B_0 vs SSE

2017-02-02

Problem

I asked you to select several values for the intercept (b_0) and obtain the value of the SSE using your function. Then you were to plot the SSE values versus the b_0 values. In this document I will show you how I did this.

First enter in the data and load up any libraries we might need,

```r
# Enter data into vectors
x = c(4, 0, 3, 4, 7, 0, 0, 3, 0, 2)
y = c(53, 56, 37, 55, 50, 36, 22, 75, 37, 42)

# Load libraries
library(tidyverse)
```

The `tidyverse` library is a shortcut for loading `dplyr`, `ggplot2`, and several other useful libraries in one statement. Now we read in our function.

```r
sse = function(b0, b1){
  sse = sum( (y - b0 - b1*x)^2 )
  return(sse)
}
```

Note that I did not include the arguments `x` and `y` in this function as inputs, even though they are included as part of the function's computations. So long as `x` and `y` are defined in our global envioronment, the function will pull them in as needed nd we don't need them in the arguments for the function. This will make things easier later on.

Now we need to feed in several differet values of b_0 and record the SSE for each of them. We can do this by writing down the b_0 and SSE values in Excel (or directly into a data frame.) For example, here is how you might do this for three values of b_0. First obtain the SSE values for the three b_0 values of 0, 1, and 2.

```r
sse(b0 = 0, b1 = 0)
```

```
## [1] 23377
```

```r
sse(b0 = 1, b1 = 0)
```

```
## [1] 22461
```

```r
sse(b0 = 2, b1 = 0)
```

```
## [1] 21565
```

Then, put the b_0 and SSE values in a data frame

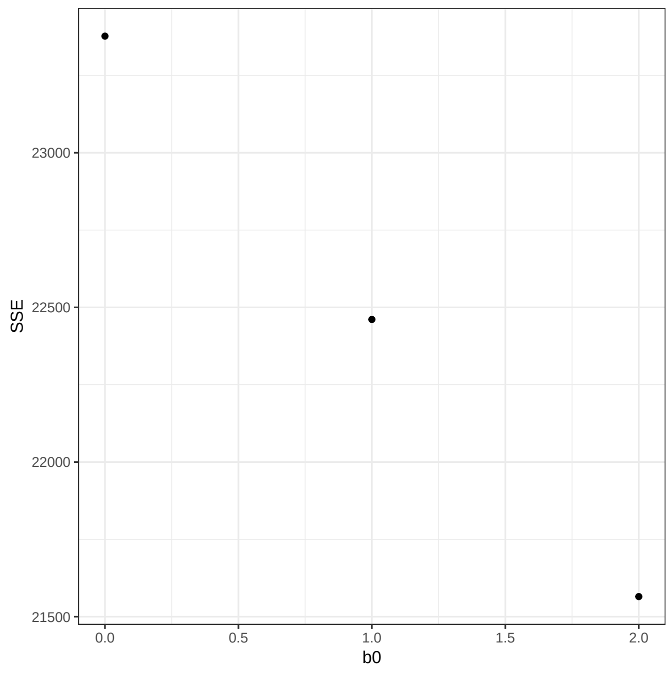
```r
plotData = data.frame(
  b0 = c(0, 1, 2),
  SSE = c(23377, 22461, 21565)
)

# Examine plotData
plotData
```

```
##   b0   SSE
## 1  0 23377
## 2  1 22461
## 3  2 21565
```

Finally, we can plot these data.

```r
ggplot(data = plotData, aes(x = b0, y = SSE)) +
  geom_point() +
  theme_bw()
```



This is the general process that we need to follow. Looking at the plot, it seems that we need to explore higher values of b_0, since the SSE seems to be getting smaller. To do this, however, would require using the function on our new values for b_0, creating a new dataframe with the new b_0 and SSE, and finally new plot. And, we would probably find we need to do it again and again until we settle on the smallest SSE. Ugh.

Let's automate this process a bit.

Automating the Computation of Several b_0 values

There are several ways to do this, but, I am going to set up a new data frame that has a column called `b0` (give it the same name as the argument in your function).

```
plotData = data.frame(
  b0 = seq(from = 0, to = 75, by = 1)
)

# Examine data
head(plotData)
```

```
##   b0
## 1  0
## 2  1
## 3  2
## 4  3
## 5  4
## 6  5
```

Now we can use our function ...almost. Note that if we try to feed our function the `b0` column it will fail to give us what we need, which is 51 different SSE values. (It might give an error or compute a single value, both of which are not what we want). We need to tell R to apply the function to each row value of `b0`. To do this we can pipe to the `rowwise()` function, and then mutate a new column called SSE that computes the SSE value.

```
plotData %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) )
```

```
## Source: local data frame [76 x 2]
## Groups: <by row>
##
## # A tibble: 76 × 2
##        b0   SSE
##     <dbl> <dbl>
## 1       0 23377
## 2       1 22461
## 3       2 21565
## 4       3 20689
## 5       4 19833
## 6       5 18997
## 7       6 18181
## 8       7 17385
## 9       8 16609
## 10      9 15853
## # ... with 66 more rows
```

Because we may need to change the sequence values of b_0, I am going to re-write this syntax so that rather than writing to an object called `plotData` that the `data.frmae()` just pipes directly to the `rowwise()` function.

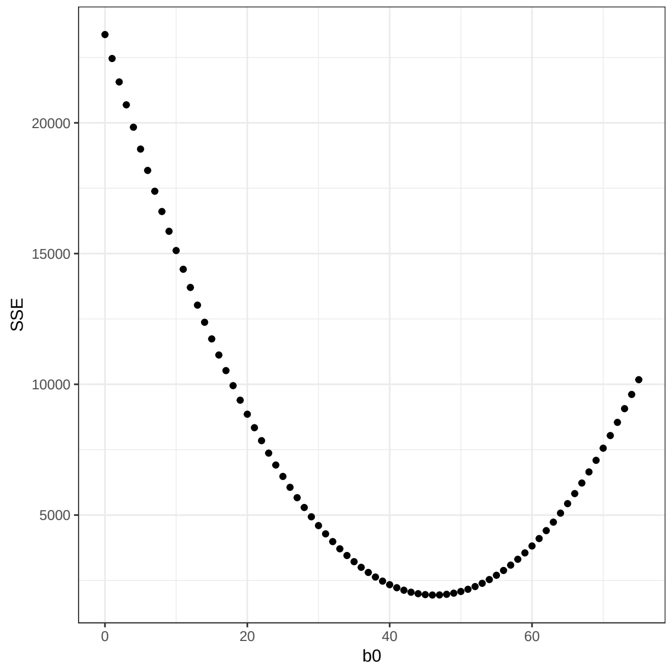
```
data.frame(
  b0 = seq(from = 0, to = 75, by = 1)
) %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) )
```

```
## Source: local data frame [76 x 2]
## Groups: <by row>
##
## # A tibble: 76 × 2
##       b0   SSE
##    <dbl> <dbl>
## 1      0 23377
## 2      1 22461
## 3      2 21565
## 4      3 20689
## 5      4 19833
## 6      5 18997
## 7      6 18181
## 8      7 17385
## 9      8 16609
## 10     9 15853
## # ... with 66 more rows
```

Exact same result, but we just don't save it as an object. This allows us to more easily change values, and also does not pollute our workspace environment with temporary objects.

Now we can pipe this directly into the `ggplot()` function. Rather than assigning computations to objects and then using those objects, I use piping from the very beginning so that there is not a lot of interim objects created in the environment. We use `data = .` to tell ggplot that the dat it should use is the stuff we pipe into it.

```
data.frame(
  b0 = seq(from = 0, to = 75, by = 1)
) %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) ) %>%
  ggplot(data = ., aes(x = b0, y = SSE)) +
    geom_point() +
    theme_bw()
```



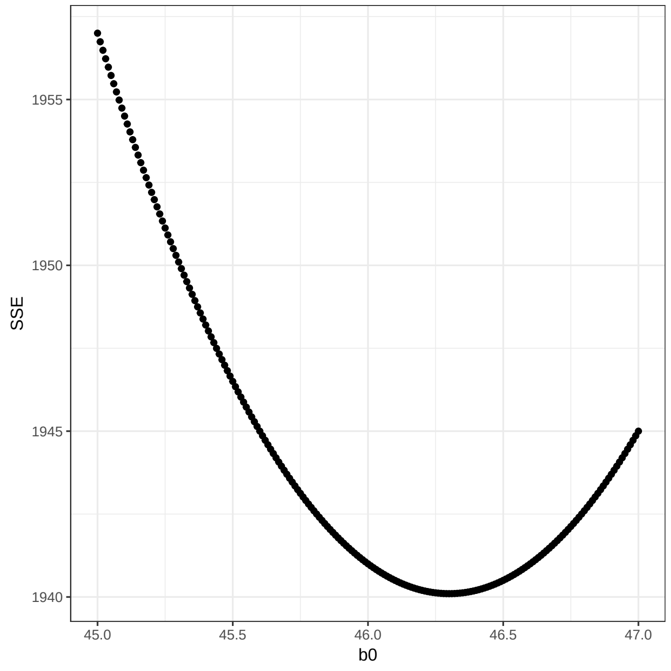
Using this plot, we can pretty clearly see that the b_0 that produces the SMALLEST (least) SSE is around 46 or 47. We can pipe our results into the `arrange()` function, rather than plotting them, to determine the smallest value for SSE.

```
data.frame(
  b0 = seq(from = 0, to = 75, by = 1)
) %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) ) %>%
  arrange(SSE)
```

```
## # A tibble: 76 × 2
##       b0   SSE
##    <dbl> <dbl>
## 1     46  1941
## 2     47  1945
## 3     45  1957
## 4     48  1969
## 5     44  1993
## 6     49  2013
## 7     43  2049
## 8     50  2077
## 9     42  2125
## 10    51  2161
## # ... with 66 more rows
```

Here we find that it is 46 that gives the smallest SSE. If we need this value to be more specific, we can zoom in on the search space by changing the values we give to sequence. Here we search from 45 to 47 by 0.01 increments. I arrange the output by SSE and plot the results.

```r
data.frame(
  b0 = seq(from = 45, to = 47, by = 0.01)
) %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) ) %>%
  ggplot(data = ., aes(x = b0, y = SSE)) +
    geom_point() +
    theme_bw()
```



```r
# Display the results arranged in order
data.frame(
  b0 = seq(from = 45, to = 47, by = 0.01)
) %>%
  rowwise() %>%
  mutate( SSE = sse(b0 = b0, b1 = 0) ) %>%
  arrange(SSE)
```

```
## # A tibble: 201 × 2
##       b0      SSE
##    <dbl>    <dbl>
## 1  46.30 1940.100
## 2  46.31 1940.101
## 3  46.29 1940.101
## 4  46.28 1940.104
## 5  46.32 1940.104
## 6  46.27 1940.109
## 7  46.33 1940.109
## 8  46.26 1940.116
## 9  46.34 1940.116
## 10 46.25 1940.125
## # ... with 191 more rows
```

We can continue to zoom in on the search space until we find the specificity on b_0 that we need.

Optimal Solution

Of course the smallest value for the SSE is the least squares solution. This is what we get when we use the `lm()` function, fitting an intercept-only model.

```
summary( lm( y ~ 1) )
```

```
##
## Call:
## lm(formula = y ~ 1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -24.3   -9.3   -0.3    8.2   28.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   46.300      4.643   9.972 3.66e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.68 on 9 degrees of freedom
```

This gives the same value for b_0 as we got using our grid search. What about the SSE? From our grid search we found that the SSE associated with a b_0 of 46.30 was 1940.1. We can get the SSE associated with the fitted regression by using the `anova()` function.

```
anova( lm( y ~ 1) )
```

```
## Analysis of Variance Table
##
## Response: y
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  9 1940.1  215.57
```

Looks good.

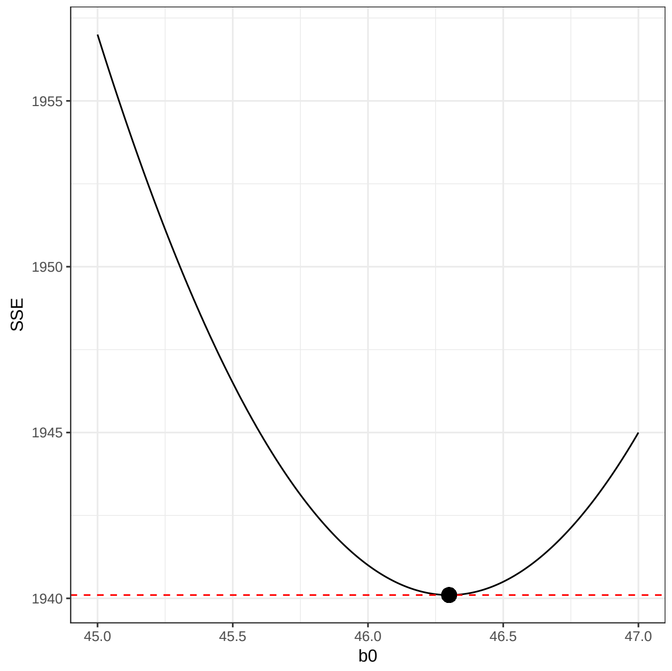
Optional Section: Calculus Solution

Grid search is a reasonable way to find the minimum SSE, but it can be computationally expensive since it essentially uses trial-and-error. A quicker, albeit, more mathematical method of determining the least squares solution is to use calculus. Go back and look at the plot of SSE vs. b_0. It is a smooth function, namely a quadratic. This should not be that surprising if we look at the mthmeatical relationship between SSE and b_0. (Here we assume that $b_1 = 0$, so that part drops out of the formula.)

$$\text{SSE} = \sum \left[y_i - b_0\right]^2$$

SSE is, mathematically, a quadratic function of b_0.

Now consider the tangent line that intersects the smooth quadratic function at the lowest value of SSE. That line would be flat and parallel to the x-axis; its slope would be zero. The plot below illustrates this. The point of intersection between the quadratic function (black, solid line) and the tangent line at the smallest SSE value (red, dashed line) is also shown.



In calculus, the slope of the tangent line is equivalent to the derivative of the function evaluated at a particular point. Thus, to find the minimum SSE, we find the derivative of the quadratic function, set the derivative equal to zero and solve for b_0.

Find the Derivative

Writing out the formula for SSE to expand the sum term, we have,

$$[y_1 - b_0]^2 + [y_2 - b_0]^2 + [y_3 - b_0]^2 + \ldots + [y_{10} - b_0]^2$$

To find the derivative w.r.t. b_0, we need to use the chain rule.

$$2\,[y_1 - b_0]\,(-1) + 2\,[y_2 - b_0]\,(-1) + 2\,[y_3 - b_0]\,(-1) + \cdots + 2\,[y_{10} - b_0]\,(-1)$$

We can simplify this a bit,

$$\begin{aligned}
&-2\,[y_1 - b_0] - 2\,[y_2 - b_0] - 2\,[y_3 - b_0] + \cdots - 2\,[y_{10} - b_0] \\
&-2\Big[y_1 - b_0 + y_2 - b_0 + y_3 - b_0 + \ldots + y_{10} - b_0\Big] \\
&-2\Big[\sum y_i - n(b_0)\Big]
\end{aligned}$$

Set Derivative Equal to 0 and Solve for b_0

Now we can set the derivative equal to zero and solve it for the parameter of interest, namely b_0.

$$\begin{aligned}
0 &= -2\left[\sum y_i - n(b_0)\right] \\
0 &= \sum y_i - n(b_0) \\
-\sum y_i &= -n(b_0) \\
b_0 &= \frac{\sum y_i}{n}
\end{aligned}$$

Thus the optimum solution (the one that produces the minimum SSE) is $b_0 = \frac{\sum y_i}{n}$. This is the marginal mean of Y.