Technical University of Munich
Institute for Electrical Energy Storage Technology
Prof. Dr.-Ing. Andreas Jossen

Manual for the validation of state estimation algorithms

This manual describes the validation process based on

[1] Campestrini, C.; Horsche, M.F.; Zilberman, I.; Heil, T.; Zimmermann, T.; Jossen, A.: *Validation and benchmark methods for battery management system functionalities – State of charge estimation algorithms,* In: *Journal of Energy Storage 7*, S. 38–51, 2016

Please follow the steps to validate your algorithm.

1. Download all files:
   - Text file with the synthetic load cycle
   - Matlab files
2. Example data can be downloaded from:
   https://openbattery.org/dataset/validation-data-set-state-charge-estimation-algorithms
   The RawData.mat contains the measurement data for all temperatures and profiles.
   The FilterData.mat contains the estimation results of the Kalman Filter and the extended Kalman Filter. Copy both files into the Example folder (<./Benchmark/Example>).
3. Apply profile A, B and C to your battery at the five different temperatures based on [1] with your test system. Measure voltage, current and temperature also with your battery management system.
4. When all measurements are completed, apply the data to your state estimator
5. Prepare your estimation results. Please see the example data for help.
   Your result data has to be a struct in the following shape:
   `FilterResult.<AlgoName>.<TestName>.<Temperature>.<Profile>.<Data>`

   Placeholders:
   - `AlgoName`: Name of your state estimator. You can also test and compare more algorithms. To do so, add all results.
   - `TestName`: Add all test results according to [1]
     - `InitOk`: all ok
     - `InitFail`: wrong initial state of charge
     - `IOffset`: with current offset
     - `UOffest`: with voltage offset
     - `ROffset`: with resistance offset
   - `Temperature`: Add the results of all temperatures
     - `n10`: -10 °C
     - `p00`: 0 °C
     - `p10`: 10 °C
     - `p25`: 25 °C
     - `p40`: 40 °C
     - `xxx`: varying temperature (only for Profile C)

- `Profile`: Add the results of all profiles
  - `A`
  - `B`
  - `C`
  - For the InitFail test add also `A2` instead of C. A2 is like profile A but the start point is shifted (for details see [1]).
- `Data`: Add the result data
  - `RefSOC`: reference SOC
  - `RefSOCBMS`: the SOC development of the BMS coulomb counter
  - `RefTime`: timestamp of the reference
  - `EstSOC`: estimated SOC
  - `EstTime`: timestamp of the estimation
  - `ResChSOC`: residual charge

Example: `FilterData.SEKF_1RC.IOffset.p00.B.EstSOC` is the estimation of the extended Kalman filter during profile B at 0 °C. The test was performed with a current offset. To convert your data please use function `FilterResult=PrepResultData()`. Here you can write your own convert script.

6. The function `ProfileTiming=ProfileDataTiming()` sets the start and end time for the constant charge and discharge periods of profile A. This is required to calculate the drift behaviour during the constant charge and discharge periods. Here you can write your own script. Add the required information to a data struct with the following shape: `ProfileTiming.<TestName>.<Temperature>.<Profile>.<Data>=[Start, End]`

   Placeholder:
   - `Profile`: only Profile A and A2 require this information
   - `Data`: Add here the start and end time for all constant charge and discharge periods
     - EstTimeCha1: Index of start and end time for the constant charge period of the BMS data
     - RefTimeCha1: Index of start and end time for the constant charge period of the reference data
     - EstTimeDis1: Index of start and end time for the first constant discharge period of the BMS data
     - RefTimeDis1: Index of start and end time for the first constant discharge period of the reference data
     - EstTimeDis2: Index of start and end time for the second constant discharge period of the BMS data
     - RefTimeDis2: Index of start and end time for the second constant discharge period of the reference data

7. The function `InputDataCheck(FilterResult)` checks your result data. It returns an error in case of a non-complete data set.

8. Now you can run the Benchmark function:
   `Benchmark(Setup,FilterResult,ProfileTiming)`

For questions or help please contact c.campestrini@tum.de