Lab 11 (In Class)

Maria Barouti

3/23/2020

Simulation study

```
library("plot3D")
```

```
## Warning: package 'plot3D' was built under R version 3.5.3
```

In particular, we will simulate samples of size `n = 100` from the model

$$Y_i = 5 + -2X_{i1} + 6X_{i2} + \epsilon_i, \qquad i = 1, 2, \ldots, n$$

where $\epsilon_i \sim N(0, \sigma^2 = 16)$. Here we have two predictors, so $p = 3$.

```
set.seed(1337)
n = 100 # sample size
p = 3

beta_0 = 5
beta_1 = -2
beta_2 = 6
sigma  = 4
```

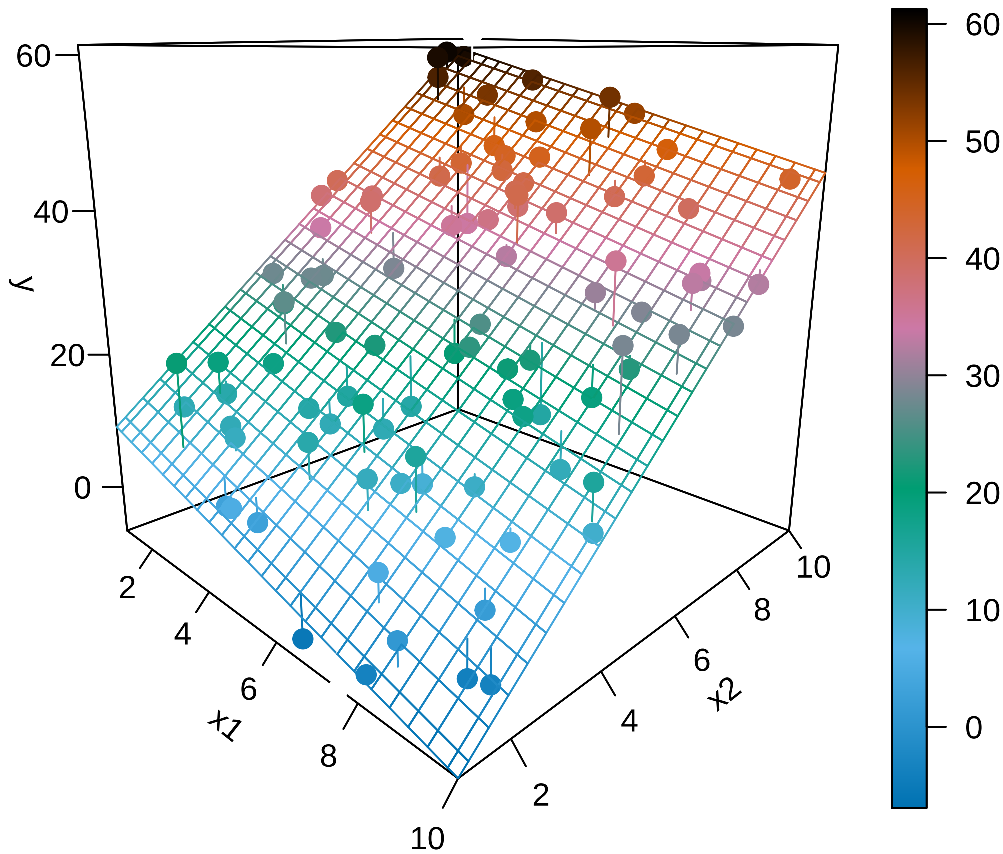
As is the norm with regression, the X values are considered fixed and known quantities, so we will simulate those first, and they remain the same for the rest of the simulation study. Also note we create an `x0` which is all `1`, which we need to create our `X` matrix. If you look at the matrix formulation of regression, this unit vector of all `1`s is a "predictor" that puts the intercept into the model. We also calculate the `C` matrix for later use.

```
x0 = rep(1, n)
x1 = sample(seq(1, 10, length = n))
x2 = sample(seq(1, 10, length = n))
X = cbind(x0, x1, x2)
C = solve(t(X) %*% X)
```

We then simulate the response according the model above. Lastly, we place the two predictors and response into a data frame. Note that we do **not** place `x0` in the data frame. This is a result of `R` adding an intercept by default.

```
eps      = rnorm(n, mean = 0, sd = sigma)
y        = beta_0 + beta_1 * x1 + beta_2 * x2 + eps
sim_data = data.frame(x1, x2, y)
```

Plotting this data and fitting the regression produces the following plot.



We then calculate

$$\mathbf{b} = \left(X^\top X\right)^{-1} X^\top Y.$$

```
(b = C %*% t(X) %*% y)
```

```
##          [,1]
## x0  5.293609
## x1 -1.798593
## x2  5.775081
```

Notice that these values are the same as the coefficients found using `lm()` in `R`.

```
coef(lm(y ~ x1 + x2, data = sim_data))
```

```
## (Intercept)          x1          x2
##    5.293609   -1.798593    5.775081
```

Also, these values are close to what we would expect.

```r
c(beta_0, beta_1, beta_2)
```

```
## [1]  5 -2  6
```

We then calculated the fitted values in order to calculate s_e, which we see is the same as the `sigma` which is returned by `summary()`.

```r
y_hat = X %*% b
(s_e = sqrt(sum((y - y_hat) ^ 2) / (n - p)))
```

```
## [1] 3.976044
```

```r
summary(lm(y ~ x1 + x2, data = sim_data))$sigma
```

```
## [1] 3.976044
```

So far so good. Everything checks out. Now we will finally simulate from this model repeatedly in order to obtain an empirical distribution of b_2.

We expect b_2 to follow a normal distribution,

$$b_2 \sim N\left(\beta_2, \sigma^2 C_{22}\right).$$

In this case,

$$b_2 \sim N\left(6, \sigma^2 = 16 \times 0.0014534 = 0.0232549\right).$$

Note that C_{22} corresponds to the element in the **third** row and **third** column since β_2 is the **third** parameter in the model and because `R` is indexed starting at `1`. However, we index the C matrix starting at `0` to match the diagonal elements to the corresponding β_j.

```r
C[3, 3]
```

```
## [1] 0.00147774
```

```r
sigma ^ 2 * C[3, 3]
```

```
## [1] 0.02364383
```

We now perform the simulation a large number of times. Each time, we update the `y` variable in the data frame, leaving the `x` variables the same. We then fit a model, and store b_2.

```r
num_sims = 10000
beta_hat_2 = rep(0, num_sims)
for(i in 1:num_sims) {
  eps           = rnorm(n, mean = 0 , sd = sigma)
  sim_data$y    = beta_0 * x0 + beta_1 * x1 + beta_2 * x2 + eps
  fit           = lm(y ~ x1 + x2, data = sim_data)
  beta_hat_2[i] = coef(fit)[3]
}
```

We then see that the mean of the simulated values is close to the true value of β_2.

```r
mean(beta_hat_2)
```

```
## [1] 5.99871
```

```r
beta_2
```

```
## [1] 6
```

We also see that the variance of the simulated values is close to the true variance of b_2.

$$\text{Var}[\hat{\beta}_2] = \sigma^2 \cdot C_{22} = 16 \times 0.0014534 = 0.0232549$$

```
var(beta_hat_2)
```

```
## [1] 0.02360853
```

```
sigma ^ 2 * C[2 + 1, 2 + 1]
```

```
## [1] 0.02364383
```

The standard deviations found from the simulated data and the parent population are also very close.

```
sd(beta_hat_2)
```

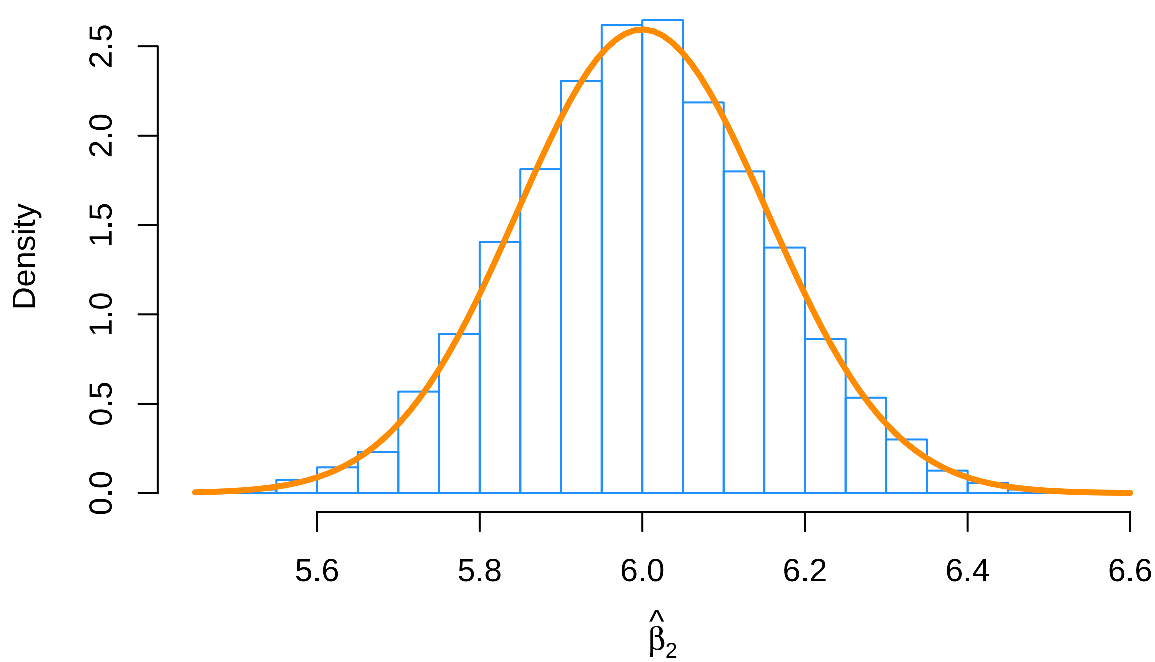
```
## [1] 0.1536507
```

```
sqrt(sigma ^ 2 * C[2 + 1, 2 + 1])
```

```
## [1] 0.1537655
```

Lastly, we plot a histogram of the *simulated values*, and overlay the *true distribution*.

```
hist(beta_hat_2, prob = TRUE, breaks = 20,
     xlab = expression(hat(beta)[2]), main = "", border = "dodgerblue")
curve(dnorm(x, mean = beta_2, sd = sqrt(sigma ^ 2 * C[2 + 1, 2 + 1])),
      col = "darkorange", add = TRUE, lwd = 3)
```



This looks good! The simulation-based histogram appears to be Normal with mean 6 and spread of about 0.15 as you measure from center to inflection point. That matches really well with the sampling distribution of b_2.