In today's section, we'll look at the efficiency gains from a specialized case of generalized least squares using simulated data. To visualize our results, we'll introduce and start using the powerful graphics package, `ggplot2`. Note well that the `ggvis` package will soon supersede `ggplot2` - both have been developed by RStudio's Chief Scientist and fellow Kiwi, Hadley Wickham. `ggvis` introduces interactive plots that work in RStudio or browsers; however, it's still very new, potentially buggy. and doesn't yet integrate nicely into `knitr`. `ggvis` has similar syntax, so should be easy to learn once you know `ggplot2` well.

Efficiency gains from GLS

We will specifically look at the efficiency gains from a special case of GLS, weighted least squares (WLS). We will then create graphs similar to Figures 2.6 and 2.7 in the notes using `ggplot2`.

Let $x \sim U(0, 2000)$ and $\varepsilon \sim N(0, 400 \cdot \frac{1}{100}x_i^2)$, where $\sigma^2 = 400$ and $w_i(x_i) = \frac{1}{100}x_i^2$.[1] The underlying data generating process in (2.102) is $y_i = \alpha + x_i\beta + \varepsilon_i$, 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 $n = 1000$.

```
library(misc212)
```

```
set.seed(2222015)
pop.n <- 1e+06
sigma <- 400
pop.x <- runif(pop.n, min = 0, max = 2000)
pop.w <- (1/100) * pop.x^2
pop.eps <- rnorm(pop.n, mean = 0, sd = sqrt(sigma * pop.w))
pop.y <- 0.5 + pop.x * 1.5 + pop.eps
```

Note that since `rnorm()` takes the standard deviation as its third argument, so we pass it the square root of the variance we want to generate. 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 $\boldsymbol{X}$ is just the x vector bound to a column of ones. We will only examine $\hat{\beta}$ for this section, rather than both parameters.

```
n <- 1000
indices <- sample(1:pop.n, n, replace = FALSE)
x <- pop.x[indices]
y <- pop.y[indices]
X <- cbind(1, x)  # add an intercept

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

```
## [1] 1.45536
```

The `sample()` function samples randomly from a vector, here without replacement. We create `x` by

[1] Careful readers will note that putting some constants into σ^2 and other constants in $w_i(x_i)$ is totally arbitrary, since different choices in this regard will result in weighting matrices `C` that differ only by a constant multiple. Since these are weighting matrices, this will not change our result.

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 = FALSE)
    x <- pop.x[indices]
    y <- pop.y[indices]
    X <- cbind(1, x)  # add an intercept
    b <- OLS(y, X)
    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()
```

```
## [1] 1.73452
```

```
rnd.beta()
```

```
## [1] 1.250775
```

We'll use `sapply` to apply the function to a list of effective indices. Now replicating the process for B draws is straightforward:

```
B <- 1000
beta.vec <- sapply(1:B, rnd.beta)
head(beta.vec)
```

```
## [1] 1.396956 1.390012 1.487341 1.673766 1.657756 1.308225
```

```
mean(beta.vec)
```

```
## [1] 1.505794
```

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. Note that we follow the notes exactly in creating `C.Mat`, which is a diagonal matrix with $\frac{1}{\sqrt{(w_i(x_i))}} = \frac{1}{\sqrt{(\frac{1}{100}x_i^2)}} = \frac{10}{x_i}$ on the diagonals.

```
rnd.wls.beta <- function(i) {
    indices <- sample(1:pop.n, n, replace = FALSE)
    x <- pop.x[indices]
    y <- pop.y[indices]
    w <- pop.w[indices]
    C.Mat <- diag(1/sqrt(w))  # equivalent to: C.Mat <- diag(10 / x)
    y.wt <- C.Mat %*% y
    X.wt <- C.Mat %*% cbind(1, x)
    b.wt <- OLS(y.wt, X.wt)
    return(b.wt[2])
}
```

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

```
## [1] 1.494834 1.384711 1.518708 1.479496 1.514039 1.540588
```

Now, quickly, let's look at how to do WLS using canned packages.

```
indices <- sample(1:pop.n, n, replace = FALSE)
x <- pop.x[indices]
y <- pop.y[indices]
w <- pop.w[indices]
C.Mat <- diag(1/sqrt(w))  # equivalent to: C.Mat <- diag(10 / x)
y.wt <- C.Mat %*% y
X.wt <- C.Mat %*% cbind(1, x)
b.wt <- OLS(y.wt, X.wt)
b.wt[2]
```

```
## [1] 1.530335
```

```
# Now use lm()
tmp <- lm(y ~ x, weights = 1/w)
tmp$coefficients[2]
```

```
##        x
## 1.530335
```

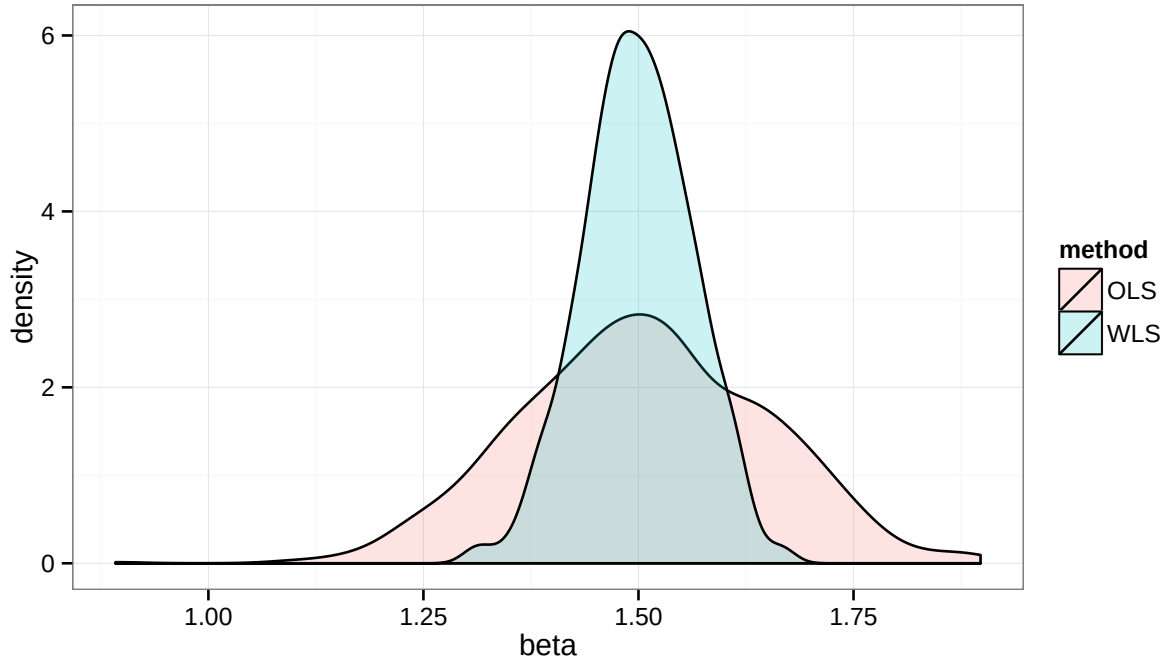
```
# Also can use the more flexible gls() function from the nlme package
library(nlme)
tmp2 <- gls(y ~ x, weights = ~w)
tmp2$coefficients[2]
```

```
##        x
## 1.530335
```

You can see that the weights specification is different for `lm()` and `gls()`. Make sure you know what you're doing if you're going to use these in real life!

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))
graphData <- data.frame(beta = c(beta.vec, wls.beta.vec), method = labels)
myplot <- ggplot(graphData, aes(x = beta, fill = method)) + geom_density(alpha = 0.2) +
    theme_bw()
myplot
```



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