We'll start section with some brief comments on the problem set and a plug for `knitr`. Then, we'll look at the efficiency gains from a specialized case of generalized least squares using simulated data. With remaing time, we'll try out an empirical example to demonstrate the use of categorical dummies. Along the way, we'll apply some more advanced tools of the trade, in particular using `ggplot2` and exporting `R` tables in LaTeX format.

Problem set retrospective

In general, you guys did really well on the problem set. I've put the answer key on bSpace and will be handing your problem sets back today at the end of section. Many students found it useful to use LaTeX and `knitr` (in `RStudio`) to format their assignments. This is great! The basic idea of `knitr` is that you can answer math or non-code questions using text or LaTeX and then "knit" in your `R` code and/or results[1] in the same file; this means that you don't have to maintain separate files for your notes, results, and code. This process is a form of "Literate Programming"[2], and it can be very useful for developing research ideas, collaborating on projects, and, of course, writing problem sets. If you're interested in learning how to use `knitr`, have a look at `ps1_answers.rnw` on bSpace — it's the code Max and I used to generate the answer key.

Efficiency gains from 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`.

Let $x \sim U(0, 2000)$ and $\varepsilon \sim N(0, 4x^2)$. The underlying data generating process in (2.102) is $y_i = \alpha + x_i\beta + \varepsilon$, where $\alpha = 0.5$ and $\beta = 1.5$. The objective is to plot the simulated sampling distribution of the OLS estimator applied to $B = 1000$ draws, each of size $n = 1000$. After creating a slightly more complex OLS function that returns standard errors, we'll create our simulated population.

```
set.seed(42)
OLS <- function(y,X) {
  n <- nrow(X); k <- ncol(X)
  b <- solve(t(X) %*% X) %*% t(X) %*% y; names(b) <- "Estimate"
  e <- y - X %*% b
  s2 <- t(e) %*% e / (n - k)
  XpXinv <- solve(t(X) %*% X)
  se <- sqrt(s2 * diag(XpXinv)); names(se) <- "Std. Error"
  return(data.frame(b,se))
}

pop.n <- 1000000
pop.x <- runif(pop.n, min=0, max=2000)
```

[1] `sweave` is similar — in fact, `knitr` is an update of sorts to `sweave`.
[2] A term coined originally by the computer scientist Donald Knuth

```
pop.eps <- rnorm(pop.n, 0, sqrt((4*pop.x)^2))
pop.y <- 0.5 + pop.x*1.5 + pop.eps

n <- 1000
```

Now we'll pull 1000 observations at random from our population — this is our sample. With these, 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.

```
indices <- sample(1:pop.n,n,replace=F)
x <- pop.x[indices]
y <- pop.y[indices]
X <- cbind(1, x) # add an intercept

b <- OLS(y,X)[ , 1]
print(b[2])
```

```
[1] 1.614584
```

The `sample()` function samples randomly from a vector, here without replacement. We create `x` by using requesting a randomly sampled set of indices from `pop.x`. 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.

```
rnd.beta <- function(i) {
  indices <- sample(1:pop.n,n,replace=F)
  x <- pop.x[indices]
  y <- pop.y[indices]
  X <- cbind(1, x) # add an intercept
  b <- OLS(y,X)[ , 1]
  return(b[2])
}
```

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] 1.776229
[1] 1.376166
```

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 <- 10
beta.vec <- sapply(1:B, rnd.beta)
head(beta.vec)
mean(beta.vec)
```

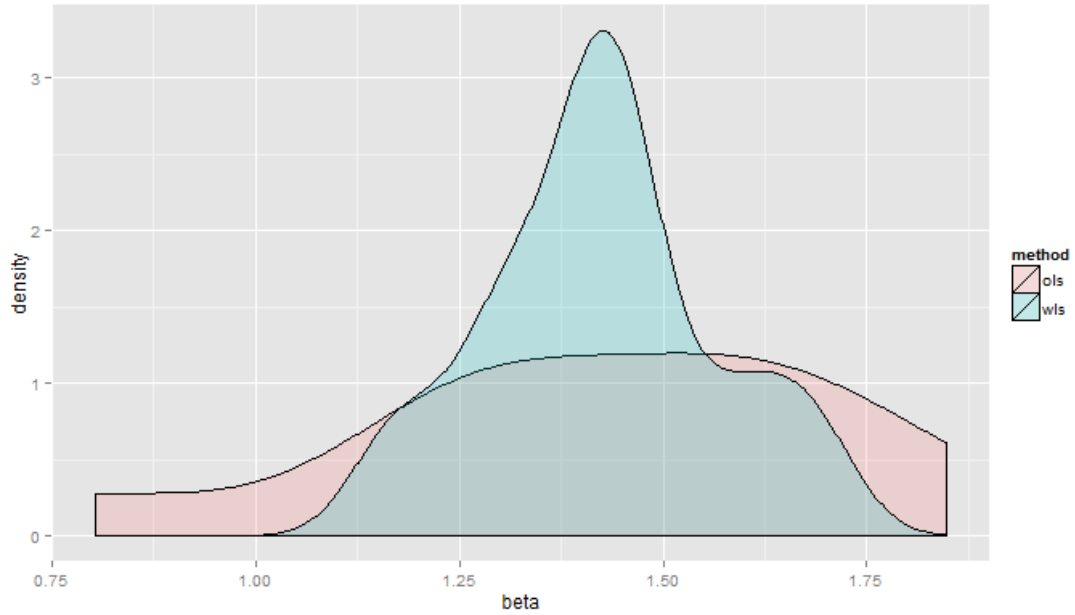
```
[1] 1.649468 1.572407 1.688690 1.374366 1.298503 1.197730
[1] 1.420332
```

All right. Looking good. The average of the simulated sample is much closer to β than almost 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) {
  indices <- sample(1:pop.n,n,replace=F)
  x <- pop.x[indices]
  y <- pop.y[indices]
  C <- diag(1 / sqrt(0.5 * x))
  y.wt <- C %*% y
  X.wt <- C %*% cbind(1, x)
  b.wt <- OLS(y.wt,X.wt)[ , 1]
  return(b.wt[2])
}
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)
```



As in the notes, WLS is clearly the more efficient estimator.

Returns to education example

The purpose of this part o fsection is to estimate the returns to education using `R`. There is nothing causal about the results found here; but the empirical application gives us a chance to explore categorical dummies and to use `ggplot2` package again. First, we load the required libraries.

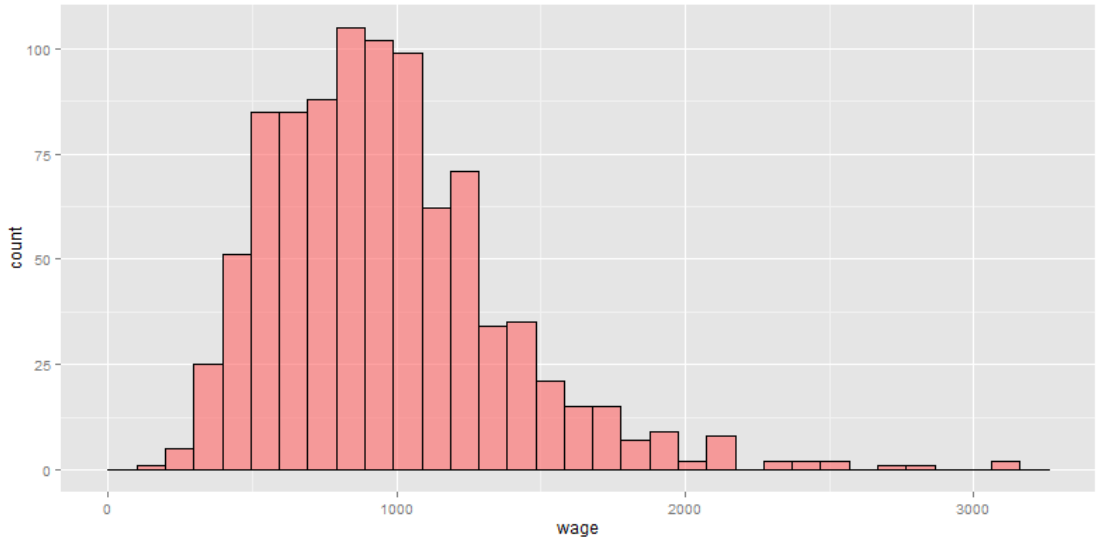
```
library(foreign)
library(xtable)
```

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:

```
ggplot(data, aes(x=wage)) + geom_histogram(colour="black", fill="#FF6666", alpha=0.6)
```



Before we continue, I'll introduce the `%in%` operator, since we'll be using it shortly. The `%in%` operator generates a boolean vector[3], 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")
```

[3] A boolean vector is a vector composed entirely of `TRUE` and/or `FALSE` values.

```
[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`. We can now 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% 17:999, 1, 0)
```

The categorical education variables sum to one, and the `lm()` function will force-drop one of the variables.

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

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1196.96	38.93	30.74	0.00
e1	-350.46	42.68	-8.21	0.00
e2	-229.97	49.23	-4.67	0.00
e3	-90.51	47.64	-1.90	0.06

I snuck in `xtable()` to format the output from `lm()`. `xtable()` outputs `R` data in a beautified LaTeX table format, and it's sufficiently powerful to produce publication-quality tables[4]. It works quite well for output from `lm()`, and we'll see soon that with some tweaks it will work for the output from our own `OLS()` function as well.

Note that the intercept in this regression reflects the mean wage of the `e4` class. The other coefficients reflect the wages of the other three classes relative to `e4`. Interpretation is a common challenge when using 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.

```
X <- cbind(1,e1,e2,e3,e4)
y <- data$wage
b <- OLS(y,X)[ , 1]
```

```
Error in solve.default(t(X) %*% X) (from #3) :
  system is computationally singular: reciprocal condition number = 1.53843e-18
```

Uh oh! What happened? `OLS()`, sadly, 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.

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

	b	se
e1	846.49	17.48
e2	966.99	30.13
e3	1106.45	27.46
e4	1196.96	38.93

We can show 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 omit 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?

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

	b	se
	1106.45	27.46
e1	-259.96	32.55
e2	-139.46	40.76
e4	90.51	47.64

We could similarly compute the differences represented here by calculating the difference in mean wage across groups. For example, comparing the mean wages of groups 3 and 2.

```
mean3 <- mean(data[e3 == 1, c("wage")]); mean2 <- mean(data[e2 == 1, c("wage")]);
cbind(mean3, mean2, mean3 - mean2)
```

```
        mean3    mean2
[1,] 1106.451 966.9877 139.4636
```

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:

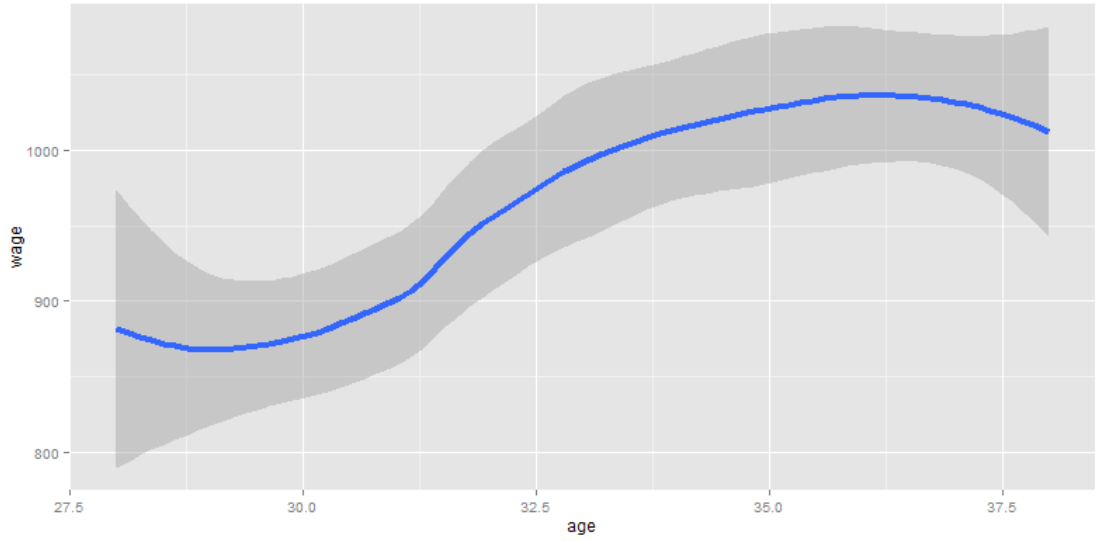
```
wage <- data$wage; age <- data$age; age2 <- age^2; names(age2) <- "age^2"
xtable(OLS(wage,cbind(1,e2,e3,e4,age,age2)))
```

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))
```

[4] `stargazer` is an even more powerful package designed to create LaTeX output

	b	se
	-148.00	1660.16
e2	142.09	34.60
e3	276.45	32.33
e4	329.37	42.43
age	38.50	100.47
age2	-0.26	1.51



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. We can see this by adding the actual data to this graph.

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