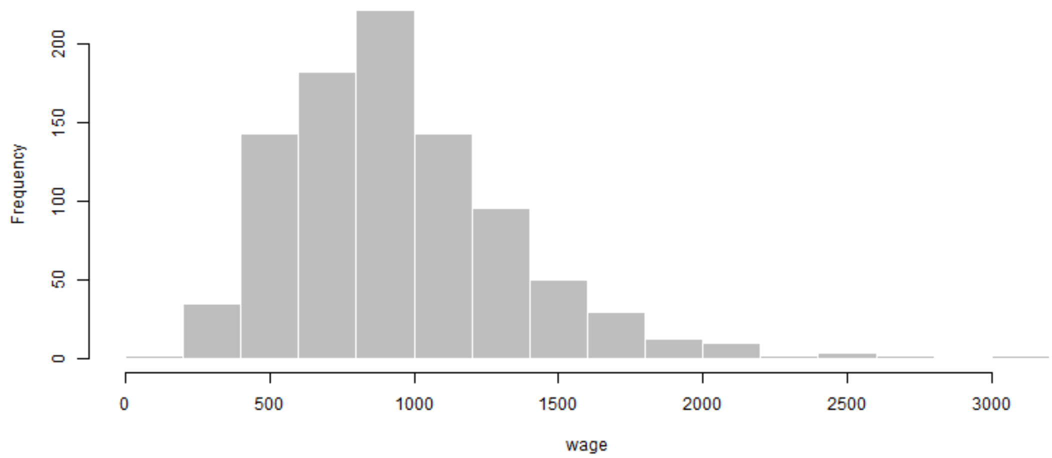
```
library(foreign)
library(ggplot2)
library(xtable)
```

We can then read the wage data directly from the online repository for the supplementary data sets for the Wooldridge (2002) text. You will need an internet connection. We only need the `wage`, `educ`, and `age` variables, and we omit all observations with missing observations using `na.omit()`.

```
f <- "http://fmwww.bc.edu/ec-p/data/wooldridge/wage2.dta"
data <- read.dta(f)
data <- data[ , c("wage", "educ", "age")]
data <- na.omit(data)
```

A quick visualization reveals the distribution of wages in the data set:

```
hist(data$wage, xlab = "wage", main = "", col = "grey", border = "white")
```



Before we continue, I'll introduce the `%in%` operator, since we'll be using it shortly. The `%in%` operator generates a boolean vector[1], depending on whether or not the element in the variable that preceeds is contained within the variable that follows it. That's a confusing explanation. To make this more clear, here's an example:

```
4:12 %in% 1:30
c(5:7) %in% c(1,1,2,3,5,8,13)
c("green","blue","red") %in% c("blue","green")
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[1]  TRUE FALSE FALSE
[1]  TRUE  TRUE FALSE
```

We can use `%in%` along with the `ifelse()` command to easily create dummy variables from variables that take on more than two values, like `educ`. Roughly following page 38 of the lecture notes, we create a rough measure of educational attainment from the `educ` variable.

```
e1 <- ifelse(data$educ %in% 1:12, 1, 0)
e2 <- ifelse(data$educ %in% 13:14, 1, 0)
e3 <- ifelse(data$educ %in% 15:16, 1, 0)
e4 <- ifelse(data$educ %in% 17:18, 1, 0)
```

The categorical education variables sum to one, and the `lm()` function will force-drop one of the variables. Note that the intercept in this regression reflects the mean wage of the `e4` class. The other coefficients reflect the relative wages of the other three classes.

```
lm(wage ~ 1 + e1 + e2 + e3 + e4, data = data)
coef(summary(m1 <- lm(wage ~ 1 + e1 + e2 + e3 + e4, data = data)))
```

```
Call:
lm(formula = wage ~ 1 + e1 + e2 + e3 + e4, data = data)
```

[1]A boolean vector is a vector composed entirely of =TRUE=s and =FALSE=s.

```
Coefficients:
(Intercept)          e1          e2          e3          e4
    1196.96     -350.46     -229.97      -90.51          NA
              Estimate Std. Error   t value      Pr(>|t|)
(Intercept) 1196.95876   38.93314 30.743953 7.969254e-144
e1          -350.46396   42.67867 -8.211689  7.239709e-16
e2          -229.97111   49.22796 -4.671555  3.429280e-06
e3           -90.50748   47.64240 -1.899726  5.777793e-02
```

Interpretation is a common challenge in dummy variables, particularly when we start including more complicated dummies or interaction terms. It's a good idea to think hard about what the coefficients from a regression represent. To sharpen your intuition in this regard, we'll use our own `OLS()` function to return the coefficients from a regression of wage on the dummy variables.

```
OLS <- function(y,X) {
  return(solve(t(X) %*% X) %*% t(X) %*% y)
}
X <- cbind(1,e1,e2,e3,e4)
y <- data$wage
b <- OLS(y,X)
```

```
Error in solve.default(t(X) %*% X) (from #2) :
  system is computationally singular: reciprocal condition number = 1.53843e-18
```

Uh oh! What happened? `OLS()` is not as smart as `lm()`, so it didn't automatically drop any of our dummy variables. Since the dummies sum to a column vector of ones, we violated A2: `X` does not have full column rank. We have a couple of options here: first, we can try dropping the intercept.

```
(b_dropint <- OLS(y,X[ , 2:5]))
```

```
        [,1]
e1  846.4948
e2  966.9877
e3 1106.4513
e4 1196.9588
```

We can show (but won't) that the coefficients are just the average wage amongst each dummy group. Think of each dummy here as forming its own intercept. Since there are no other covariates, each captures the average wage for all of the observations in that group. We can also see that b_4 corresponds to the intercept from the `lm()` output, which is as we expect. It's also simple arithmatic to see that b_1, b_2, and b_3 in the `lm()` results correspond to the difference in the average wage between dummy group 4 and dummy groups 1, 2, and 3 respectively.

We can also choose to a different group than group 4. Here, we can choose to drop dummy group 3 and to keep the intercept. What do the intercept and coefficients represent now?

```
(b_drop3 <- OLS(y,X[ , c(1,2,3,5)]))
```

```
         [,1]
   1106.45128
e1 -259.95648
e2 -139.46363
e4   90.50748
```

Suppose we want to estimate the premium on education, relative to the least educated class. We can then specify the following regression and print the output directly to LaTeX using the `xtable` package[2]:

```
xtable(m2 <- lm(wage ~ 1 + e2 + e3 + e4, data = data))
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	846.4948	17.4837	48.42	0.0000
e2	120.4929	34.8322	3.46	0.0006
e3	259.9565	32.5528	7.99	0.0000
e4	350.4640	42.6787	8.21	0.0000

None that the coefficients, now, indicate the premium over the base level of education for all subsequent levels. Consider, for example, the premium on the `e4` class. The average wage for people in this class, the class with the highest educational attainment levels, is found by:

```
mean(data[e4 == 1, c("wage")])
```

```
[1] 1196.959
```

This is equivalent to adding the coefficient on `e4` to the intercept from the well-specified regression above. Specifically:

```
b <- m2$coefficients
b[["(Intercept)"]] + b[["e4"]]
```

```
[1] 1196.959
```

This equality only holds because there are no other covariates in the regression. If we condition on age, for example, then the simple addition does not yield an average wage. For illustration, consider the previous regression with `age` and squared `age` as cofactors. Note also the manner by which the `lm()` function accepts a nested function to specify squared `age` within the line:

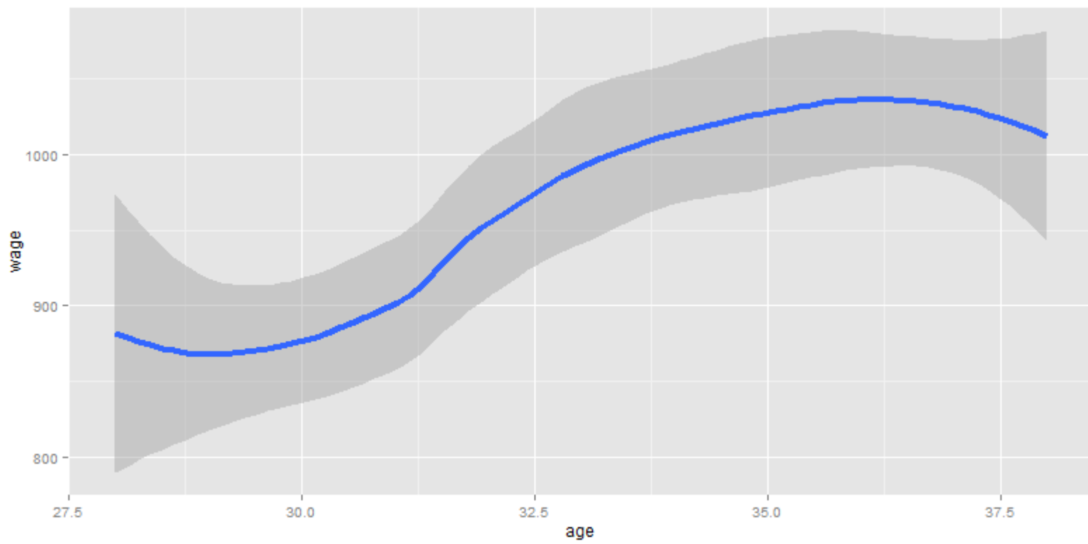
```
coef(summary(lm(wage ~ 1 + e2 + e3 + e4 + age + I(age^2), data = data)))
```

```
                Estimate  Std. Error     t value     Pr(>|t|)
(Intercept) -148.0041478 1660.163718 -0.08915033 9.289817e-01
e2           142.0919504   34.603321  4.10630965 4.374401e-05
e3           276.4462130   32.327306  8.55147690 4.954754e-17
e4           329.3717601   42.427654  7.76313861 2.181854e-14
age           38.4978013  100.467589  0.38318628 7.016693e-01
I(age^2)      -0.2572671    1.508597 -0.17053405 8.646273e-01
```

[2]Note that this is a little superfluous, but it's worth examining the different ways to export tables.

It looks like age has a positive but diminishing effect on wage. This makes sense, maybe, but the coefficients are not significantly different from zero. Why might this be the case? This is where some non-parametric graphing comes in handy.

```
(g <- ggplot(data, aes(x=age, y=wage)) + geom_smooth(method="loess", size=1.5))
```



We use the `ggplot2` package instead of the base `R` plotting facilities. The plots reveal a reasonable relationship between wage and age, but there is a significant amount of variation in wage, relative to the short time frame of age.

```
(g <- g + geom_point())
```