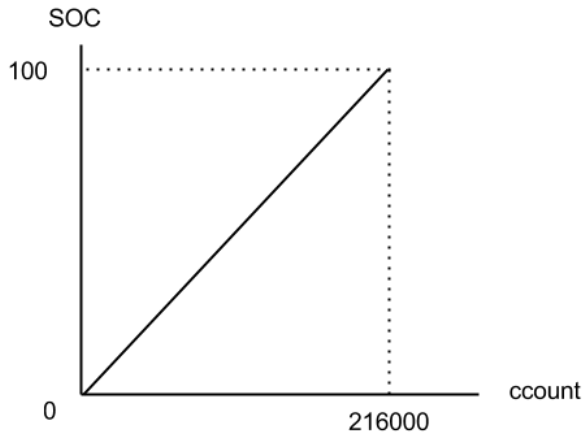
State Of Charge Algorithm

The battery pack needs to display state of charge of the cells and the battery pack as a whole. Assuming that the cells are balanced, the SOC of the battery pack is the same as SOC of each individual cell. The SOC algorithm implemented uses a coulomb counting and linear regression to determine the current state of charge.

Coulomb Counting

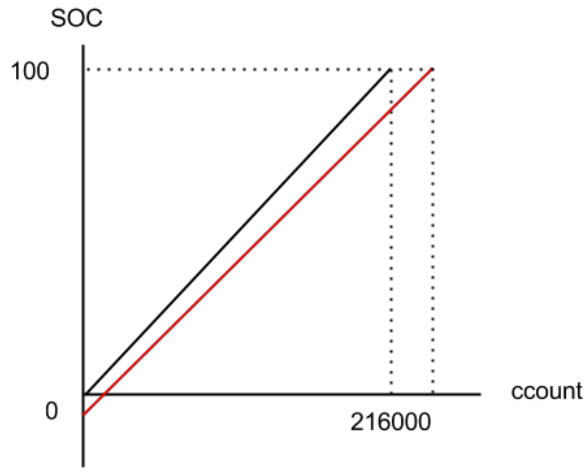
The program uses coulomb counting to determine how much energy (in Amp-second) the battery pack has. Coulomb counting is done by integrating the current over time.The battery cell specification states that it has 60Ah so we can estimate 216000 (Amp-sec) for coulomb counting.Therefore, 0% SOC will mean the battery has 0 coluomb count (ccount) and 100% SOC will mean ccount of 216000.

Linear Regression



As shown in the plot above, we can transform the SOC -ccount into a linear equation. The equation is
y = alpha_gain * x + alpha_bias
where y = SOC, x = ccount, alpha_gain is 0.00046 and alpha_bias is 0.0 for the above plot.

In reality however, the cells might have more or less than 60Ah. We are also not discharging till 0Ah. So, 0% SOC doesn't necessarily mean 0 ccount and 100% doesn't mean 216000 ccount. The real graph could look like below assuming that the cell has a bit more than 60Ah.



In such case, the alpha_bias and alpha_gain are going to be different. Therefore, we need an algorithm that can determine the alpha_bias and alpha_gain over some period of time.

Algorithm

First, we need to assume that coulomb counting is consistent over charge cycles. While this might be an invalid assumption, the algorithm should neglect it.

There are only two cases where we do not predict but actually know what the state of charge of the battery pack is.

a) SOC is 100% when the battery pack has charged to the fullest.

b) SOC is 0% when the battery pack has been drained to the safety limit.

If we are coulomb counting, then we have data that look like below over charge cycles.

x (ccount)	y (SOC)
210	0
217000	100
230	0
217880	100

This becomes a linear regression problem where we figure out the slope and intercept. Gradient Descent algorithm starts with an initial bias and gain and slowly descent to the correct values. In short, the equation used is

alpha_gain -= ccount * learning_rate * (predictedSOC - actualSOC);

alpha_bias -= learning_rate * (predictedSOC - actualSOC);

where, predictedSOC is the SOC predicted with the old bias and gain. Learning rate is the rate at which we want the algorithm to approach the real values. A high rate will overshoot and a very low rate will be very slow to reach the real values.

During trials and experiments, it was found that the gain approaches faster than the bias because x is huge compared to y. Therefore, to slow down the gain approaching,
alpha_gain -= ccount * learning_rate * (predictedSOC - actualSOC) / 1000000000;

Display SOC

A disadvantage of gradient descent is that we will not get the real gain and bias values immediately but will approach them. While the algorithm will reach the correct values over some charge cycles, SOC calculation will be off until we reach the correct gain and bias values. One of the requirements of SOC algorithm is that it displays 0% when discharged and 100% as soon as charging ends.

To solve this, the predicted SOC is called real_SOC and a new variable called disp_SOC was created. At the end of a discharge, real_SOC is around 20% if we haven't reached the correct bias and gain values. However, we want to display 0% instead. Therefore, we just create an offset.
Offset = real_SOC - (0% or 100%) .
disp_SOC = real_SOC + offset.
This will ensure that the algorithm displays 0% or 100% at the end of discharge or charge cycle until the correct gain and bias values are reached.