ARE212: Section 05

Dan Hammer

August 21, 2012

This section will briefly walk through two general concepts, one econometric and one `R` oriented. 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). This is the econometric concepts. We will then recreate the graphs from Figures 2.6 and 2.7, roughly, in the notes using `ggplot2` a very popular, very powerful 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 population in (2.102) is given as

$$y_i = \alpha + x_i\beta + \epsilon,$$

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] 1.499929
```

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) {
  # the argument 'i' is not used within the function, but rather to
  # index the function call; useful for =apply= functions
  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 a different random sample for each call:

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

```
 [1] 1.461774
 [1] 1.475344
```

This is convenient for bootstrapping without loops, but rather applying the function to a list of effective indices.[1] Now replicating the process for B draws is straightforward:

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

```
 [1] 1.499225
```

Alright. 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 appropriately centered. 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]
```

[1] This is much more comfortable for me, with a background in functional programming. There is some inherent value, however, in keeping code compact by mapping across indices rather than incrementing an index within a `for` loop; the code is more readable and less prone to typos.

```
  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)
ols <- cbind(beta.vec, 0)
wls <- cbind(wls.beta.vec, 1)
data <- data.frame(rbind(ols, wls))
names(data) <- c("beta", "mt")
data$method <- "ols"
data$method[data$mt==1] <- "wls"
m <- ggplot(data, aes(x=beta, ..density.., fill=method))
m + geom_density(alpha=0.2)
```