Report: Parameter Identification and State Estimation of an Battery by Using Second Order Equivalent Circuit Model

İ. Ç. Yılmaz *Apr 23, 2021*

1 Introduction

It is well known that accurate identification of the key state parameters and State of Charge (SOC) estimation method for a Li-ion battery cell is of great significance for advanced battery management system (BMS) of electric vehicles (EVs). Systematic experimental data-driven parameter identification techniques and sophisticated SOC estimation algorithms (also can be called as filtering algorithm) for a battery equivalent circuit model in EV applications. The key state parameters of the battery cell were identified based on the second-order resistor capacitor (RC) equivalent circuit model [4].

For EVs and HEVs, the accurate state parameters and state-of-charge (SOC) of an battery is of great significance in real-time control and high-performance operation for advanced battery management system (BMS) of EVs because these parameters are often used to implement the optimum control of charging and discharging process, which is not only beneficial for efficient vehicular BMS, but also for the diagnosis and prognosis of the battery behavior. Therefore, it is necessary to obtain the inner state parameters and to make an accurate SOC estimation for the battery accurately [4].

There are a wide range of approaches proposed in the literature, most of which can be seen in the Fig 1. In this study, model-based estimation strategy with electrical models are intended to identify key parameters of the second order equivalent circuit. Adaptive filters and some observers are extensively employed for estimating the SOC. These filter and observer may include separate or collaborative usage of the Kalman Filter (KF), Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), particle filters, and Least-squares-based filters due to batteries' closed-loop nature and concerning various uncertainties.

As the battery capacity, the impedance parameters of the battery (RC parameters) in a new state are mainly defined by the battery design, but changes significantly over the lifetime due to aging processes. The knowledge of the present battery impedance parameters is mainly required for

- estimation of the energy losses in the battery during the operation,
- SOC estimation based on electrical models,
- prediction of the available power of the battery.

Especially for the latest task the impedance parameters must be determined very accurate including their dependence on the current. All methods for the estimation of the battery

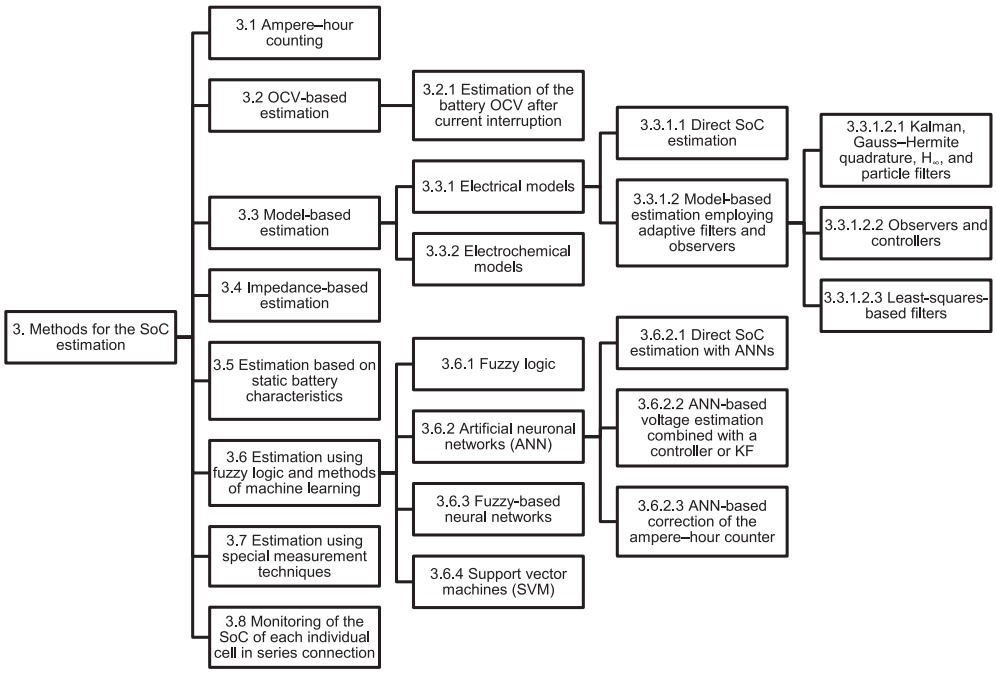

Figure 1: Classification of the methods for the SOC estimation. [8]

impedance can be divided into the three groups and the mentioned groups can be seen in Fig 2.

2 An Equivalent Second-Order Circuit Model (ECM) of An Battery

The research showed that the second-order RC model has the highest precision and is more suitable for the voltage estimation of battery cells. He et. al. [2] use online parameter identification method to determine the relationship between the model accuracy and the number of RC networks, and concluded that the model with a second-order RC network has the best performance.

As shown in Fig 3, the second-order RC battery model is composed of an open-circuit voltage (OCV) denoted by $U_{oc}(SOC)$ which is a function of SOC, a resistance R_Ω, and two parallel RC networks connected in series (i.e., $R_1 = R_{ep}$, $C_1 = C_{ep}$ and $R_1 = R_{cp}$,$C_1 = C_{cp}$). The resistance R_Ω is the Ohmic resistances caused by the accumulation and dissipation of charge in the electrical double-layer; R_1 and C_1 are the electro-chemical polarization resistance and capacitance, respectively; R_2 and C_2 are the concentration polarization resistance and capacitance, respectively; $U_1 = U_{ep}$ and U_{cp} are the polarization voltage across C_1 and C_2, respectively; $I(t)$ is the load current (supposed positive for discharge and negative for charge); and U_t is battery the terminal voltage.

The electrical behavior of the second-order RC battery model can be governed in the con-

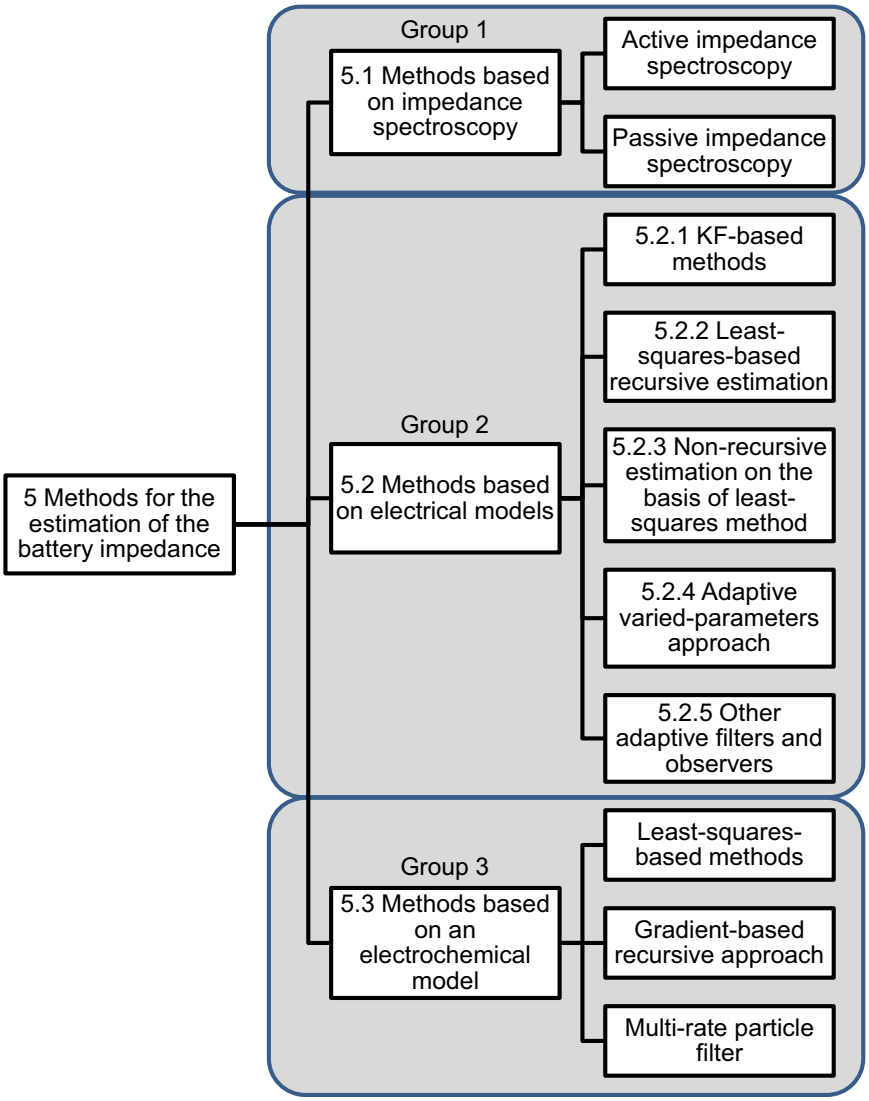

Figure 2: Classification of the methods for the estimation of the battery impedance.[8]

tinuous time domain by following three equations as follows:

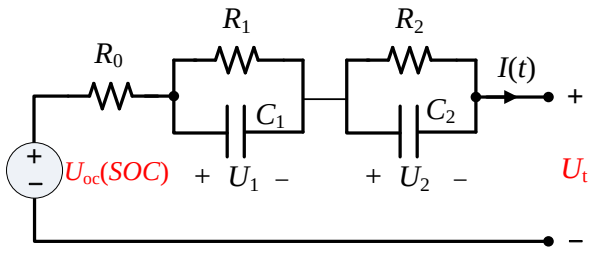

Figure 3: Schematic diagram of the second-order RC model.

$$\dot{U}_1(t) = -\frac{1}{R_1 C_1} U_1(t) + \frac{1}{C_1} I(t) \tag{1a}$$

$$\dot{U}_2(t) = -\frac{1}{R_2 C_2} U_2(t) + \frac{1}{C_2} I(t) \tag{1b}$$

$$U_t(t) = U_{oc}(SOC)(t) - U_1(t) - U_2(t) - I(t)R_0 \tag{1c}$$

Mathematical relations involving SOC in continuous time domain can be written as follows:

$$SOC(t) = SOC(t_0) - \frac{1}{Q} \int_{t_0}^{t} \eta I(\tau) d\tau \tag{2}$$

or

$$\dot{SOC}(t) = -\frac{\eta I(t)}{Q} \tag{3}$$

where $I(t)$ is instantaneous cell current (assumed positive for discharge, negative for charge). The sign of $I(t)$ is positive on discharge. Hence, positive (discharge) current lowers the cell's state of charge, and negative (charge) current increases the cell's state of charge. Care must be taken with units: $I(t)$ is measured in amperes, and to be compatible, Q must be converted to ampere-seconds (i.e., coulombs) [5]. Q is the cell nominal capacity. Cell Coulombic efficiency η is $\eta = 1$ for discharge, and $\eta = \eta \leq 1$ for charge. Using a rectangular approximation for integration and a "suitably small" sampling period Δt, a discrete-time approximate recurrence may then be written as

$$SOC[k+1] = SOC[k] - \left(\frac{\eta \Delta t}{Q}\right) I[k] \tag{4}$$

where $SOC[k+1]$ and $SOC[k]$ are the SOC at $[k+1]$th and kth sampling time, respectively; η is the Coulomb efficiency that is assumed to be 1 at charging and 0.98 at discharging as the battery works in a limited current range. Then, the continuous time state-space representation with 1, 3 can be written as:

$$\textit{Continuous Time LTI State Space} \begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \\ h(t) = y(t) - U_{OC}(SOC(t)) = \boldsymbol{C}\boldsymbol{x}(t) + Du \end{cases} \tag{5}$$

where $\boldsymbol{A} = \begin{bmatrix} -\frac{1}{R_1 C_1} & 0 & 0 \\ 0 & -\frac{1}{R_2 C_2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} \frac{1}{C_1} \\ \frac{1}{C_2} \\ -\frac{\eta}{Q} \end{bmatrix}$, $\boldsymbol{C} = \begin{bmatrix} -1 & -1 & 0 \end{bmatrix}$, $D = R_0$. State vector of the state space system is denoted by $\boldsymbol{x} = [U_1 \; U_2 \; SOC]^T$ and $u(t) = I(t)$ indicates the input of the battery system and $y(t) = U_t(t) - U_{OC}(SOC)$ is the output.

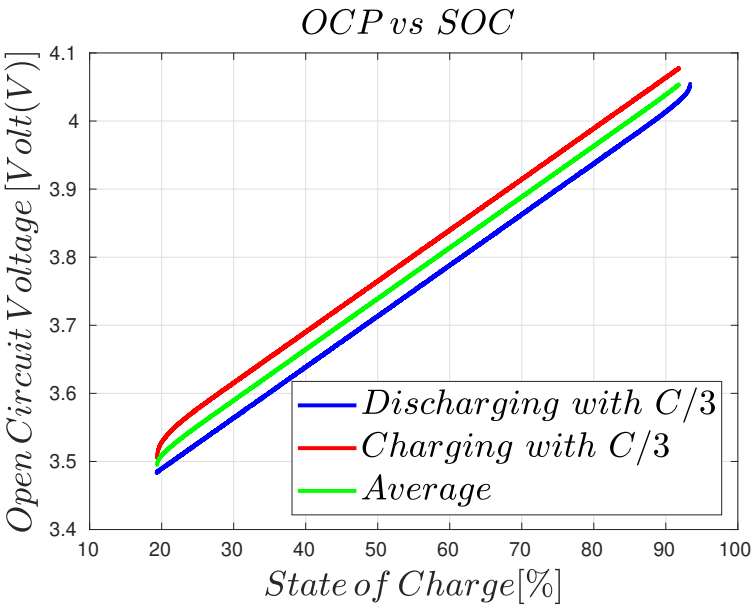

Figure 4: Experimental SOC-OCV mapping when battery is charged/discharged with $C/3$ capacity.

3 SOC - OCP Relationship

The SOC-OCV function is therefore representative for a particular battery, and is generally a nonlinear monotone function between SOC and OCV for all lithium-ion batteries. It is hence widely used in battery management systems (BMS) for correcting SOC calculation.[11] The accuracy of the SOC-OCV curve has great influence on the SOC value estimated.It is consequently important to determine the SOC-OCV relationship precisely, if an accurate estimation of the battery state is necessary.

NOTE [Verification of OCV Model]: To validate the proposed OCV model, a series of experiments are performed on different classes of batteries to obtain SOC-OCV mapping data. Some experiment is performed with the battery charged from its fully discharged state at a current of 0.05 C. The charging continued until the terminal voltage of the battery reached the charging voltage limit. The battery was subsequently open-circuited for 2 h, after being discharged at 0.05 C until the battery terminal voltage reached the discharge voltage limit. Taking the average potential between the charge and discharge branch at $C/20$ or $C/30$ and the normalized $C/20$ or $C/30$ capacity, the voltage and its corresponding SOC can be regarded as OCV versus SOC curve [1]

To reach a acceptable OCV-SOC relation, three OCV models are selected in the literature and all of them is compared to each others. All of the parameters in the OCV models are indicated 1 and they are found by employing the MATLAB *Curve Fitting* toolbox in R2018b. All three screenshot of the curve fittings and related plots can be found at the Appendix A section.

3.1 Matlab Code to Obtain OCV-SOC RelationShip

In this code snippet, 'SoC_OCV_x_axis' and 'SoC_OCV_Y_axis' vectors are employed in the curve fitting. The SOC of the battery is taken between 0 and 1 instead of percentage value.

```
load('BAT_data.mat');

%% Find Open Circuit Voltage (OCV)
%  Obtain the the interval when circuit discharging with Q/3
%  Obtain the the interval when circuit charging with Q/3
%  then take the value of SoC and u at the time

% This matrix includes time, SoC, and OCV when current is zero

SOC_OCV_Discharge = [t_vec(9761:17818) SoC(9761:17818,1) u
    (9761:17818)];

SOC_OCV_Charge = [t_vec(17819:25820) SoC(17819:25820,1) u
    (17819:25820)];


if min(SOC_OCV_Discharge(:,2)) > min(SOC_OCV_Charge(:,2))
     SOC_x_min = min(SOC_OCV_Discharge(:,2));
else
     SOC_x_min = min(SOC_OCV_Charge(:,2));
end

if max(SOC_OCV_Discharge(:,2)) > max(SOC_OCV_Charge(:,2))
     SOC_x_max = max(SOC_OCV_Charge(:,2));
else
     SOC_x_max = max(SoC_OCV_Discharge(:,2));
end

 % Create Vector Of Common X-Values
OCV_SOC_x = linspace(SOC_x_min, SOC_x_max, 8000);

% Interploate To New Values For Discharging
OCV_SOC_y_Disc = interp1(SOC_OCV_Discharge(:,2), ...
                         SOC_OCV_Discharge(:,3), OCV_SOC_x(:), '
                            spline', ...
                         'extrap');

% Interploate To New Values For Charging
OCV_SOC_y_Charge = interp1(SOC_OCV_Charge(:,2), ...
                         SOC_OCV_Charge(:,3), OCV_SOC_x(:), '
                            spline', ...
```

```
                                    'extrap');

% Mean Of Y Values
OCV_SOC_y_mean = mean([OCV_SOC_y_Disc  OCV_SOC_y_Charge], 2);

% Axes for Curve Fitting Application
SoC_OCV_x_axis =  0.01 * OCV_SOC_x;
SoC_OCV_y_axis =  OCV_SOC_x;
%%
```

The RMSE value of each model is taken as a criterion for selection of the most suitable OCV model. It can be understood that more precise experiment must be done to be reached the appropriate OCV-SOC models. In literature, batteries is charged and discharged nearly at $C/20$ or $C/30$. Then the average potential between the charge and discharge branch is calculated to achieve experimental OCV-SOC curvature. The purpose of the slow rate is to minimize the excitation of the dynamic parts of the cell model. The cell in a quasi-equilibrium state is desired to keep at all times. So, $C/30 - C/20$ rate is generally used for which is a compromise between the desire to have zero current to have a true equilibrium and the practical realization that the test will already require on the order of 40 to 60 h to complete within above mentioned charge and discharged levels. Model - 2 is employed to able to associate between OCV and SOC in this study with the parameters found in A.2 **In the given data, slow charging and and discharging was not performed during the experiment, so that charging and discharging with $C/3$ capacity rate is taken into consideration**.

4 Parameters Identification of the Battery Model

4.1 Preliminary Work

To identify the model parameters, a constant-current discharge pulse is generally subjected to the cell and then the cell is allowed to rest while recording the cell's voltage response during a fair amount of time [5]. A model type of sort having a single parallel resistor–capacitor subcircuit is assumed to this point, exactly as drawn in Fig 5.

Total capacity denoted by the symbol Q is a parameter of the cell model; that is, it is a constant that may differ from cell to cell. Total capacity is not a function of temperature or current. It is not taken into consideration in the parameter identification calculation, its

#	OCV Model	Reference	RMS Error (mV)
1	$U_{OC}(s) = a + b \times (-ln(s))^m + c \times s + d \times \exp^{n(s-1)}$	[11]	0.4798
2	$U_{OC}(s) = K_0 - \frac{K_1}{s} - K_2 s + K_3 ln(s) + K_4 ln(1-s)$	[6]	0.2431
3	$U_{OC}(s) = K_0 + K_1 \exp^{-\alpha(1-s)} - \frac{K_2}{s}$	[3]	0.3060

Table 1: Compared fitting results of OCV models.

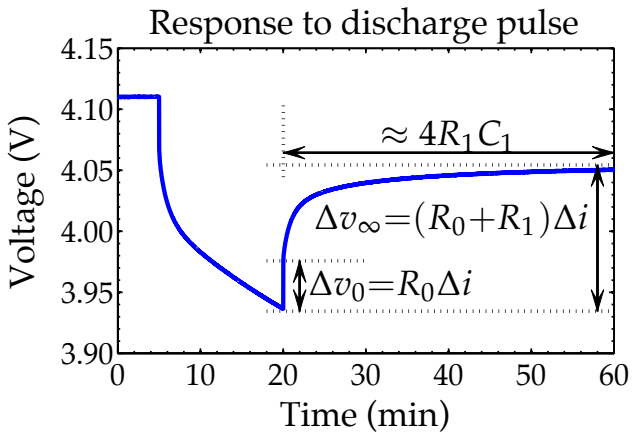

Figure 5: Measuring parameter values from a pulse response for first order circuit [5].

value is directly obtained from the first discharge step which is given in the experimental data by using discrete time equation 4. The MATLAB code in Appendix B snippet is used in the calculation of Q value. Note that the coulombic efficiency, $\eta(t)$ is adopted 1 for discharging.

The circuit of the battery battery model however includes extra concentration polarization resistance and capacitance, R_2 and C_2, respectively. Instead of making an rough mathematical calculation that is explained in detail literature, constrained optimization problem is solved to find parameter values in the time interval between constant current discharge pulse and rest time. Model parameters includes R_0, R_1, τ_1, R_2 and τ_2. Sometimes ohmic resistance value, R_0, is separately calculated for charging, R_0^-, and discharging, R_0^+, process. For the sake of simplicity, R_0 is only identified from discharging pulse. Time constants, $\tau_1 = R_1C_1$ and $\tau_2 = R_2C_2$ are sometimes utilized as parameters rather than using resistor values C_1 and C_2. Eventually, parameter vector can be written as $\boldsymbol{\Theta} = [R_0\ R_1\ \tau_1\ R_2\ \tau_2]^T$. By using experimental data, the constant current discharge pulse start at $t = 28785$ sec and continues until $t = 31109$ sec. Then, battery is allowed to rest approximately half an hour (Rest time finishes at $t = 31109$ sec). The Fig includes the current and terminal voltage measurements during the above mentioned intervals. As it can be understood that, voltage characteristic of the battery at this region resembles like the behaviour which is already mentioned at the beginning of this chapter. State of charge (SOC) is also saved in this interval to calculate OCV of the battery. Total capacity of the battery is found as $Q = 1129.4$ Ampere-second or 313.7 milli-Ampere hour (mAh).

Before starting the parameter identification, continuous time dynamic equation in 5, i.e. $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t)$, must be discretized. Some minor arrangement in the state vector, $\boldsymbol{x}$, state transition matrix, $\boldsymbol{A}$, and input matrix, $\boldsymbol{B}$, is made for identification. State vector consists of series RC circuit voltages, $\boldsymbol{x} = [U_1\ U_2]^T$, and state transition matrix becomes $\boldsymbol{A} = \begin{bmatrix} -\frac{1}{R_1C_1} & 0 \\ 0 & -\frac{1}{R_2C_2} \end{bmatrix}$. Input matrix denoted as $\boldsymbol{B} = \begin{bmatrix} \frac{1}{C_1} \\ \frac{1}{C_2} \end{bmatrix}$. If the state equations $\dot{\boldsymbol{x}}(t) =$

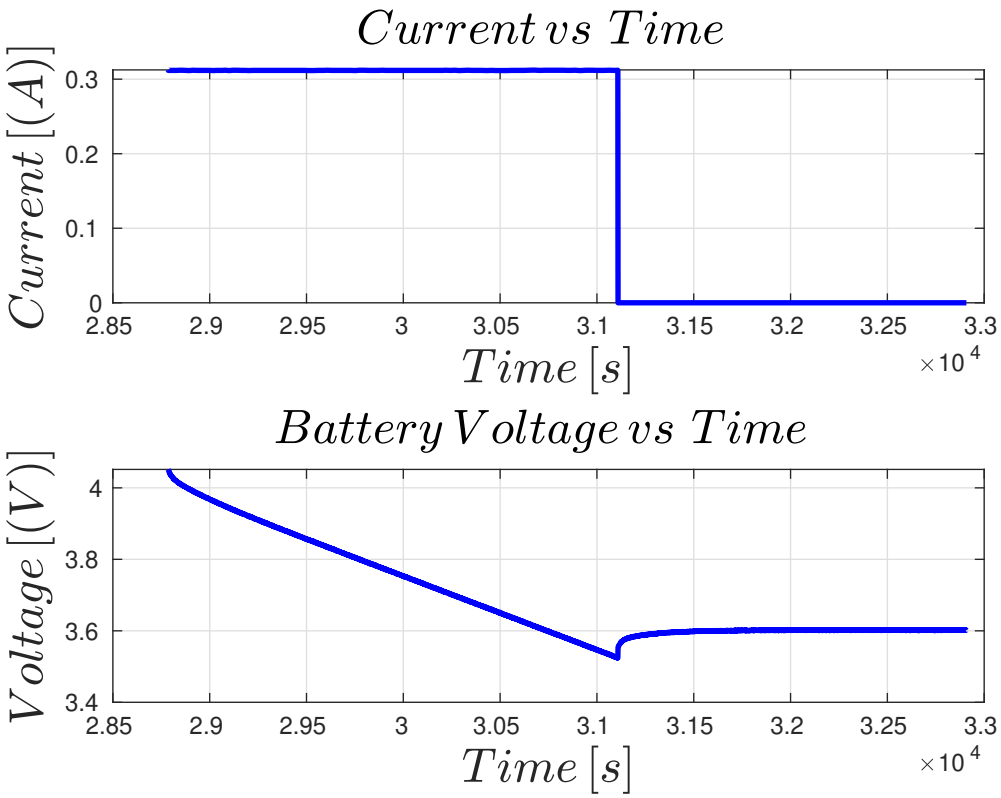

Figure 6: Measurements of current and voltage from a pulse response for second order equivalent circuit parameters identification.

$\boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t)$ are rearranged, and all terms pre-multiplied by the square matrix $e^{-\boldsymbol{A}t}$:

$$e^{-\boldsymbol{A}t}\dot{\boldsymbol{x}}(t) - e^{-\boldsymbol{A}t}\boldsymbol{A}\boldsymbol{x}(t) = \frac{d}{dt}\left(e^{-\boldsymbol{A}t}\boldsymbol{x}(t)\right) = e^{-\boldsymbol{A}t}\boldsymbol{B}u(t) \tag{6}$$

Integration of Eq. 6 gives

$$\int_0^t \frac{d}{d\tau}\left(e^{-\boldsymbol{A}\tau}\boldsymbol{x}(\tau)\right)d\tau = \int_0^t e^{-\boldsymbol{A}t}\boldsymbol{B}u(t) = e^{-\boldsymbol{A}t}\boldsymbol{x}(t) - e^{-\boldsymbol{A}0}\boldsymbol{x}(0) = \int_0^t e^{-\boldsymbol{A}\tau}\boldsymbol{B}u(\tau)d\tau \tag{7}$$

where $e^{-\boldsymbol{A}0} = \boldsymbol{I}$ is 2×2 identity matrix and $[e^{-\boldsymbol{A}t}]^{-1} = e^{\boldsymbol{A}t}$ the complete state vector response may be written in two similar forms

$$\boldsymbol{x}(t) = e^{\boldsymbol{A}t}\boldsymbol{x}(0) + e^{\boldsymbol{A}t}\int_0^t e^{-\boldsymbol{A}\tau}\boldsymbol{B}u(\tau)d\tau \tag{8a}$$

$$\boldsymbol{x}(t) = e^{\boldsymbol{A}t}\boldsymbol{x}(0) + \int_0^t e^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}u(\tau)d\tau \tag{8b}$$

A continuous time signal $\{x(t)\}$ can be obtained from a discrete time (DT) signal $x[k]$, by holding the value of the DT signal constant for one sampling period T or Δt, such that:

$$x(t) = x[k], \quad kT \leq t < (k+1)T \tag{9}$$

This is known as the zero-order hold. The discrete-time equivalent of equation 5 is of the form:

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{A_d}\boldsymbol{x}[k] + \boldsymbol{B_d}u[k] \\ y[k] &= \boldsymbol{C_d}\boldsymbol{x}[k] + D_d u[k] \end{aligned} \tag{10}$$

Starting from the solution of the continuous state-space equation 8, that the corresponding discrete matrices are obtained as:

$$\boldsymbol{A_d} = e^{\boldsymbol{A}\boldsymbol{T}} \quad \text{and} \quad \boldsymbol{B_d} = \left(\int_0^T e^{A\tau} d\tau \right) \boldsymbol{B} \tag{11}$$

It should be note that that this is the *exact* solution to the differential equation, there are no discretization errors. While it is exact, information is still lost by the discretization: the inter-sample behavior. So discrete time dynamic equation can be explicity written as follows:

$$\boldsymbol{x}[k+1] = \underbrace{\begin{bmatrix} e^{-\frac{Tt}{\tau_1}} & 0 \\ 0 & e^{-\frac{T}{\tau_2}} \end{bmatrix}}_{\boldsymbol{A}_d} \boldsymbol{x}[k] + \underbrace{\begin{bmatrix} R_1(1 - e^{\frac{T}{\tau_1}}) \\ R_2(1 - e^{\frac{T}{\tau_2}}) \end{bmatrix}}_{\boldsymbol{B}_d} u[k] \tag{12}$$

where state vector is of the form $\boldsymbol{x}[k] = \begin{bmatrix} U_1[k] \; U_2[k] \end{bmatrix}^T$, and input $u[k]$ is the measured current at each sampling time $I[k]$. And the continuous time matrices $\boldsymbol{C} = \boldsymbol{C_d}$ and $D = D_d$ remain same in discrete time. The discretized equations of the second order RC model may also expressed as:

$$\begin{aligned} U_1[k+1] &= U_1[k]e^{-\frac{T}{\tau_1}} + I[k]R_1\big(1 - e^{\frac{T}{\tau_1}}\big) \\ U_2[k+1] &= U_2[k]e^{-\frac{T}{\tau_2}} + I[k]R_2\big(1 - e^{\frac{T}{\tau_2}}\big) \\ U_t[k] &= U_{OC}\big[SOC[k]\big] - I[k]R_0 - U_1[k] - U_2[k] \end{aligned} \tag{13}$$

4.2 Parameter Identification Procedure

To identify the parameters of the given battery model as $\boldsymbol{\Theta} = [R_0 \; R_1 \; \tau_1 \; R_2 \; \tau_2]^T$, the sum of squares errors of the measured terminal voltage data denoted as U_t^{exp} and the simulated terminal voltage data denoted as U_t^{sim} for the terminal voltage at each sampling point of input current is chosen as the objective function, which is represented by the summation of square of the $\ell_2 - norm$ or $Euclidean\, norm$, $\|x\|_2^2$ as follows:

$$\begin{aligned} \min_{\boldsymbol{\Theta}} \quad & J(\boldsymbol{\Theta}, k) = \min_{\boldsymbol{\Theta}} \sum_{k=k_0}^{N} \left(U_t^{exp}[k] - U_t^{sim}[\boldsymbol{\Theta}, k] \right)^2 \\ \text{st.} \quad & \boldsymbol{\Theta}_i \geq 0, \;\; i = 1, \ldots, 5 \\ & \boldsymbol{x}[k+1] = \boldsymbol{A}_d\boldsymbol{x}[k] + \boldsymbol{B}_d u[k], \;\; k = k_0, \ldots, N-1 \\ & U_t^{sim}[k] = U_{OC}\big[SOC[k]\big] + \boldsymbol{C}_d x[k] + D_d u[k], \;\; k = k_0, \ldots, N \end{aligned} \tag{14}$$

where k_0 is starting time and N is the final time of the identification process, input is the measurement current, $I[k] = u[k]$ in the input current and $\boldsymbol{\Theta}$ is the identified parameter vector. (Some ready-to-run implementation is also available for parameter estimation in MATLAB.) Initial value of the state vector is accepted is zero, $\boldsymbol{x}[k = k_0] = \mathbf{0}$, because of the fact that battery is at the idle state before the parameter estimation procedure.

[**NOTE**]: **Solution of this optimization technique is postponed. Particle Swarm Optimization (PSO) technique will be considered for solution.**

4.3 Parameter Identification Based On Recursive Least Square (RLS)[9]

From equation 1, the transfer function of the second-order equivalent circuit model can be written as:

$$G(s) = \frac{U_t(s) - U_{oc}(s)}{I(s)} = -\left(R_0 + \frac{R_1}{1 + \tau_1 s} + \frac{R_2}{1 + \tau_2 s}\right) \tag{15}$$

Then, the transfer function is transformed from $\mathcal{L}$ (Laplace) domain to z domain based on bilinear transformation, which can be expressed as:

$$s = \frac{2(1 - z^{-1})}{T(1 + z^{-1})} \tag{16}$$

where T is the sampling time of the system. Substituting Equation 15 for 16, the discrete transfer function of the system can be written as:

$$G(z^{-1}) = \frac{\theta_3 + \theta_4 z^{-1} + \theta_5 z^{-2}}{1 - \theta_1 z^{-1} - \theta_2 z^{-2}} \tag{17}$$

where parameters $\theta_i (i = 1, 2, ..., 5)$ can be identified by recursive least square method. The relation between RLS parameters and equivalent circuit RC parameters are explained in Appendix C in detail. The RC parameters of the circuit can be obtained by the following equations. Let $y[k] = U_t[k] - U_{oc}[k]$, Equation 17 can be written as follows:

$$y[k] = \theta_1 y[k-1] + \theta_2 y[k-1] + \theta_3 I[k] + \theta_4 I[k-1] + \theta_5 I[k-2] \tag{18}$$

Defining $\boldsymbol{\phi}$ and new parameter vector, $\boldsymbol{\theta}$ as,

$$\boldsymbol{\phi}[k] = \Big[y[k-1]\ y[k-2]\ I[k]\ I[k-1]\ I[k-2]\Big]^T \tag{19}$$

$$\boldsymbol{\theta}[k] = \big[\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\big]^T \tag{20}$$

Then Equation 18 can be written in vector form as follows:

$$y[k] = \boldsymbol{\phi}[k]^T \boldsymbol{\theta}[k] \tag{21}$$

Defining the estimator of $\boldsymbol{\theta}$ as $\hat{\boldsymbol{\theta}}$, Equation 21 can be expressed as:

$$y[k] = \boldsymbol{\phi}[k]^T \hat{\boldsymbol{\theta}}[k] + \varepsilon[k] \tag{22}$$

When the square sum of ε the output error is minimum, the parameters are optimal, and the mathematical formula can be written as:

$$\min_{\hat{\boldsymbol{\theta}}} J(\hat{\boldsymbol{\theta}}, k) = \min_{\hat{\boldsymbol{\theta}}} \sum_{k=k_0}^{N} \varepsilon[k]^2 = \min_{\hat{\boldsymbol{\theta}}} \sum_{k=k_0}^{N} \Big(y[k] - \boldsymbol{\phi}[k]^T \hat{\boldsymbol{\theta}}[k] \Big)^2 \tag{23}$$

Taking the derivation of $\hat{\boldsymbol{\theta}}$ with respect to $\boldsymbol{\theta}$ and then equal to zero, the solution can be obtained:

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{\Psi}^T \boldsymbol{\Psi})^{-1} \boldsymbol{\Psi}^T \boldsymbol{Y} \tag{24}$$

where vectors compose of past measurements $\boldsymbol{\Psi} = \big[\boldsymbol{\phi}[k_0]\ \boldsymbol{\phi}[k_0+1]\ \dots\ \boldsymbol{\phi}[N]\big]^T$ and big output vector $\boldsymbol{Y} = \big[y[k_0]\ y[k_0+1]\ \dots\ y[N]\big]^T$. Step of the RLS algorithm is explained following chapter.

4.3.1 The Parameter Estimation Algorithm

The algorithm for recursive least squares estimation is summarized as follows.

- Initialize the estimator:

$$\begin{aligned} \hat{\boldsymbol{\theta}}[k_0] &= E\big[\boldsymbol{\theta}[k_0]\big] \\ \boldsymbol{P}[k_0] &= E\Big[\big(\boldsymbol{\theta}[k_0] - \hat{\boldsymbol{\theta}}[k_0]\big)\big(\boldsymbol{\theta}[k_0] - \hat{\boldsymbol{\theta}}[k_0]\big)^T\Big] \end{aligned} \tag{25}$$

  In the case of no prior knowledge about $\boldsymbol{\theta}[k_0]$, simply let $\boldsymbol{P}[k_0] = \infty \boldsymbol{I}$. In the case of perfect prior knowledge, let $\boldsymbol{P}[k_0] = 0$.

- Update the estimate $\hat{\boldsymbol{\theta}}[k]$ and the covariance of the estimation error sequentially, which are listed below:

$$\boldsymbol{K}[k] = \boldsymbol{P}[k-1]\,\boldsymbol{\phi}[k]\big(\boldsymbol{\phi}[k]^T \boldsymbol{P}[k-1] \boldsymbol{\phi}[k] + 1\big)^{-1} \tag{26a}$$

$$\boldsymbol{P}[k] = \big(\boldsymbol{I} - \boldsymbol{K}[k]\boldsymbol{\phi}[k]^T\big)\boldsymbol{P}[k-1] \tag{26b}$$

$$\hat{\boldsymbol{\theta}}[k] = \hat{\boldsymbol{\theta}}[k-1] + \boldsymbol{K}[k]\big(y[k] - \boldsymbol{\phi}[k]^T \hat{\boldsymbol{\theta}}[k-1]\big) \tag{26c}$$

```
clc;
clear all
close all;

```

```
5  load('BAT_data.mat');
6
7
8  %% Find the Charge, Discharge and Idle States
9   % Add a state of the battery
10
11   i = -1 * i;
12
13 %% Determination of the Parameters Identification Region
14 %  Constant discharge pulse start at t =  28785 sec
15 %  Then this pulse ends at t = 31109 sec and a rest period start
16 %  Battery stayes at rest until t = 32908 sec
17
18 Param_Identification_Vec =  [t_vec(28784:32909) i(28784:32909) ...
19                             SoC(28784:32909,1) u(28784:32909)];
20
21 % INITIALIZATION
22 %
      ****************************************************************
23 %Inital guess of circuit values comes from "main_depracated.m"
24 R_0 = 0.1402; R_1 = 0.1152; tau_1 = 3.2236;
25 R_2 = 0.1136; tau_2 = 2.4826;
26
27 C_1 = tau_1 / R_1; C_2 = tau_2 / R_2; % capacitor values
28
29 RC_Values = [R_0; R_1; C_1; R_2; C_2];
30
31 % Get the initial parameter vector Look Eqn (25)
32 theta_vector = RC_Values_to_RLS_Parameters(RC_Values);
33
34 % Check the following function to calculcation are valid or not!
35 RLS_Parameters_to_RC_Values(theta_vector);
36
37 % initial estimation error covariance
38 % Take P vector as large as big
39 % http://www.cs.tut.fi/~tabus/course/ASP/LectureNew10.pdf (page -
      8)
40
41 delta = 10e10;
42 P = delta * eye(5);
43 %
      ****************************************************************
44
```

```
% ESTIMATION ALGORITM
%
    ************************************************************************

% Take two sample before

% RC Vector
RC_Vector = [];

for index = 3:1:length(Param_Identification_Vec(:,1))

    y_k_1 = Param_Identification_Vec(index-1,4) - ...
                OCV_SOC_Function(Param_Identification_Vec(index
                    -1,3));

    y_k_2 = Param_Identification_Vec(index-2,4) - ...
                OCV_SOC_Function(Param_Identification_Vec(index
                    -2,3));

    I_k = Param_Identification_Vec(index,2);

    I_k_1 = Param_Identification_Vec(index-1,2);

    I_k_2 = Param_Identification_Vec(index-2,2);

    phi_vector = [y_k_1; y_k_2; I_k; I_k_1; I_k_2];


    % Calculate gain vector
    K = P * phi_vector * (phi_vector' * P * phi_vector + 1)^(-1);

    % Update estimation error covariance
    P = (eye(5) - K * phi_vector') * P;

    % Update the recursive least square paramater vector
    y_k = Param_Identification_Vec(index,4) -...
            OCV_SOC_Function(Param_Identification_Vec(index,3));

    theta_vector = theta_vector + K * (y_k - phi_vector' *
        theta_vector);

    RC_Vector = [RC_Vector; Param_Identification_Vec(index,1) ...
        RLS_Parameters_to_RC_Values(theta_vector)'];

end
```

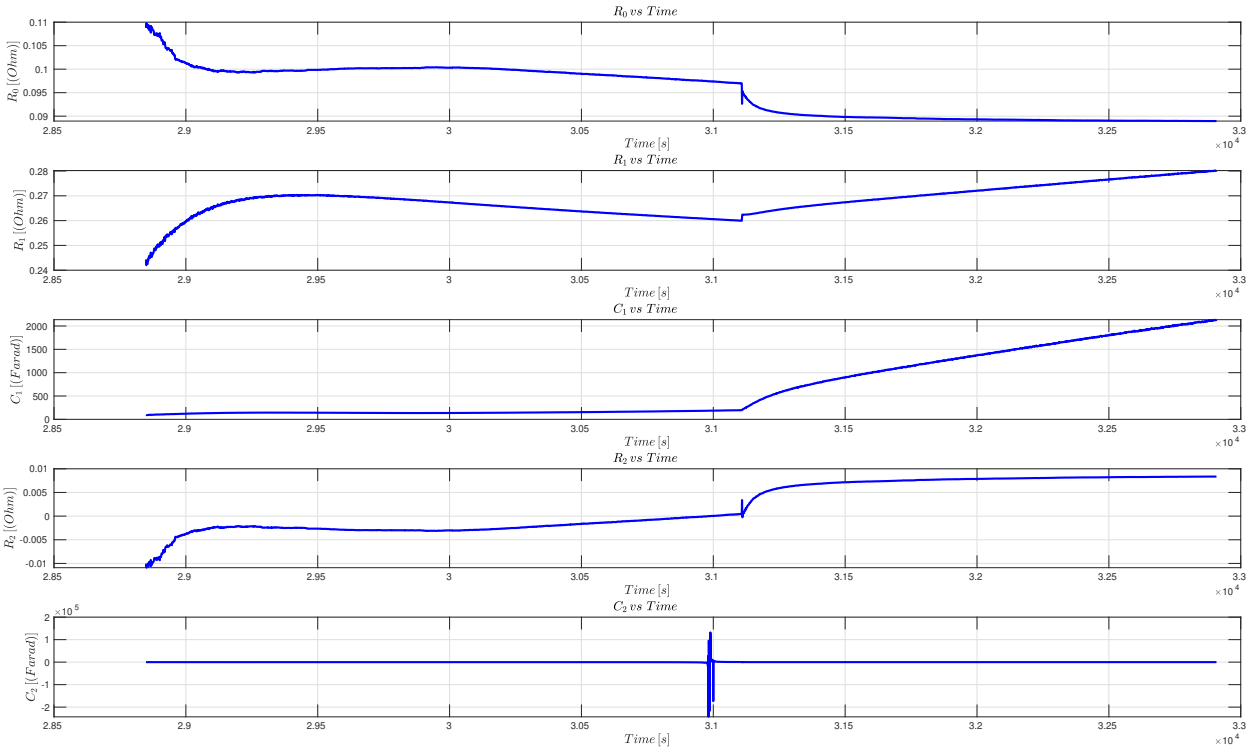

Figure 7: Parameter identification results of the RLS algorithm for parameters R_0, R_1, R_2, C_1 and C_2.

```
%
**********************************************************************************************
```

```
% Remove first and second rows of the vector
Param_Identification_Vec(1:2,:) = [];
```

Parameter identification results of least square algorithm is given in the Fig . As it can be seen in subplots, some parameters of the equivalent circuit element is found in negative. The main reason of this observation is thought that OCV-SOC relation is inadequately obtained.

5 SOC Estimation Method Based on the Adaptive EKF [4]

The adaptive extended Kalman filter is used to make the SOC estimation because the covariance parameters in AEKF approach are not taken as constant, but adaptively updated online with a dedicated SOC estimator which can enhance the estimation performance with respect to the EKF. The control input of the SOC estimator is the current representing the behavior of the battery during discharge or charge process, the output the SOC estimator is the SOC value estimated by the AEKF algorithm.

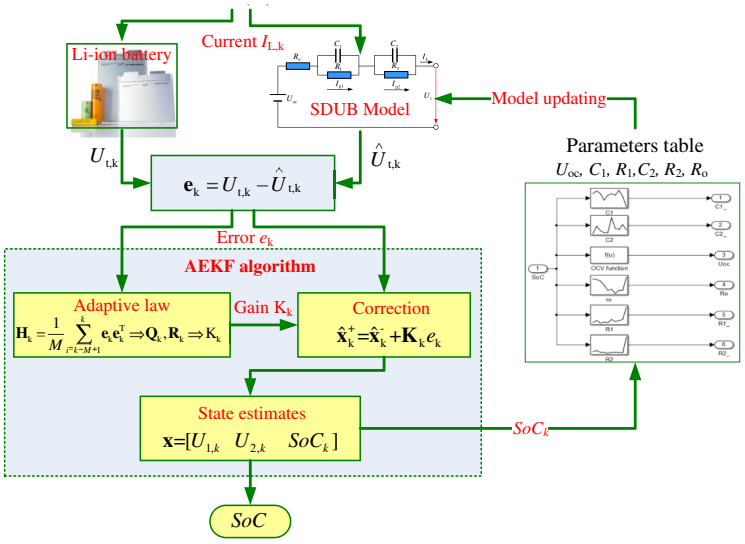

Figure 8: The implementation flowchart of the data driven-based SoC estimation approach with AEKF algorithm. [10]

5.1 AEKF Algorithm

Extended Kalman filter's performance is strongly dependent on the accuracy of the predetermined noise matrix. Thus, it is necessary for the AEKF algorithm to adopt this problem in battery applications. To apply the AEKF for the SOC estimation, it is necessary to reform a state-space form as shown in Equation 12. Discrete-time SOC equation 4 is also written as a dynamic equation and new state space representation becomes:

$$\begin{aligned} \boldsymbol{x}[k+1] &= f(\boldsymbol{x}[\boldsymbol{k}], u[k]) = \boldsymbol{A_d}\boldsymbol{x}[k] + \boldsymbol{B_d}u[k] + \boldsymbol{w}[k] \\ y[k] &= h(\boldsymbol{x}[\boldsymbol{k}], u[k]) + v[k] \end{aligned} \tag{27}$$

Explicity,

$$\boldsymbol{x}[k+1] = \underbrace{\begin{bmatrix} e^{-\frac{Tt}{\tau_1}} & 0 & 0 \\ 0 & e^{-\frac{T}{\tau_2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}_d} \boldsymbol{x}[k] + \underbrace{\begin{bmatrix} R_1(1-e^{\frac{T}{\tau_1}}) \\ R_2(1-e^{\frac{T}{\tau_2}}) \\ -\frac{\eta T}{Q} \end{bmatrix}}_{\boldsymbol{B}_d} u[k] + \boldsymbol{w}[k] \tag{28}$$

The output equation is the same as it used to be 13. The state vector with SOC is denoted by $\boldsymbol{x}[k] = \begin{bmatrix} U_1[k] & U_2[k] & SOC[k] \end{bmatrix}^T$. $\boldsymbol{w}[k]$ is the unmeasured process noise that affects the system state and $v[k]$ is the measurement noise which does not affect the system state, but can be reflected in the system output estimation $y[k]$. $\boldsymbol{w}[k] \sim \mathcal{N}(0, \boldsymbol{Q})$ is assumed to be Gaussian white noise with zero mean and covariance $\boldsymbol{Q}[k]$; $v[k] \sim \mathcal{N}(0, \boldsymbol{R})$ is assumed to be Gaussian white noise with zero mean and covariance $R[k]$. The AEKF provides a further innovation using the filter's innovation sequence and the innovation allows the parameters

$\boldsymbol{Q}$ and R to be estimated and updated iteratively.

The matrix $\boldsymbol{C}$ that is employed in the AEKF filter is the derivative of the output equation 27 with respect to state vector before estimation time, i.e. $\boldsymbol{C}[k] = \left.\frac{\partial h(\boldsymbol{x}[k], u[k])}{\partial \boldsymbol{x}}\right|_{x=\hat{x}[k]^-} = \left[-1 \;\; -1\,\frac{dU_{oc}(SOC)}{dSOC}\Big|_{S\hat{O}C[k]^-}\right]$. As it can be seen in C matrix, the derivation of the OCV-SOC model function, $\frac{dU_{oc}(SOC)}{dSOC} = K_1 SOC^{-2} - K_2 + \frac{K_3}{SOC} - \frac{K_4}{1-SOC}$ is utilized from the second row of Table 1. The AEKF Algorithm is explained in the following Algorithm 1 by using above equations and relations.

Algorithm 1: The AEKF Algorithm

Input: $u[k] = I[k],\ k \in \{k_0, \dots N-1\}$

Output: $\widehat{SOC}[k],\ k \in \{k_0, \dots N-1\}$

Data: $y[k] = U_t^{exp}[k],\ k \in \{k_0, \dots N\}$, $R_0[k]$, $R_1[k]$, $C_1[k]$, $R_2[k]$, $C_2[k]$

Step 1: Initialization:

$\hat{\boldsymbol{x}}[k_0-1]^+ = E\big[\boldsymbol{x}[k_0-1]\big]$, $\hat{\boldsymbol{P}}[k_0-1]^+ = E\left[\left(\boldsymbol{x}[k_0-1] - E\big[\hat{\boldsymbol{x}}[k_0-1]^+\big]\right)\left(\boldsymbol{x}[k_0-1] - E\big[\hat{\boldsymbol{x}}[k_0-1]^+\big]\right)^T\right]$

Step 2: Calculation:

for $k = k_0 \to N$ **do**

1. **State Estimation Propagation:** $\hat{\boldsymbol{x}}[k]^- = f\left(\hat{\boldsymbol{x}}[k-1]^+, u[k]\right)$ (look 27)
2. **State Estimation Covariance:** $\boldsymbol{P}[k]^- = \boldsymbol{A_d}[k]\boldsymbol{P}[k-1]\boldsymbol{A_d}[k]^T + \boldsymbol{Q}[k-1]$
3. **Error Innovation:** $e[k] = y[k] - h\left(\hat{\boldsymbol{x}}[k]^-, u[k]\right)$
4. **Adaptive Law:** $H[k] = \frac{1}{M}\sum_{i=k-M+1}^{k} e[k]^2$, $R[k] = H[k] - \boldsymbol{C}[k]\boldsymbol{P}[k]^-\boldsymbol{C}[k]^T$
5. **Kalman Gain Matrix:** $\boldsymbol{K}[k] = \boldsymbol{P}[k]^-\boldsymbol{C}[k]^T\left(\boldsymbol{C}[k]\boldsymbol{P}[k]^-\boldsymbol{C}[k]^T + R[k]\right)^{-1}$
6. **State Estimate Measurement Update:** $\hat{\boldsymbol{x}}[k]^+ = \hat{\boldsymbol{x}}[k]^- + \boldsymbol{K}[k]e[k]$
7. **State Covariance Measurement Update:** $\boldsymbol{Q}[k] = \boldsymbol{K}[k]H[k]\boldsymbol{K}[k]^T$, $\boldsymbol{P}[k]^+ = \left(\boldsymbol{I} - \boldsymbol{K}[k]\boldsymbol{C}[k]\right)\boldsymbol{P}[k]^-\left(\boldsymbol{I} - \boldsymbol{K}[k]\boldsymbol{C}[k]\right)^T + \boldsymbol{K}[k]\boldsymbol{R}[k]\boldsymbol{K}[k]^T$

/* $\boldsymbol{A}_d[k] = \left.\frac{\partial f(\boldsymbol{x}[k], u[k])}{\partial \boldsymbol{x}}\right|_{x=\hat{x}[k]^-}$, $\boldsymbol{C}[k] = \left.\frac{\partial h(\boldsymbol{x}[k], u[k])}{\partial \boldsymbol{x}}\right|_{x=\hat{x}[k]^-}$ */

It should be pointed out that $\hat{\boldsymbol{x}}[k]^-$ and $\hat{\boldsymbol{x}}[k]^+$ are both estimations of the same vector $\boldsymbol{x}[k]$.

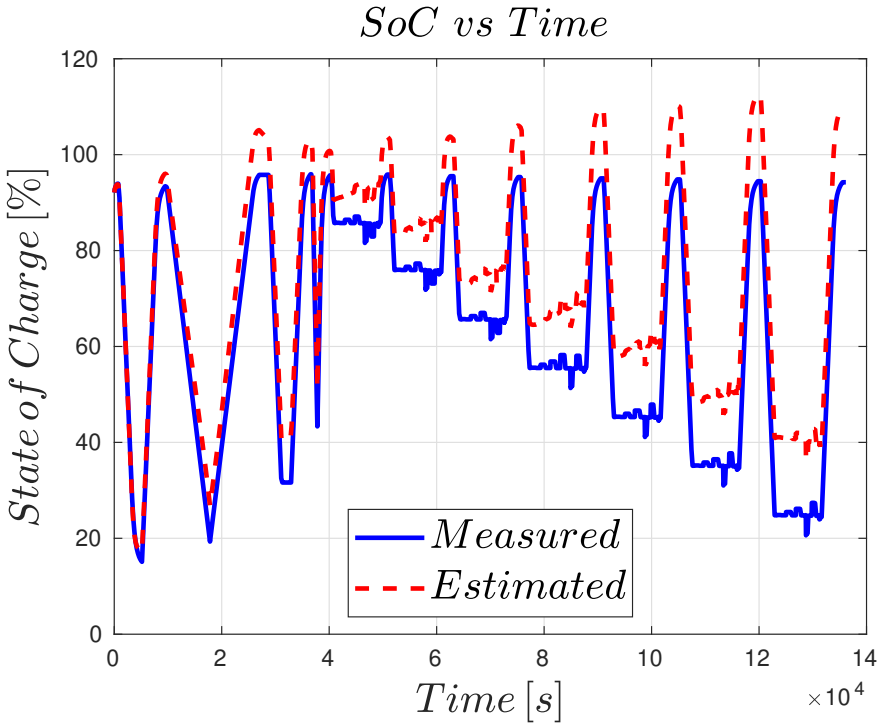

Figure 9: Validation results for SOC estimation by using AEKF.

However,$\hat{\boldsymbol{x}}[k]^-$ is the estimate of $\hat{\boldsymbol{x}}[k]$ before the measurement $y[k]$ is considered, which is

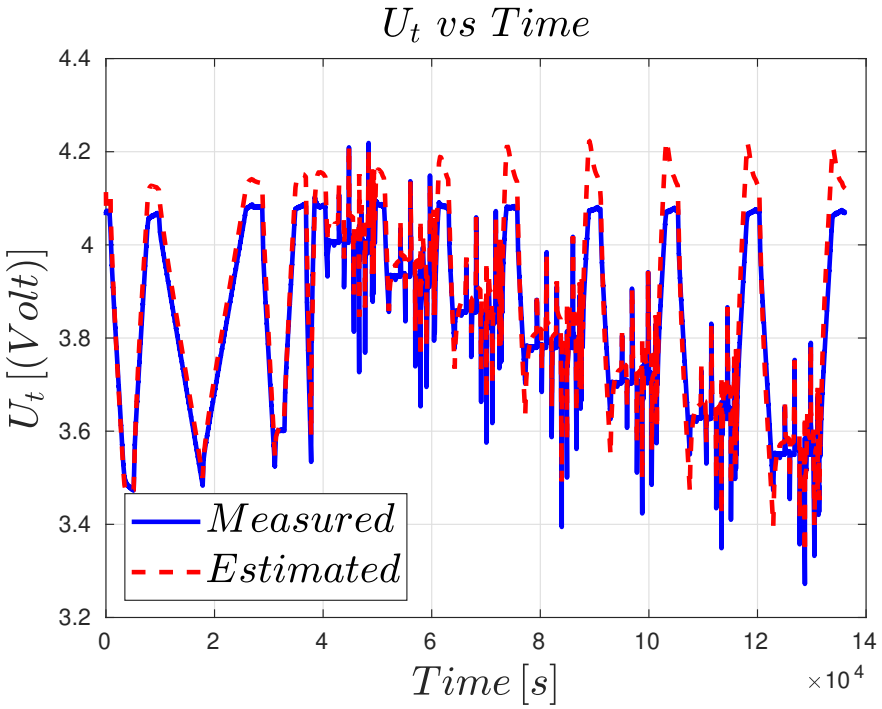

Figure 10: Validation results for terminal voltage, U_t, by using AEKF

called the a priori estimate, and $\hat{\boldsymbol{x}}[k]^+$ is the estimate of $\boldsymbol{x}[k]$ after the measurement $y[k]$ is taken into account, which is called the a posteriori estimate. The voltage error $e[k]$ is computed and the adaptive law $\boldsymbol{H}[k]$ is employed to update $\hat{\boldsymbol{x}}[k]$, $\boldsymbol{P}[k]$, $\boldsymbol{Q}[k]$ and $\boldsymbol{K}[k]$. Then, the updated gain is used to compensate for the state estimation error. The SOC estimation is fed back to update the parameters of the battery model for the SOC estimation at the next sampling time. Implementation of the AEKF is at Appendix,. In the simulation study, *Adaptive Law* is skipped due to the high values coming from $\boldsymbol{C}[k]\boldsymbol{P}[k]^-\boldsymbol{C}[k]^T$. So that, $R[k]$ taken as constant during the simulation. The following figures show the results of the AEKF, actually EKF, technique.

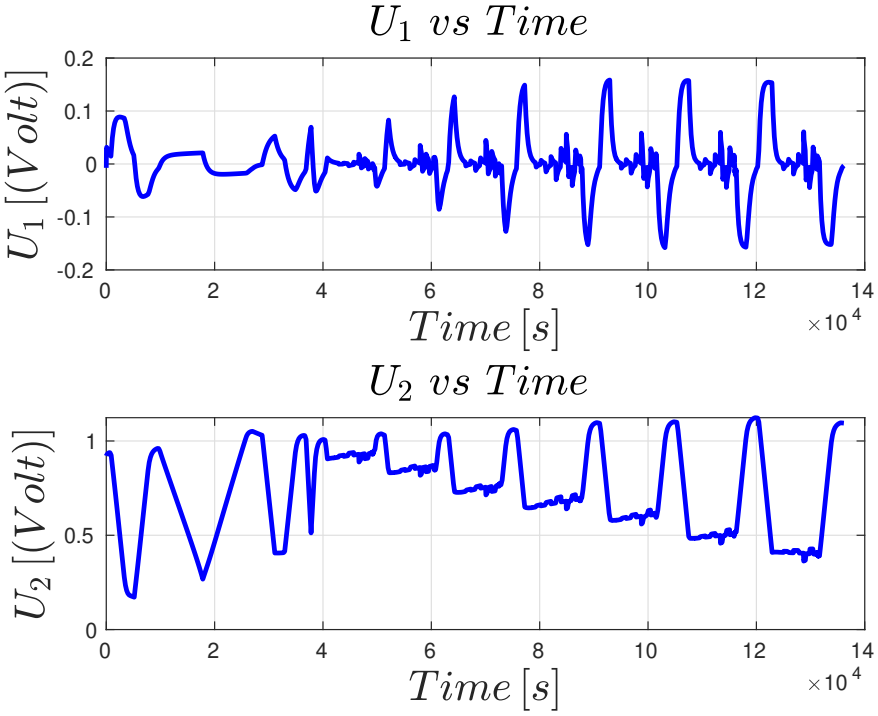

Figure 11: AEKF's state estimation for RC voltages U_1 and U_2, respectively

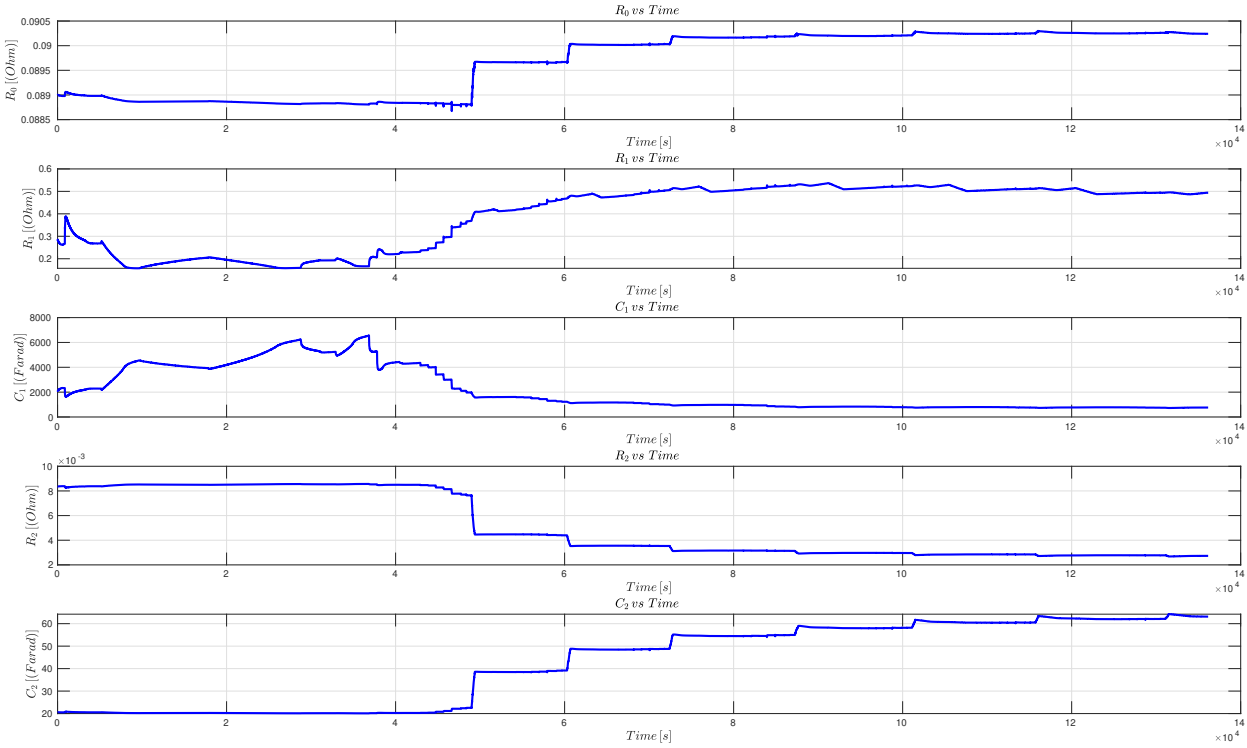

Figure 12: Estimated Parameters by using RLS through whole experiment time.

References

[1] M. Dubarry, V. Svoboda, R. Hwu, and B. Y. Liaw. Capacity loss in rechargeable lithium cells during cycle life testing: The importance of determining state-of-charge. Journal of Power Sources, 174(2):1121–1125, 2007. 13th International Meeting on Lithium Batteries.

[2] H. He, X. Zhang, R. Xiong, Y. Xu, and H. Guo. Online model-based estimation of state-of-charge and open-circuit voltage of lithium-ion batteries in electric vehicles. Energy, 39(1):310–318, 2012. Sustainable Energy and Environmental Protection 2010.

[3] D. E. Neumann and S. Lichte. A multi-dimensional battery discharge model with thermal feedback applied to a lithium-ion battery pack. 2011 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM, 2011.

[4] H. Pang and F. Zhang. Experimental data-driven parameter identification and state of charge estimation for a li-ion battery equivalent circuit model. Energies, 11(5), 2018.

[5] G. Plett. Battery Management Systems. Number v. 1 in Artech House power engineering and power electronics. Artech House, 2015.

[6] G. L. Plett. Extended kalman filtering for battery management systems of lipb-based hev battery packs: Part 2. modeling and identification. Journal of Power Sources, 134(2):262–276, 2004.

[7] X. Sun, J. Ji, B. Ren, C. Xie, and D. Yan. Adaptive forgetting factor recursive least square algorithm for online identification of equivalent circuit model parameters of a lithium-ion battery. Energies, 12(12), 2019.

[8] W. Waag, C. Fleischer, and D. U. Sauer. Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles. Journal of Power Sources, 258:321–339, 2014.

[9] F. Wen, B. Duan, C. Zhang, R. Zhu, Y. Shang, and J. Zhang. High-accuracy parameter identification method for equivalent-circuit models of lithium-ion batteries based on the stochastic theory response reconstruction. Electronics, 8(8), 2019.

[10] R. Xiong, X. Gong, C. C. Mi, and F. Sun. A robust state-of-charge estimator for multiple types of lithium-ion batteries using adaptive extended kalman filter. Journal of Power Sources, 243:805–816, 2013.

[11] C. Zhang, J. Jiang, L. Zhang, S. Liu, L. Wang, and P. C. Loh. A generalized soc-ocv model for lithium-ion batteries and the soc estimation for lnmco battery. Energies, 9(11), 2016.

Appendices

A Curve Fitting for OCV Models

A.1 Model-1

The first proposed generalized SOC-OCV model is shown in Equation 29, where a logarithmic function with real (not complex) power, a linear function, and an exponential function with a shifted exponent can clearly be seen:

$$U_{OC}(SOC(t)) = a + b \times (-ln(SOC(t)))^m + c \times SOC(t) + d \times \exp^{n(SOC(t)-1)} \tag{29}$$

where U_{OC} and s represent the OCV and SOC of the battery, respectively, and $0 \leq s \leq 1$, $m > 0$ and $n > 0$. The logarithmic function must be tuned to play a predominant role at low SOC, where charge accumulations on the surfaces of the active materials happen within the lithium-ion batteries. The linear function, in turn, dominates the middle SOC range, where primary phase transformation of the active materials occurs. The last exponential function then contributes to the high SOC behavior, where both partial redox reaction and charge accumulation occur. The three functions in Equation 29 are therefore essential, and will interact with each other to form the generalized OCV model over the whole SOC range [11].

Parameters of the function is found via **NonlinearLeastSquare** method and *Trust-Region* algorithm within the boundary of the m and n values. Coefficients is calculated with 95%

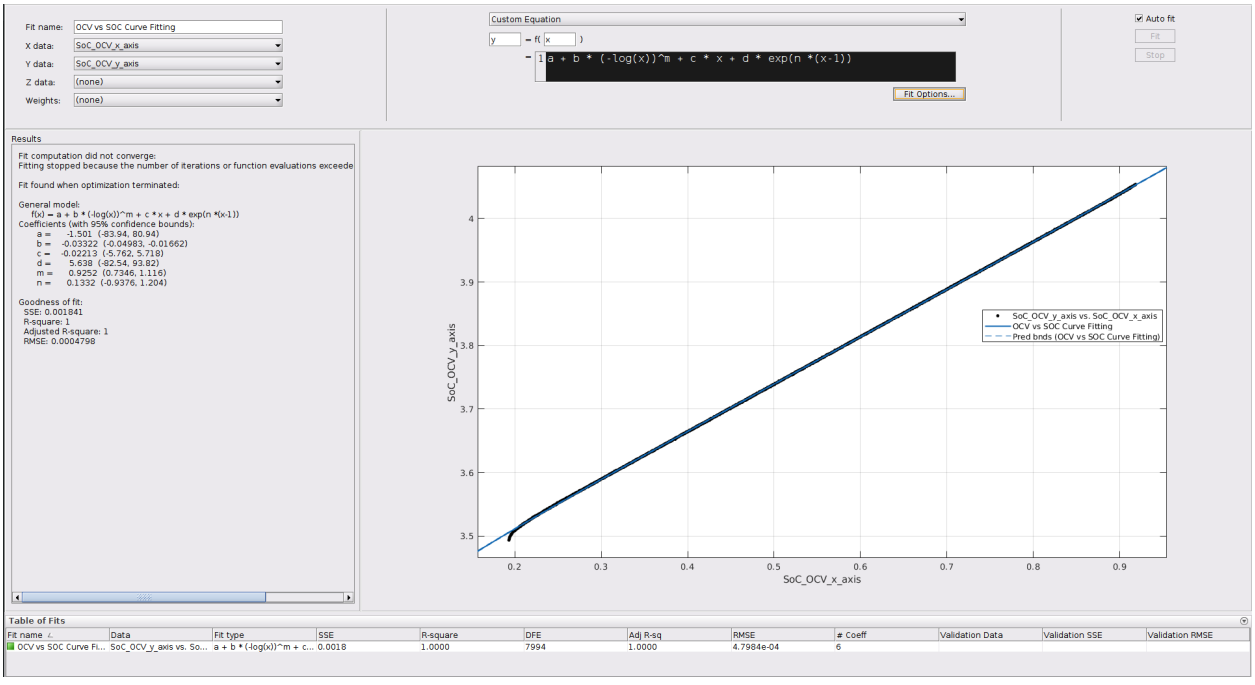

Figure 13: MATLAB Curve Fitting Toolbox for Model - 1

Coefficient	Value with 95% Confidence Bounds	Lower Bound	Upper Bound
a	-1.501	$-\infty$	∞
b	-0.03322	$-\infty$	∞
c	-0.02213	$-\infty$	∞
d	5.638	$-\infty$	∞
m	0.9252	0	∞
n	0.1332	0	∞

Table 2: Coefficients of the model-1 with bounds.

SSE	RMSE	R-Square	Adjusted R-sq
0.001841	0.0004798	1	1

Table 3: The Goodness of fit of the model-1.

confidence bounds. Table 2 shows the coefficient values and their limits and next Table 3 includes the sum of the squared prediction errors (SSE), the root mean square error (RMSE), R-square (i.e. it is a statistical measure of how close the data are to the fitted regression line), and adjusted R-sq.

A.2 Model-2

With SOC available as part of the model state, terminal voltage may be predicted in a number of different ways. Several different forms are adapted from the literature [6].

Shepherd model:	$U_t(t) = U_{OC}(SOC(t)) - R_0 I(t) - K_i/SOC(t)$
Unnewehr universal model:	$U_t(t) = U_{OC}(SOC(t)) - R_0 I(t) - K_i/SOC(t)$
Nernst model:	$U_t(t) = U_{OC}(SOC(t)) - R_0 I(t) + K_2 ln(SOC(t))$ $+K_3 ln(1 - SOC(t))$

K_i is the polarization resistance and K_1, K_2 and K_3 are constants chosen to make the model fit the data well. All of the terms of these models may be collected to make a "combined model" that performs better than any of the individual models alone. This model is

$$U_t(t) = K_0 - R_0 I(t) - \frac{K_1}{SOC(t)} - K_2 SOC(t) + K_3 ln(SOC(t)) + K_4 ln(1 - SOC(t)) \quad (30)$$

The unknown quantities in the combined model may be estimated using a system identification procedure. This model has the advantage of being *linear in the parameters*; that is, the unknowns occur linearly in the output equation. From the Equation 30 nonlinear OCV-SOC model can be extracted as follows:

$$U_{OC}(SOC(t)) = K_0 - \frac{K_1}{SOC(t)} - K_2 SOC(t) + K_3 ln(SOC(t)) + K_4 ln(1 - SOC(t)) \quad (31)$$

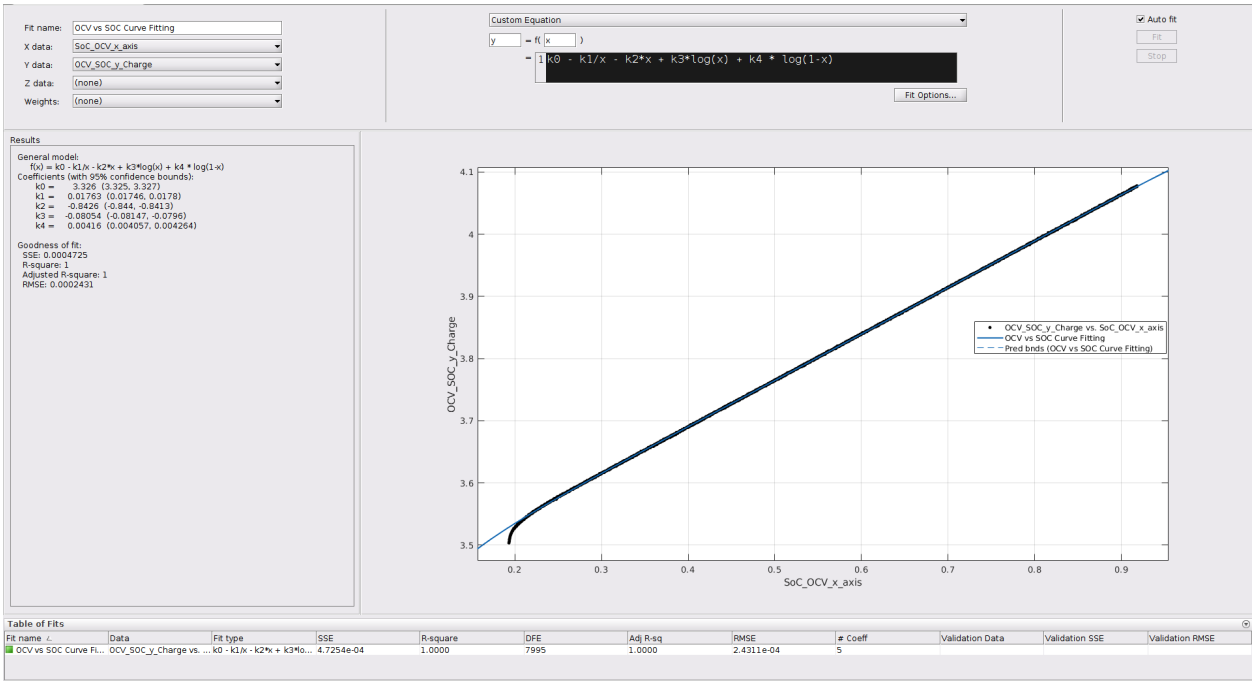

Figure 14: MATLAB Curve Fitting Toolbox for Model - 2

Parameters of the above function, 31, is found via **NonlinearLeastSquare** method and *Trust-Region* algorithm within the pretty large boundary. Coefficients is calculated with 95% confidence bounds. Table 2 shows the coefficient values and their limits and next Table 3 includes the sum of the squared prediction errors (SSE), the root mean square error (RMSE), R-square (i.e. it is a statistical measure of how close the data are to the fitted regression line), and adjusted R-sq.

Coefficient	Value with 95% Confidence Bounds	Lower Bound	Upper Bound
K_0	3.326	$-\infty$	∞
K_1	0.01763	$-\infty$	∞
K_2	-0.8426	$-\infty$	∞
K_3	-0.08054	$-\infty$	∞
K_4	0.00416	$-\infty$	∞

Table 4: Coefficients of the model-2 with bounds.

SSE	RMSE	R-Square	Adjusted R-sq
0.0004725	0.0002431	1	1

Table 5: The Goodness of fit of the model-2.

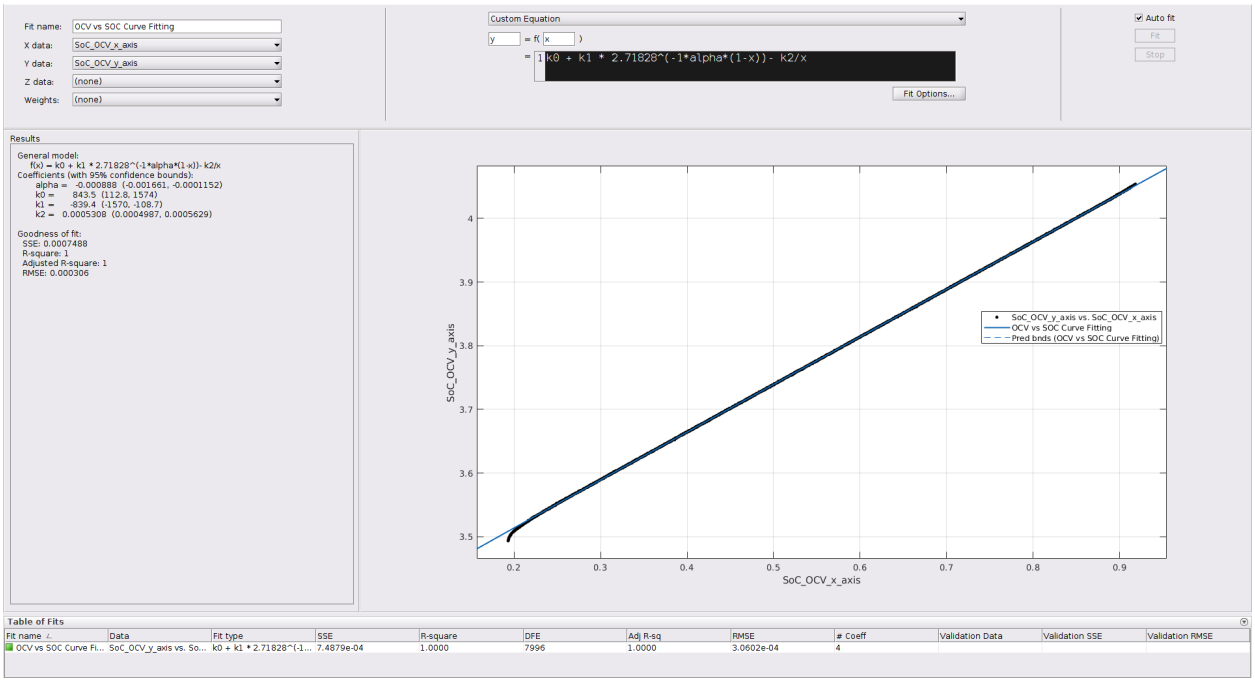

Figure 15: MATLAB Curve Fitting Toolbox for Model - 3

A.3 Model-3

It can be understood that the last model, Model-3 is somehow a simplified version of Model-1 A.1. Usage of this model can be found in a few studies [3]. In this model number of parameters are reduced to four. OCV-SOC relation in the model is indicated at the following equation:

$$U_{OC}(SOC(t)) = K_0 + K_1 \exp^{-\alpha(1-SOC(t))} - \frac{K_2}{SOC(t)} \tag{32}$$

Parameters of the above function, 32, is found via **NonlinearLeastSquare** method and *Trust-Region* algorithm within the pretty large boundary. Coefficients is calculated with 95% confidence bounds. Value of irrational number e, also represented as exp is taken as 2.71828 in order to obtain an approximate solution. Table 6 shows the coefficient values and their limits and next Table 7 includes the sum of the squared prediction errors (SSE), the root mean square error (RMSE), R-square (i.e. it is a statistical measure of how close the data are to the fitted regression line), and adjusted R-sq.

Coefficient	Value with 95% Confidence Bounds	Lower Bound	Upper Bound
α	-0.000888	$-\infty$	∞
K_0	843.5	$-\infty$	∞
K_1	-839.4	$-\infty$	∞
K_2	0.0005308	$-\infty$	∞

Table 6: Coefficients of the model-3 with bounds.

SSE	RMSE	R-Square	Adjusted R-sq
0.0007488	0.000306	1	1

Table 7: The Goodness of fit of the model-3.

B Determination of Nominal Capacity, Q

```
load('BAT_data.mat');

%% Find the Charge, Discharge and Idle States
 % Add a state of the battery

 i = -1 * i;

SoC(:,2) = 0; %% add initially all state is zero

for time_index = 1:length(t_vec)-1

    if( i(time_index) > 0 ) % battery is discharging
        SoC(time_index, 2 ) = -1;

    elseif( i(time_index) < 0 ) % battery is charging
        SoC(time_index, 2 ) = 1;

    else
         SoC(time_index, 2 ) = 0; % battery is iddle
    end

end
%%

%% Determine First Discharge Region and Find Charge Capacity as Ah
select_first_discharge = [];

for time_index = 2:length(t_vec)-1

    if SoC(time_index,2) == -1
        % Calculate cell nominal capacity
        % Coulombic efficiency (eta) taken as one for discharging

        eta = 1;
        delta_t = t_vec(time_index) - t_vec(time_index-1);

        nominal_capacity = (i(time_index-1) * eta * delta_t ) /
```

```
                ...
38                      (0.01 * (SoC(time_index-1,1) - SoC(time_index,1))
                           );
39
40          select_first_discharge = [select_first_discharge; ...
41                time_index i(time_index) SoC(time_index,1)
                     nominal_capacity];
42      end
43
44      if SoC(time_index-1,2) == -1 && SoC(time_index,2) ~= -1
45          break
46      end
47
48  end
49
50  % Q => 1129.4 Ampere-Second
51  % Q => 313.7 mAh | 0.3137 Ah
52  %%
```

C The Relation Between RLS Parameters and Equivalent Circuit Parameters

Equation 15 is rewritten in explicit form as follows:

$$G(s) = -\frac{R_0 s^2 + \dfrac{R_0R_1C_1 + R_0R_2C_2 + R_2R_1C_1 + R_1R_2C_2}{R_1C_1R_2C_2}s + \dfrac{R_0 + R_1 + R_2}{R_1C_1R_2C_2}}{s^2 + \dfrac{R_1C_1 + R_2C_2}{R_1C_1R_2C_2}s + \dfrac{1}{R_1C_1R_2C_2}} \tag{33}$$

Using the bilinear transformation equation that is already given at 16, parameters of the 17 can be formulated as:

$$\begin{aligned}
\theta_1 &= \frac{2T^2 - 8R_1C_1R_2C_2}{-T^2 - 2T(R_1C_1 + R_2C_2) - 4R_1C_1R_2C_2} \\
\theta_2 &= \frac{T^2 - 2T(R_1C_1 + R_2C_2) + 4R_1C_1R_2C_2}{-T^2 - 2T(R_1C_1 + R_2C_2) - 4R_1C_1R_2C_2} \\
\theta_3 &= \frac{T^2(R_0 + R_1 + R_2) + 2T(R_0R_1C_1 + R_0R_2C_2 + R_1R_2C_2 + R_2R_1C_1) + 4R_0R_1C_1R_2C_2}{-T^2 - 2T(R_1C_1 + R_2C_2) - 4R_1C_1R_2C_2} \\
\theta_4 &= \frac{2T^2(R_0 + R_1 + R_2) - 8R_0R_1C_1R_2C_2}{-T^2 - 2T(R_1C_1 + R_2C_2) - 4R_1C_1R_2C_2} \\
\theta_5 &= \frac{T^2(R_0 + R_1 + R_2) - 2T(R_0R_1C_1 + R_0R_2C_2 + R_1R_2C_2 + R_2R_1C_1) + 4R_0R_1C_1R_2C_2}{-T^2 - 2T(R_1C_1 + R_2C_2) - 4R_1C_1R_2C_2}
\end{aligned} \tag{34}$$

The terms T comes from the bi-linear transformation. Above equation, 34, may be simply rewritten with few terms. Suppose that $a = R_0$, $b = R_1C_1R_2C_2$, $c = R_1C_1 + R_2C_2$, $d = R_0 + R_1 + R_2$, and $f = R_0R_1C_1 + R_0R_2C_2 + R_1R_2C_2 + R_2R_1C_1$ are function of RC circuit elements. [**Note**:Notation e is not chosen intentionally due to the fact that it is denoted for irrational Euler's number.] [7]

$$\begin{aligned}
\theta_1 =& \frac{-2T^2 + 8b}{T^2 + 2cT + 4b} \\
\theta_2 =& \frac{4cT}{T^2 + 2cT + 4b} - 1 \\
\theta_3 =& -\frac{dT^2 + 2fT + 4ab}{T^2 + 2cT + 4b} \\
\theta_4 =& \frac{-2dT^2 + 8ab}{T^2 + 2cT + 4b} \\
\theta_5 =& -\frac{dT^2 - 2fT + 4ab}{T^2 + 2cT + 4b}
\end{aligned} \tag{35}$$

Therefore, previously defined a, b, c, d, and f terms can be written in the form of parameters of the RLS estimation technique:

$$\begin{aligned}
a =& \frac{\theta_4 - \theta_3 - \theta_5}{1 + \theta_1 - \theta_2} \\
b =& \frac{T^2(1 + \theta_1 - \theta_2)}{4(1 - \theta_1 - \theta_2)} \\
c =& \frac{T(1 + \theta_2)}{1 - \theta_1 - \theta_2} \\
d =& \frac{-\theta_3 - \theta_4 - \theta_5}{1 - \theta_1 - \theta_2} \\
f =& \frac{T(\theta_5 - \theta_3)}{1 - \theta_1 - \theta_2}
\end{aligned} \tag{36}$$

Time constants of the circuit are the form of $\tau_1 = \dfrac{c + \sqrt{c^2 - 4b}}{2}$ and $\tau_2 = \dfrac{c - \sqrt{c^2 - 4b}}{2}$. Finally, the resistance and capacitance parameters can be formulated as:

$$\begin{aligned}
R_0 =& a \\
R_1 =& \frac{\tau_1(d - a) + ac - f}{\tau_1 - \tau_2} \\
R_2 =& d - a - R_1 \\
C_1 =& \frac{\tau_1}{R_1} \\
C_2 =& \frac{\tau_2}{R_2}
\end{aligned} \tag{37}$$

5.1 Equivalent Circuit Values to Recursive Least Square Parameters

```
function RLS_Parameters = RC_Values_to_RLS_Parameters(RC_Values)

    % Sampling time of the system is T = 1
    T = 1;  % Take this value as an function argument when
       sampling time
            % is changed


    % a = R_0;
    % b = R_1 * C_1 *R_2 * C*2;
    % c = R_1 * C_1 + R_2 * C_2;
    % d = R_0 + R_1 + R_2;
    % f = R_0 * R_1 *C_1 + R_0 * R_2 * C_2 + R_1 * R_2 * C_2 + R_2
        * R_1 *C_1;

    a = RC_Values(1);
    b = RC_Values(2) * RC_Values(3) * RC_Values(4) * RC_Values(5);
    c = RC_Values(2) * RC_Values(3) + RC_Values(4) * RC_Values(5);
    d = RC_Values(1) + RC_Values(2) + RC_Values(4);
    f = RC_Values(1) * RC_Values(2) * RC_Values(3) + ...
        RC_Values(1) * RC_Values(4) * RC_Values(5) + ...
        RC_Values(2) * RC_Values(4) * RC_Values(5) + ...
        RC_Values(4) * RC_Values(2) * RC_Values(3);

   theta_1 = (8*b - 2*T^(2))/(4*b + 2*c*T + T^(2));
   theta_2 = (4*c*T)/(4*b + 2*c*T + T^(2)) - 1;
   theta_3 = -1*(4*a*b + 2*f*T + d*T^(2))/(4*b + 2*c*T + T^(2));
   theta_4 = (8*a*b - 2*d*T^(2))/(4*b + 2*c*T + T^(2));
   theta_5 = -1*(4*a*b - 2*f*T + d*T^(2))/(4*b + 2*c*T + T^(2));

  %% Return this value as Recursive Least Square Parameters
   RLS_Parameters = [theta_1; theta_2; theta_3; theta_4; theta_5];

end
```

5.2 Recursive Least Square Parameters to Equivalent Circuit Values

```
function RC_Values = RLS_Parameters_to_RC_Values(theta_vector)

    % Sampling time of the system is T = 1
    T = 1;  % Take this value as an function argument when
```

```
        sampling time
5           % is changed
6
7       a = (theta_vector(4) - theta_vector(3) - theta_vector(5)) /
            ...
8               (1 + theta_vector(1) - theta_vector(2) );
9
10      b = (T^(2) * (1 + theta_vector(1) - theta_vector(2))) / ...
11              (4 * (1 - theta_vector(1) - theta_vector(2)));
12
13      c = (T * (1 + theta_vector(2))) / ...
14              (1 - theta_vector(1) - theta_vector(2));
15
16      d = (-1 * theta_vector(3) - theta_vector(4) - theta_vector(5))
            / ...
17              (1 - theta_vector(1) - theta_vector(2));
18
19      f = (T * (theta_vector(5) - theta_vector(3))) / ...
20              (1 - theta_vector(1) - theta_vector(2));
21
22      % Time constant of the circuit
23      tau_1 = (c + sqrt(c^2 - 4*b))/2;
24
25      tau_2 = (c - sqrt(c^2 - 4*b))/2;
26
27      R_0 = a;
28
29      R_1 = (tau_1 * (d - a) + a * c - f) / (tau_1 - tau_2);
30
31      C_1 = tau_1 / R_1;
32
33      R_2 = d - a - R_1;
34
35      C_2 = tau_2 / R_2;
36
37
38      % Return this value as RC Values at estimation time
39      RC_Values = [R_0; R_1; C_1; R_2; C_2];
40
41  end
```

6 MATLAB Code for State and Parameter Estimation

```
1   clc;
2   clear all
```

```
close all;

load('BAT_data.mat');

%% Find the Charge, Discharge and Idle States
 % Add a state of the battery
i = -1 * i;

SoC(:,2) = 0; %% add initially all state is zero

for time_index = 1:length(t_vec)-1

    if( i(time_index) > 0 ) % battery is discharging
        SoC(time_index, 2 ) = -1;

    elseif( i(time_index) < 0 ) % battery is charging
        SoC(time_index, 2 ) = 1;

    else
         SoC(time_index, 2 ) = 0; % battery is iddle
    end

end
%%

%% Adaptive Extended Kalman Filter Algorithm with Recursive Least
   Square

%Inital guess of circuit values comes from previous RLS
R_0 = 0.089; R_1 = 0.2802; C_1 = 2.135e+03;
R_2 = 0.00836; C_2 = 20.61;

RC_Values = [R_0; R_1; C_1; R_2; C_2];

% Get the initial parameter vector Look Eqn (25)
theta_vector = RC_Values_to_RLS_Parameters(RC_Values);

% Check the following function to calculcation are valid or not!
%RLS_Parameters_to_RC_Values(theta_vector);

delta = 0.01;
P_RLS = delta * eye(5);

% RC Vector
RC_Vector = [];
```

```
47
48 % Initially take U1 and U2 same and SOC from measurement
49 SOC_init = 0.01 * SoC(1,1);
50 U_1 = (u(1) - OCV_SOC_Function(SoC(1,1)) - R_0 * i(1)) /2;
51 U_2 = U_1; % Indeed, U1 is not equal to U2, this is just
      initiation
52 x_prior_1 = [U_1; U_2; SOC_init];
53
54 % Initial State Covariance Matrix
55 P = [0 0 0; 0 0 0; 0 0 0];
56
57 % Initial process noise input with covariance
58 Q = [0.1 0 0; 0 0.1 0; 0 0 0.01];
59
60 % Coulumb efficiency
61 eta = 1;
62
63 % Total Capacity
64 Total_Capacity = 1129.4; %Ampere-Second
65
66 % Number of Residual for Adaptive Law
67 M = 5;
68
69 %Adaptive Law H
70 H = 0;
71
72 State_Log = []; % State_Log_Vector
73 Simulation_Output_Log = []; % Output With Simulation
74 Output_Error = []; % Terminal Voltage Error
75
76 % T represents the sampling time
77 T = 1;
78 for index = 1:1:length(i)
79
80     %  Recursive Least Sqaure estimates Circuit Parameters
81     if(index < 3)
82
83         RC_Vector = [RC_Vector; index RC_Values'];
84
85     else
86
87         y_k_1 = u(index-1) - OCV_SOC_Function(SoC(index-1,1));
88
89         y_k_2 = u(index-2) - OCV_SOC_Function(SoC(index-2,1));
90
```

```
91          I_k = i(index);
92
93          I_k_1 = i(index-1);
94
95          I_k_2 = i(index-2);
96
97          phi_vector = [y_k_1; y_k_2; I_k; I_k_1; I_k_2];
98
99
100         % Calculate gain vector
101         K_RLS = P_RLS * phi_vector * ...
102                         (phi_vector' * P_RLS * phi_vector + 1)
                              ^(-1);
103
104         % Update estimation error covariance
105         P_RLS = (eye(5) - K_RLS * phi_vector') * P_RLS;
106
107         % Update the recursive least square paramater vector
108         y_k = u(index) - OCV_SOC_Function(SoC(index,1));
109
110         theta_vector = theta_vector + ...
111                               K_RLS * (y_k - phi_vector' *
                                    theta_vector);
112
113         RC_Values = RLS_Parameters_to_RC_Values(theta_vector);
114         RC_Vector = [RC_Vector; index RC_Values'];
115     end
116     %
117
118 tau_1 = RC_Values(2)*RC_Values(3);
119 tau_2 = RC_Values(4)*RC_Values(5);
120
121 A_d = [exp(-T/tau_1) 0 0; ...
122        0 exp(-T/tau_2) 0; ...
123        0 0 1];
124
125 if( i(index) < 0 ) %charging
126     eta = 0.95;
127 else
128    eta = 1;         %discharging or zero terminal current
129 end
130
131
132 B_d = [RC_Values(2)*(1-exp(-T/tau_1)); ...
133        RC_Values(4)*(1-exp(-T/tau_2)); ...
```

```
134         -(eta*T)/Total_Capacity];
135
136 %State Estimation Propagation
137 x_priori = A_d * x_prior_1 + B_d * i(index);
138
139 %State Estimation Covariance
140 P = A_d * P * A_d' + Q;
141
142 %Error Innovation
143
144 y_k_sim = OCV_SOC_Function(SoC(index,1)) - x_priori(1) - x_priori
      (2) - ...
145                                                 RC_Values(1) * i(index
                                                      );
146
147 terminal_voltage_error = u(index) - y_k_sim;
148
149
150 Output_Error = [Output_Error; index terminal_voltage_error];
151
152 if( length(Output_Error) < M + 1 )
153
154     for j = 1:1:length(Output_Error(:,2))
155         H = H + (1/M)*(Output_Error(j,2))^2;
156     end
157
158 else
159
160     for j = (length(Output_Error(:,2))- M + 1):1:length(
        Output_Error(:,2))
161         H = H + (1/M)*(Output_Error(j,2))^2;
162     end
163
164 end
165
166 % Matrix C
167 C_d = [-1 -1 Derivative_Of_OCV_SOC_Function(SoC(index,1))];
168
169 %measurements with noise covariance
170 %R = H - C_d * P * C_d';
171 R = 0.8;
172
173 % Kalman Gain Matrix
174 K = P * C_d' * (C_d * P * C_d' + R)^(-1);
175
```

```
176 %State Estimate Measurement Update
177 x_posteriori = x_priori + K * terminal_voltage_error;
178
179 %State Covariance Measurement Update
180 Q = K*H*K';
181
182 P = (eye(3) - K * C_d) * P * (eye(3) - K * C_d)' + K * R * K';
183
184 x_prior_1 = x_posteriori;
185
186 y_k_sim = OCV_SOC_Function(SoC(index,1)) - ...
187             x_posteriori(1) - x_posteriori(2) - RC_Values(1) * i(
                index);
188
189 Simulation_Output_Log = [Simulation_Output_Log; index y_k_sim];
190
191 State_Log = [State_Log; index x_posteriori'];
192
193 % At the end assign to residul value zero for new calculation
194 H = 0;
195 end
196 %%
```