Gauss-Markov Models

Today, we'll be doing some gauss markov CLRM - where the least squares estimators are the best linear unbiased estimators for the population mean line.

Our model in this case will be:

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \text{(Simple Linear Regression Model)} \beta_0 = 3 \beta_1 = 6 \sigma^2 = 4$$

Thus, using CLRM, we can expect the sampling distributions to be

$$\hat{\beta_1} \sim N(\beta_1, \frac{\sigma^2}{S_{xx}}) \hat{\beta_0} \sim N(\beta_1, \sigma^2(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}))$$

So, now we need to decide what the x values should be (as one of the conditions of the gaus markov conditions is that the explanatory variable is non stochiastic - we saw the reason why in the previous section).

```
set.seed(42)
sample_size = 100
x = seq(-1,1, length = sample_size)
S_xx = sum((x - mean(x))^2)
```

And we can setup our (underlying) model values:

```
beta_0 = 3
beta_1 = 6
sigma = 2
```

Using the CLRM assumptions, we can estimate the variance of our population coefficient line and intercept to be:

```
variance_of_beta_1_hat = (sigma^2) / S_xx
variance_of_beta_0_hat = (sigma ^2) * (1/sample_size  + (mean(x)^2)/S_xx)

c(variance_of_beta_0_hat, variance_of_beta_1_hat)
```

```
## [1] 0.0400000 0.1176238
```

We're also going to be reusing the least squares method from earlier:

```
least_squares <- function(x_vals, y_vals) {
  beta_1_hat = sum((x_vals - mean(x_vals))*(y_vals - mean(y_vals)))/ sum((x_vals - mean(x_vals))^2)
  beta_0_hat = mean(y_vals) - beta_1_hat * mean(x_vals)
  return(c(beta_0_hat, beta_1_hat))
}
```

And now here's the magic - we're going to generate 10,000 random values of sigma, use clrm and record the values of $\hat{\beta_0}$ and $\hat{\beta_1}$.

```
tests_count = 10000
beta_0_hats = rep(0, tests_count)
beta_1_hats = rep(0, tests_count)

for (i in 1:tests_count) {
  eps = rnorm(sample_size, mean = 0,  sd = sigma)
  y = beta_0 + beta_1 * x + eps
  model =  least_squares(x, y)
```

```
  beta_0_hats[i] = model[1]
  beta_1_hats[i] = model[2]
}
```

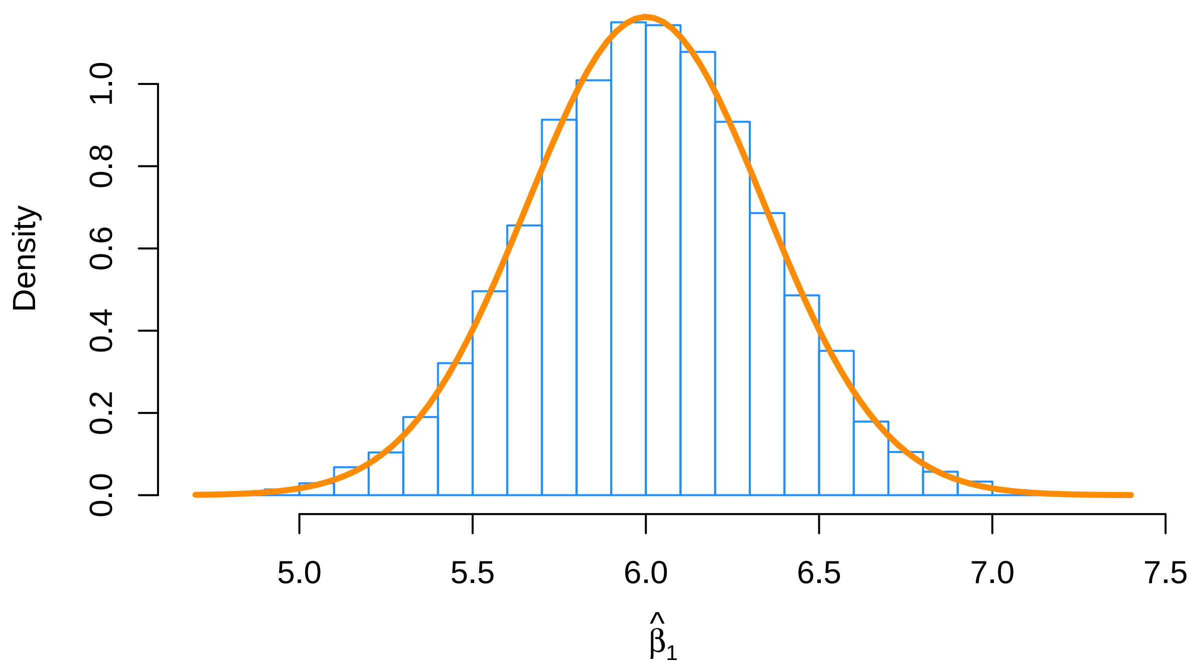
Now, let's check the mean and variance of these bad boys!

```
c(mean(beta_0_hats), var(beta_0_hats), mean(beta_1_hats), var(beta_1_hats))
```

```
## [1] 3.00114748 0.04017924 6.00199802 0.11898997
```

Not too shabby - but even better, let's plot this thing!

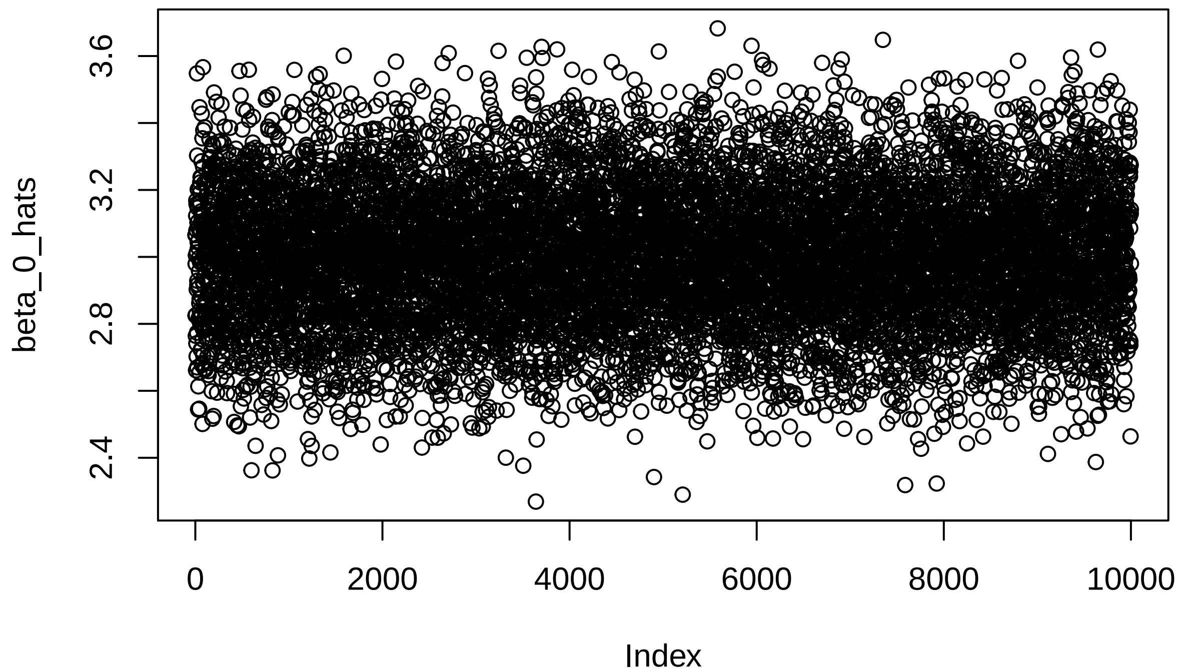
```
hist(beta_1_hats, prob=TRUE, breaks=20, xlab=expression(hat(beta)[1]), main="", border="dodgerblue")
curve(dnorm(x, mean=beta_1, sd=sqrt(variance_of_beta_1_hat)), col="darkorange", add=TRUE, lwd=3)
```



I don't know about you, but that looks pretty normal to me.

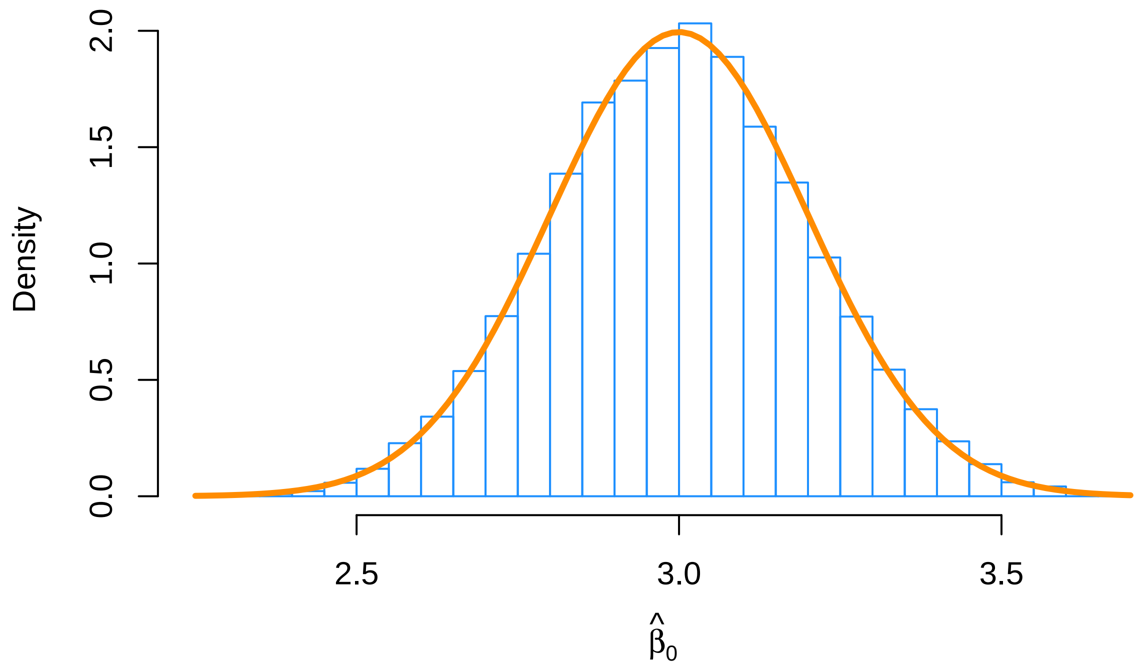
We can see this if we plot the datapoints we obtained:

```
plot(beta_0_hats)
```



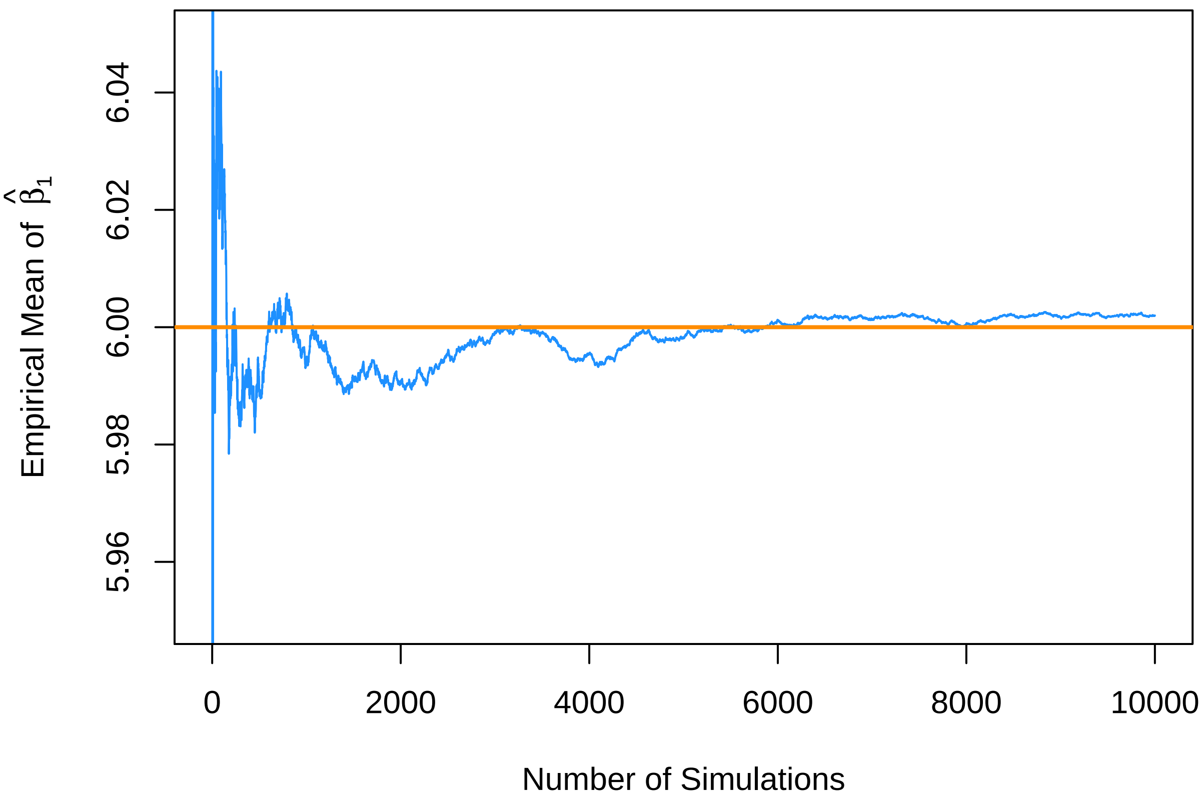
Simmilarly, let's try the same for $\hat{\beta}_0$ values:

```
hist(beta_0_hats, prob=TRUE, breaks=25, xlab=expression(hat(beta)[0]), main="", border="dodgerblue")
curve(dnorm(x, mean=beta_0, sd=sqrt(variance_of_beta_0_hat)), col="darkorange", add=TRUE, lwd=3)
```



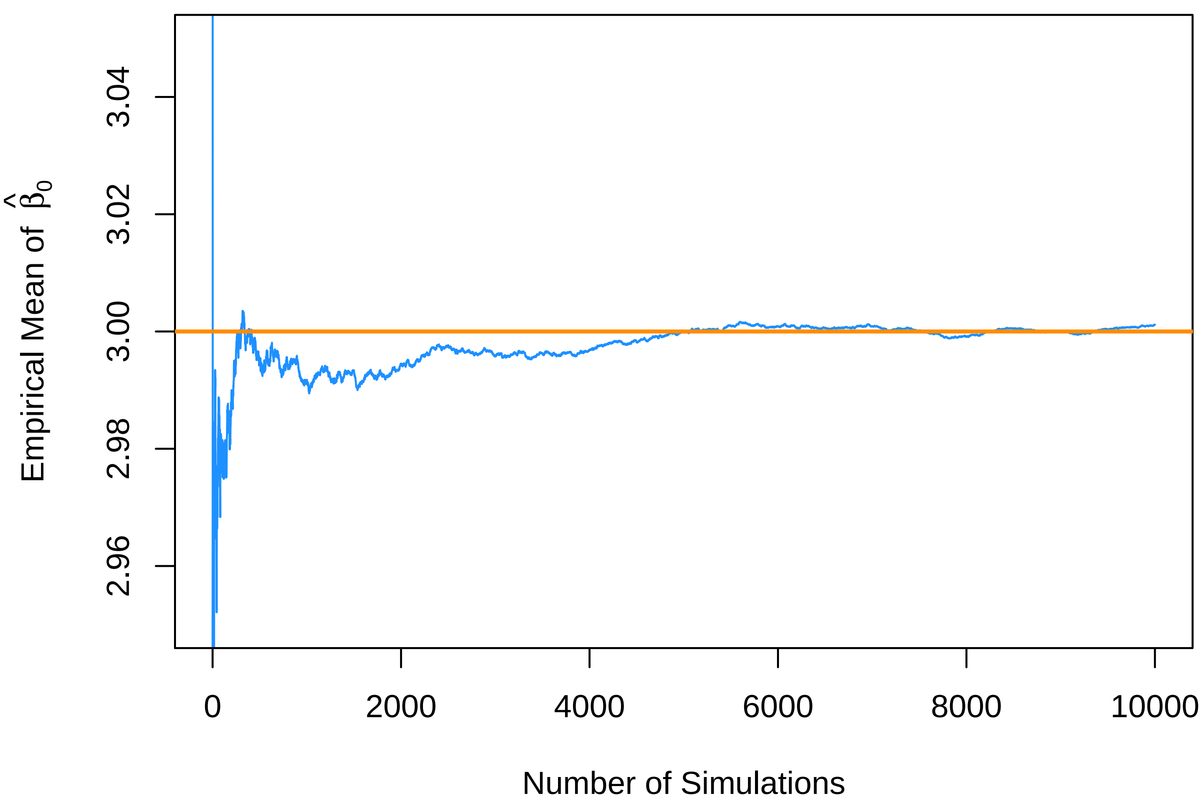
We can really see how the sample mean starts to converge if we plot the cumulative mean against the number of iterations:

```
par(mar = c(5,5,1,1))
plot(cumsum(beta_1_hats)/(1:length(beta_1_hats)), type="l", ylim = c(5.95, 6.05),
     xlab = "Number of Simulations",
     ylab = expression("Empirical Mean of " ~ hat(beta)[1]),
     col = "dodgerblue")
abline(h=beta_1, col="darkorange", lwd=2)
```



Looks like that's converging all right.

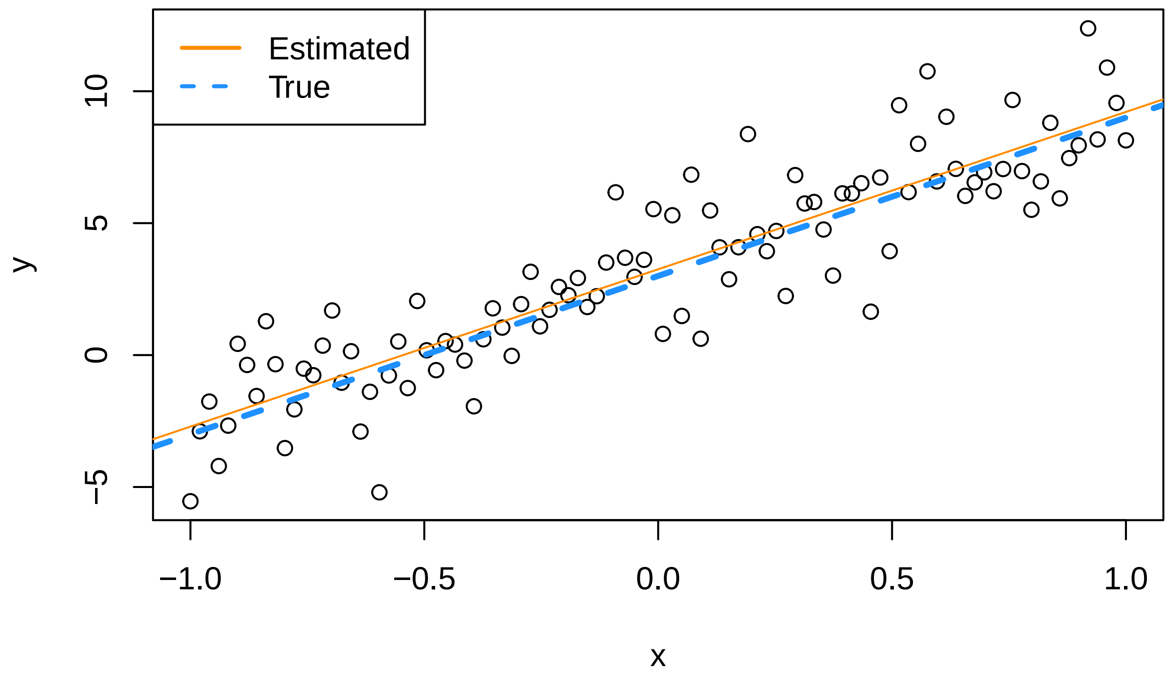
```
par(mar = c(5,5,1,1))
plot(cumsum(beta_0_hats)/(1:length(beta_0_hats)), type="l", ylim = c(2.95, 3.05),
     xlab = "Number of Simulations",
     ylab = expression("Empirical Mean of " ~ hat(beta)[0]),
     col = "dodgerblue")
abline(h=beta_0, col="darkorange", lwd=2)
```



Also converging pretty nicely.

Now, here's some cool thingss - we sometimes want to predict the mean of y given an x value, or a y value given an x value - what? Let's plot the data and see what we mean:

```
eps = rnorm(sample_size, mean = 0,  sd = sigma)
y = beta_0 + beta_1 * x + eps
model = least_squares(x, y)
plot(y ~ x)
abline(model[1], model[2], col="darkorange")
abline(beta_0, beta_1, lwd=3, lty=2, col="dodgerblue")
legend("topleft",c("Estimated", "True"),lty=c(1,2),lwd=2, col=c("darkorange", "dodgerblue"))
```



So now, let's predict the square error s_ϵ^2:

```
S_xx = sum((x-mean(x))^2)
y_hat = model[1] + model[2] * x
se_2 = sum((y_hat - y)^2)
sigma_2_hat = se_2 / (sample_size-2)
sigma_hat = sqrt(sigma_2_hat)
c(sigma_hat, sigma)
```

```
## [1] 1.73055 2.00000
```

And we can the use them to calculate error bands:

- for a mean value:

$$\bar{y_i}|x_i \sim N(\hat{\beta_0} + \hat{\beta_1}x_i, \sigma^2[\frac{1}{n} + \frac{(x-\bar{x})^2}{S_{xx}}])$$

- for a single value:

$$y_i|x_i \sim N(\hat{\beta_0} + \hat{\beta_1}x_i, \sigma^2[1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{S_{xx}}])$$

Let's try it out:

```
eps = rnorm(sample_size, mean = 0,  sd = sigma)
y = beta_0 + beta_1 * x + eps
model = least_squares(x, y)
```

```
upper_mean_values = (model[1] + model[2] * x) + qnorm(0.95) * sqrt((sigma_hat ^ 2) * ((1/sample_size) +
lower_mean_values = (model[1] + model[2] * x) - qnorm(0.95) * sqrt((sigma_hat ^ 2) * ((1/sample_size) +

upper_individual_values = (model[1] + model[2] * x) + qnorm(0.95) * sqrt((sigma_hat ^ 2) * (1 + (1/sampl
lower_individual_values = (model[1] + model[2] * x) - qnorm(0.95) * sqrt((sigma_hat ^ 2) * (1 + (1/sampl
```

```
plot(y ~ x)
abline(model[1], model[2], col="darkorange")
abline(beta_0, beta_1, lwd=3, lty=2, col="dodgerblue")
lines(x, upper_mean_values, col="dodgerblue", lty=3)
lines(x, lower_mean_values, col="dodgerblue", lty=3)


lines(x, upper_individual_values, col="dodgerblue", lty=4)
lines(x, lower_individual_values, col="dodgerblue", lty=4)

legend("topleft",c("Estimated", "True", "Mean Estimate Error Range", "Individual Estimate Error Range")
points(mean(x), mean(y), pch="+", cex=3)
```