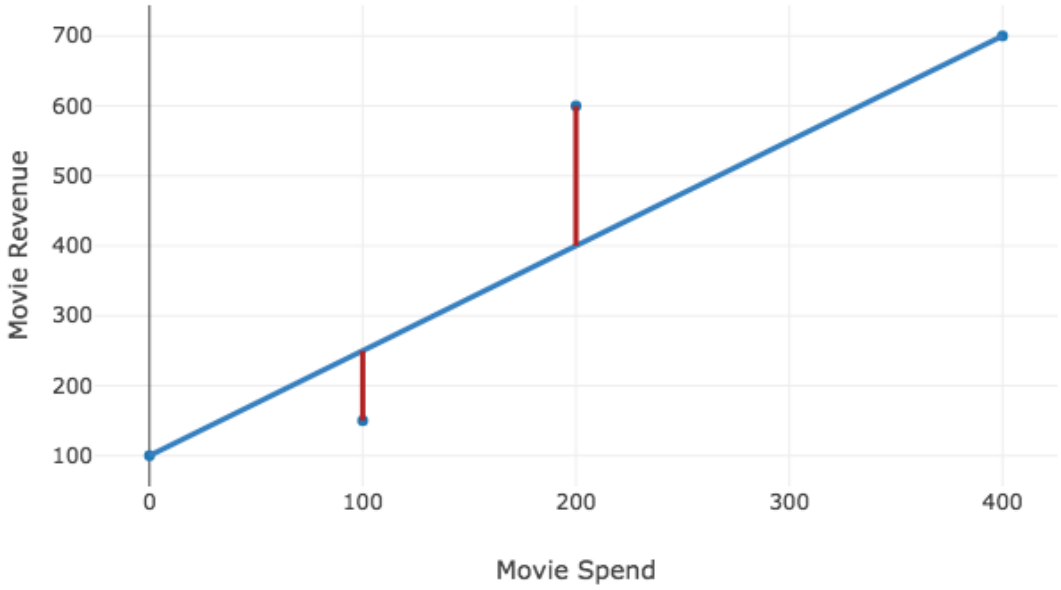
Introduction: Just a bit better

In the last section, we took a first look at the process for improving regression lines. We began with some data then used a simple regression line in the form $\hat{y} = mx + b$ to predict an output, given an input. Finally, we measured the accuracy of our regression line by calculating the differences between the outputs predicted by the regression line and the actual values.

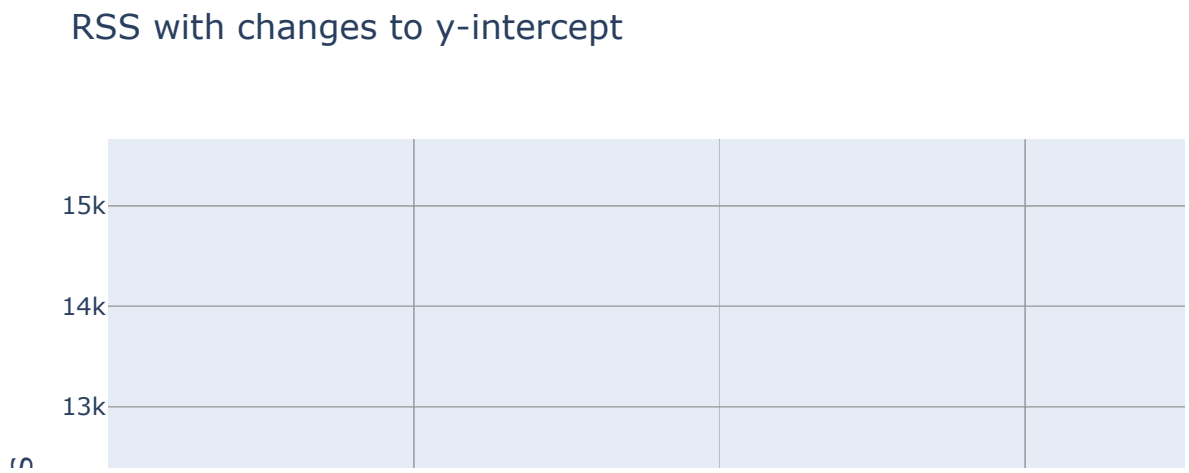


We quantify the accuracy of the regression line by squaring all of the errors (to eliminate negative values) and adding these squares together to get our residual sum of squares (RSS). Armed with a number that describes the line's accuracy (or goodness of fit), we iteratively try new regression lines by adjusting our y-intercept value, b, or slope value, m, and then comparing these RSS values. By finding the values m and b that minimize the RSS, we can find our "best fit line".

In our cost function below, you can see the sequential values of b and the related RSS values (given a constant value m).

In [1]:
```
import plotly
from plotly.offline import init_notebook_mode, iplot
from graph import m_b_trace, trace_values, plot, build_layout
init_notebook_mode(connected=True)
b_values = list(range(70, 150, 10))
rss = [10852, 9690, 9128, 9166, 9804, 11042, 12880, 15318]

layout = build_layout(options = {'title': 'RSS with changes to y-intercept'
cost_curve_trace = trace_values(b_values, rss, mode="lines")
plot([cost_curve_trace], layout)
```

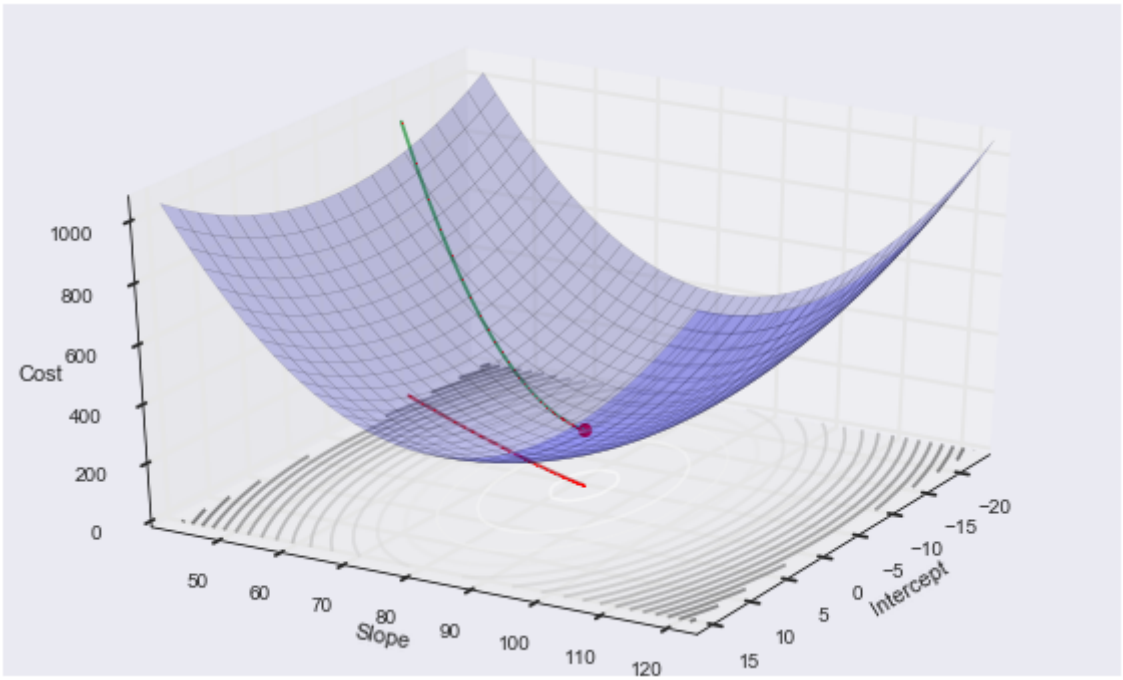


The bottom of the blue curve displays the b value that produces the lowest RSS.

Things are not so simple

At this point, our problem of finding the minimum RSS may seem simple. For example, why not simply try **all** of the different values for a y-intercept, and find the value where RSS is the lowest?

So far, we have held one variable constant in order to experiment with the other. We need an approach that will continue to work as we change both of the variables in our regression line. Altering the second variable makes things far more complicated. Here is a quick look at our cost curve if we can change both our y-intercept and slope value:



As we can see, exploring both variables, the slope and the y-intercept, requires plotting the second variable along the horizontal axis and turning our graph into a three-dimensional representation. And in the future we'll be able to change more than just that.

Furthermore, because we need to explore multiple variables in our regression lines, we are forced to rule out some approaches that are more computationally expensive, or simply not possible.

- We **cannot** simply use the derivative (more on that later) to find the minimum. Using that approach will be impossible in many scenarios as our regression lines become more complicated.
- We **cannot** alter all of the variables of our regression line across all points and calculate the result. It will take too much time, as we have more variables to alter.

However, we are on the right track by altering our regression line and calculating the resulting RSS values.

Remember in the last lesson, we evaluated our regression line by changing our y-intercept by 10 to determine whether it produced a higher or lower RSS.

b	residual sum of squared
140	24131
130	21497
120	19864
110	19230

b	residual sum of squared
100	19597
90	20963
80	23330
70	26696

Rather than arbitrarily changing our variables, as we have done by decrementing the y-intercept by 10 in the example above, we need to move carefully down the cost curve to be certain that our changes are reducing the RSS.

Our approach

We don't want to adjust the y-intercept value or another variable and hope that the RSS decreased. Doing so is like trying to fly plane just by sitting down and pressing buttons.

We want an approach that lets us be certain that we're moving in the right direction with every change. Also, we want to know how much of a **change** to make to minimize RSS.

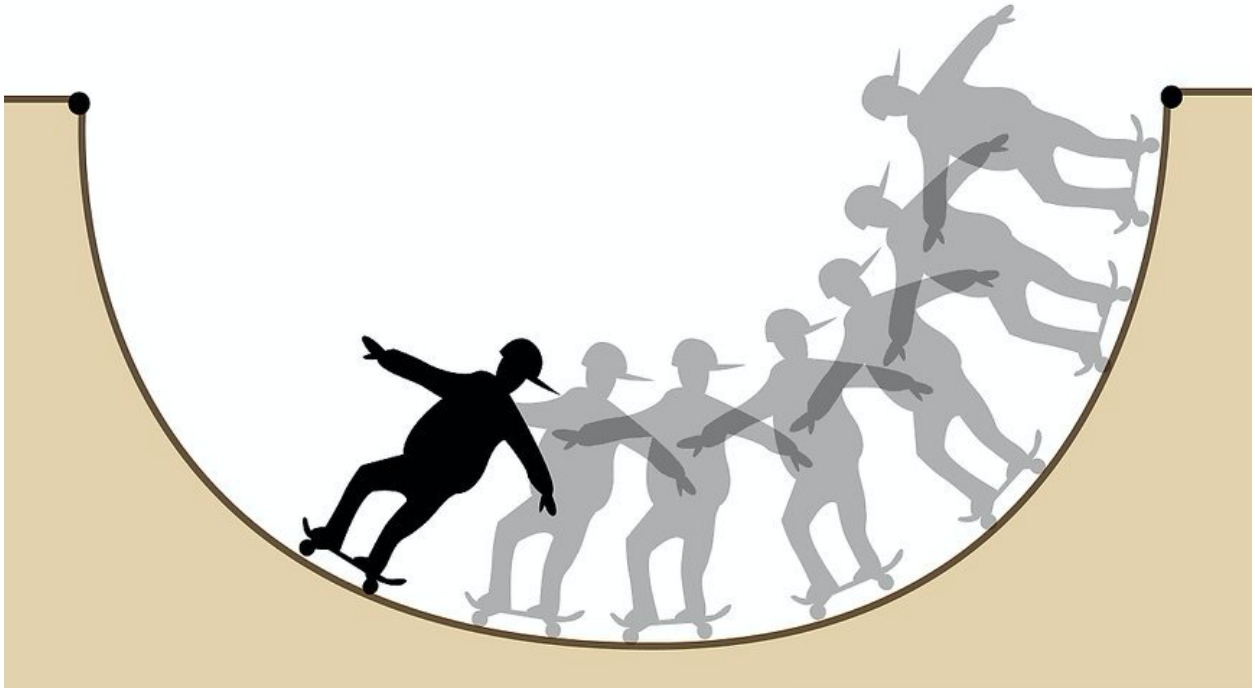
> Let's call each of these changes a **step**, and the size of the change our **step size**.

Our new task is to find step sizes that bring us to the best RSS quickly without overshooting the mark.

The slope of the cost curve tells us our step size

Believe it or not, we can determine the proper step size just by looking at the slope of our cost function.

Imagine yourself standing on our cost curve like a skateboarder at the top of a halfpipe. Even with your eyes closed, you could tell simply *by the way you tilted* whether to move forwards or backwards to approach the bottom of the cost curve.

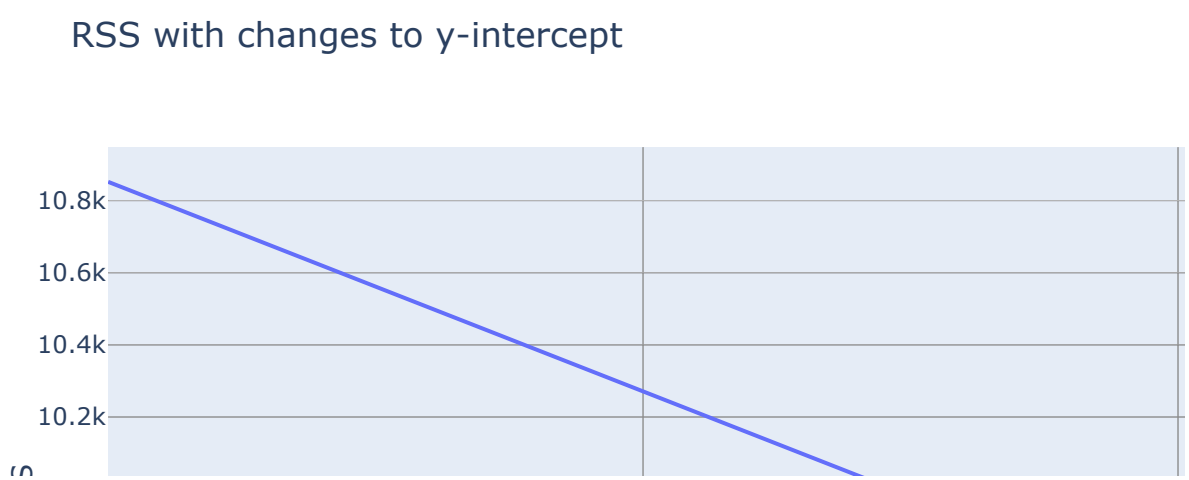


- If the slope tilts downwards, then we should walk forward to approach the minimum.
- And if the slope tilts upwards, then we should point walk backwards to approach the minimum.
- The steeper the tilt, the further away we are from our cost curve's minimum, so we should take a larger step.

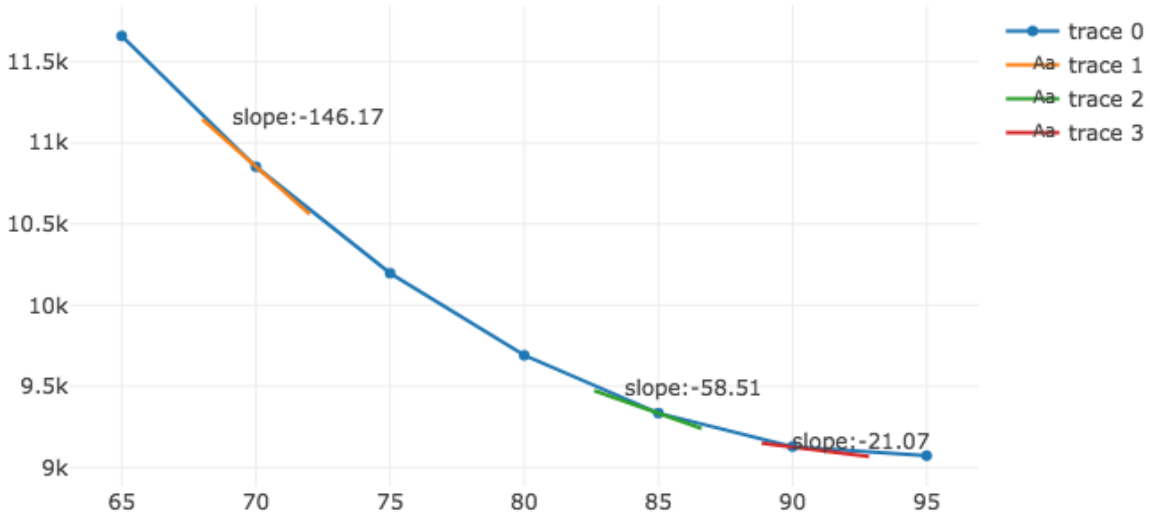
So by looking to the tilt of a cost curve at a given point, we can discover the direction of our next step and how large of step to take. The beauty of this, is that as our regression lines become more complicated, we need not plot all of the values of our regression line. We can see the next variation of the regression line to study simply by looking at the slope of the cost curve.

To demonstrate this, let's zoom in on our cost function and look at just one part of it. Looking at our zoomed in cost function below, we can get a sense of the direction and magnitude of change required to alter our y-intercept in the next iteration.

In [2]:
```
import plotly
from plotly.offline import init_notebook_mode, iplot
from graph import m_b_trace, trace_values, plot, build_layout
init_notebook_mode(connected=True)
layout = build_layout(options = {'title': 'RSS with changes to y-intercept'
b_values = list(range(70, 150, 10)[:3])
rss = [10852, 9690, 9128, 9166, 9804, 11042, 12880, 15318][:3]
cost_curve_trace = trace_values(b_values, rss, mode="lines")
plot([cost_curve_trace], layout)
```



Stepping according to the slope



We can follow our technique with more precision by adding some numbers to our slope. The slope of the curve at any given point is equal to the slope of the tangent line at that point. By tangent line, we mean the line that just barely touches the curve at that point. In the above graph, the orange, green, and red lines are tangent to our cost curve at the points where b equals 70, 85, and 90, respectively. The slopes of our tangent lines, and therefore the slopes of the cost curves at those points, are labeled above.

Let's see how this works.

We use the following procedure to find the ideal b:

1. Randomly choose a value of b, and
2. Update b with the formula $b = (-.1) * slope_{b=i} + b_i$.

The formula above tells us which b value to look at next. We start by choosing a random b value that we can plug into our formula. We take the slope of the curve at that b value and multiply it by $-.1$ then add it to our original b value to produce our next b value.

As we can surmise, the larger the slope, the larger the resulting step to the next b value.

Here's an example. We randomly choose a b value of 70. Then:

- $b_{t=0} = 70$
- $b_{t=1} = (-.1) * -146.17 + 70 = 14.61 + 70 = 84.61$
- $b_{t=2} = (-.1) * -58.51 + 85 = 5.851 + 85 = 90.85$
- $b_{t=3} = (-.1) * -21.07 + 90.85 = 90.851 + 2.11$

Notice that we don't update our values of b by just adding or subtracting the slope at that point. The reason we multiply the slope by a fraction like .1 is so that we avoid the risk of overshooting the minimum. This fraction is called the **learning rate**. Here, the fraction is negative because we always want to move in the opposite direction of the slope. When the slope of the cost curve points downwards, we want to move to a higher y-intercept. Conversely, when we are on the right side of the curve and the slope is rising, we want to move backwards to a lower y-intercept.

This technique is pretty magical. By looking at the tangent line at each point, we no longer are changing our b value and just hoping that it has the correct impact on our RSS. This is because, for one, the slope of the tangent line points us in the right direction. And as you can see above, our technique properly adjusts the amount to change the b value by without even knowing the ideal b value. When our b was far away from the ideal b value, our formula increased b by over 14. By the third step, we were updating our b value by only 2 because we were closer to the ideal slope for minimizing the RSS.

Summary

We started this section with saying that we wanted a technique to find a b value that would minimize our RSS, given a value of m. We did not want to simply try all of the values of b as doing so would be inefficient. Instead, we went with the approach of gradient descent, where we try variations of regression lines iteratively changing our b variable and assessing our RSS to see if we are making progress.

In this lesson, we focused in on how to know which direction to alter a given variable, m or b, as well as a technique for determining the size of the change to apply to one of our variables. We used the line tangent to our cost curve at a given point to indicate the direction and size of the update to b. The further away, the steeper the curve and thus the larger the step we would want to take. Appropriately, our tangent line slope would have us take a larger step. And the closer we are to the ideal b value, the flatter the tangent line to the curve, and the smaller a step we would take.