**Round Trip Efficiency Estimation for Battery Energy Storage Systems**

Chart

Description automatically generatedWe start by examining the dataset:

Observations:

* There are no round-trips, which require the SOC going from 100% to 0% and then back to 100%.
* The energy columns have missing values

Consequently, we decide to estimate the RTE based on segments of the SOC curve, that we call **trips**.

A picture containing graphical user interface

Description automatically generated

Trips are a way to replace round-trips.

A trip starts at a local maximum of the SOC curve, and ends when we reach the same State-Of-Charge again

This ignores the oscillations found at low levels of charge

In some cases, to be able to end the trip at the same charge level, we may have to move the starting point from the local maximum.

We deal with missing values in two ways:

* the energy values are taken at the start and at the end of a trip, thus any missing values in between are irrelevant.
* The contribution of a trip to the average RTE is weighted by the Δ-SOC, the difference between the maximum and minimum SOC during the trip.   
  If the difference in charge is very low, for instance because the missing values were due to the battery being disconnected, that segment will be less important

The resulting RTE estimation is:

Average RTE over 16 trips = 86.33%

Average RTE over 16 trips (weighted by each trip's Δ SOC) = **86.27%**

note: If our dataset covered a longer time period, with more trips (e.g. *n*>50), we would be able to obtain a confidence interval for the round-trip efficiency.

If *n* is significant, by the Central Limit Theorem the sample mean of the RTE of a trip follows a normal distribution, and we can define a confidence interval with

(with =sample mean, =necessary value for a confidence level α, =sample standard deviation, =number of samples)

**Code**

File: rte.py

import logging  
import sys  
from math import isnan  
import pandas as pd  
  
*# We do not have roundtrips that go 100-to-0-to-100, so the estimation of the RTE is based on trips:  
# A trip is a segment of the State-Of-Charge curve, starting at a local maximum and ending at the same SOC level***class Trip**:  
 def \_\_init\_\_(self, start\_idx):  
 self.start\_idx = start\_idx  
 self.end\_idx = 0  
 self.energy\_delivered = 0  
 self.energy\_received = 0  
 self.min\_soc = 0  
 self.max\_soc = 0  
  
 *# the minimum and maximum SOC values during the trip are used to weigh its contribution to the average RTE* def set\_max\_soc(self, y3\_soc):  
 self.max\_soc = y3\_soc[self.start\_idx]  
  
 def set\_min\_soc(self, y3\_soc):  
 trip\_soc\_values = [y3\_soc[i] for i in range(self.start\_idx, self.end\_idx)]  
 self.min\_soc = min(trip\_soc\_values)  
  
  
*# The local maxima in the SOC curve constitute the starting point of trips  
# We exclude small oscillations at a low level of charge using the min\_value parameter***def get\_local\_max**(soc\_ls, k=20, min\_value=60):  
 *# Parameters: k = time slots of monotonic increase of the SOC, followed by monotonic decrease  
 # min\_value = the minimum SOC value for a point to be recognized as a local maximum* local\_maxima\_indices = [0] *# the start is included* for i in range(k,len(soc\_ls)-k):  
 prev = soc\_ls[i-k:i]  
 next = soc\_ls[i+1:i+k+1]  
 if all([soc\_ls[i] > soc for soc in prev]):  
 if all([soc\_ls[i] > soc for soc in next]):  
 if soc\_ls[i] > min\_value:  
 local\_maxima\_indices.append(i)  
  
 return local\_maxima\_indices

*# Define the start and the end of a trip, together with the energy delivered and received***def conclude\_trip**(end\_idx, trip, y1\_ed, y2\_er, y3\_soc):  
 proposed\_endpoint\_soc = y3\_soc[end\_idx]  
  
 if abs(y3\_soc[trip.start\_idx] - proposed\_endpoint\_soc) < 1:  
 trip.end\_idx = end\_idx  
  
 else: *# we must change the extremes of the roundtrip. Either:* if proposed\_endpoint\_soc >= y3\_soc[trip.start\_idx]: *# 1) backtrack* for j in range(end\_idx, trip.start\_idx , -1):  
 if abs(y3\_soc[trip.start\_idx] - y3\_soc[j]) < 1:  
 trip.end\_idx = j  
 break  
 trip.end\_idx = None  
 else: *# or 2) bring the starting index forward* for s in range(trip.start\_idx, end\_idx):  
 trip.end\_idx = end\_idx *# endpoint unchanged* if abs(y3\_soc[s] - y3\_soc[end\_idx]) < 1:  
 trip.start\_idx = s  
 break  
 trip.start\_idx = None *# there may be no way to obtain a valid trip (e.g. at the end of the SOC line)  
  
 # update the values of energy delivered, energy received, and the SOC delta of the trip* trip.energy\_delivered = y1\_ed[trip.end\_idx] - y1\_ed[trip.start\_idx]  
 trip.energy\_received = y2\_er[trip.end\_idx] - y2\_er[trip.start\_idx]  
 trip.set\_max\_soc(y3\_soc)  
 trip.set\_min\_soc(y3\_soc)

*# Go from the first to the last time instant, defining the trips and their energy values, necessary to estimate the RTE***def process\_trips():**  
 *# load the data* df = pd.read\_csv("BESS\_op\_data.csv")  
 y1\_ed = df["ENERGY\_DELIVERED"].to\_list()  
 y2\_er = df["ENERGY\_RECEIVED"].to\_list()  
 y3\_soc = df["SOC"].to\_list()  
 lmax\_indices = get\_local\_max(y3\_soc)  
  
 trips\_ls = [Trip(0)]  
  
 for i in range(1,len(y3\_soc)):  
 trip = trips\_ls[-1] *# select the current trip  
 # if we have to start a new trip from a local maximum: conclude the previous one, record its energy values* if i in lmax\_indices:  
 conclude\_trip(i, trip, y1\_ed, y2\_er, y3\_soc)  
 if i != max(lmax\_indices):  
 new\_trip = Trip(i)  
 trips\_ls.append(new\_trip)  
  
 trips\_ls = [t for t in trips\_ls if t.start\_idx is not None and t.end\_idx is not None  
 and not(isnan(t.energy\_received)) and not(isnan(t.energy\_delivered))] *# exclude trip with missing values* return trips\_ls  
  
  
  
**if \_\_name\_\_ == "\_\_main\_\_":**  
  
 trips = process\_roundtrips()  
 rte\_ls = []  
 for trip in trips:  
 trip\_efficiency = trip.energy\_delivered / trip.energy\_received  
 rte\_ls.append(trip\_efficiency)  
  
 logging.debug("Trips efficiency: " + str(rte\_ls))  
 print("Average RTE over " + str(len(trips)) + " trips = " + str(  
 round(sum(rte\_ls) / len(rte\_ls) \* 100, 2)) + "%")  
  
 soc\_deltas = [t.max\_soc - t.min\_soc for t in trips]  
 weights = [delta / sum(soc\_deltas) for delta in soc\_deltas]  
  
 weighted\_avg = sum([weights[i] \* rte\_ls[i] for i in range(len(trips))])  
 print("Average RTE over " + str(len(trips)) + " trips (weighted by each trip's Δ SOC) = " + str(  
 round(weighted\_avg \* 100, 2)) + "%")