CHALMERS
UNIVERSITY OF TECHNOLOGY

On-line Adaptive Dual Estimation of Li-Ion Battery State and Parameters in Electric Vehicles

Master's Thesis in Engineering Mathematics and Computational Science

Jacob Klintberg

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis EX067/2018

On-line Adaptive Dual Estimation of Li-Ion Battery State and Parameters in Electric Vehicles

JACOB KLINTBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automatic Control
Chalmers University of Technology
Gothenburg, Sweden 2018

On-line Adaptive Dual Estimation of Li-Ion Battery
State and Parameters in Electric Vehicles
JACOB KLINTBERG

© JACOB KLINTBERG, 2018.

Supervisors:
Christian Fleischer, Volvo Cars Corporation
Björn Fridholm, Volvo Cars Corporation / Department of Electrical Engineering
Anton Klintberg, Department of Electrical Engineering
Examiner:
Torsten Wik, Department of Electrical Engineering

Master's Thesis EX067/2018
Department of Electrical Engineering
Division of Systems and Control
Automatic Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Volvo XC90.

Typeset in LaTeX
Gothenburg, Sweden 2018

On-line Adaptive Dual Estimation of Li-Ion Battery
State and Parameters in Electric Vehicles
JACOB KLINTBERG
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In recent years there has been an increased concern about the climate change which has lead to tougher legislation on emissions for the automotive industry. This, together with rising oil prices, has made many car manufacturers to develop electrified vehicles. A key part in this transition is the battery, and the systems that are observing and controlling the internal processes in the battery. A vital part of those systems is to estimate the internal states and parameters of the battery cells. However, in electrified vehicles a battery may consist of several hundreds of cells, and it can therefore be expensive, both financially and computationally, to perform the estimation on all cells. To save money on hardware, this work investigates how the algorithms should be implemented in a computationally efficient way. The states are estimated with an extended Kalman filter, and the different parameters are estimated with a recursive least square algorithm and various Kalman filters. Structures and symmetries in matrices and re-usage of estimation gains for series connected cells are utilized to reduce the computational cost. The results shows that the estimates converges when the states and parameters are simultaneous estimated, and that the computational cost of the algorithms can be significantly reduced.

Keywords: Battery management systems, BMS, equivalent circuit models, Kalman filtering, State of Charge, computational efficient implementation.

Acknowledgements

First and dearest, I would like to thank Anton Klintberg for many good ideas and discussions, this thesis would not have been equally good without your help. I would also like to thank Christian Fleischer and Björn Fridholm for your friendly attitudes and good ideas. Regardless of your high workload you have taken time to answer my questions, which is highly appreciated. A thank you should also be addressed to my examiner Torsten Wik for letting me do this project even though I am from another master program.

I would also like to thank the people in the Automatic control group at Chalmers and the people in the traction battery group at Volvo, for an friendly attitude, many funny discussions and great tasting fika.

Eventually, I would also like to thank my family for supporting me, not only during this work.

Jacob Klintberg, Gothenburg, April 2018

Contents

1
Introduction

1.1 Background

Increased concern about the climate change has lead to tougher legislation on emissions for the automotive industry. This, together with rising oil prices, have made many car manufactures to develop electrified vehicles. A bottleneck in this transition is the performance of the battery. The performance of the battery is, however, not necessarily just dependent on the battery itself but also on systems that are controlling and observing the internal processes in the battery, often referred to as the Battery Management Systems (BMS). A vital part of those systems is to estimate the internal states and parameters of the battery cells. However, a battery pack in an electrified vehicle may consist of several hundred of cells, and it can therefore be expensive, both financially and computationally, to perform the estimation on all cells. Therefore, to save money on hardware, it is desirable to implement the algorithms in a computationally efficient manner.

1.2 Motivation

In battery applications it is important to have information about internal states of the battery. One of those is State of Charge (SoC), which can be compared to the fuel level in a conventional vehicle with a combustion engine. SoC, however, is not only an indicator for when the battery should be refilled with energy, it is also critical for safety reasons since Lithium-ion batteries can explode if they are overcharged. Moreover, SoC is extensively used in other algorithms, and its estimate is therefore an essential part of the BMS.

The observers are often based on equivalent circuit models that mimics the current-voltage behavior of the cell [1]. The drawback is, however, that the parameters in those models have no explicit connection to the physical reactions inside the cell, making the models only valid around the current operation point. Therefore, to maintain the accuracy, it is often desirable to continuously estimate the parameters in the models.

SoC estimation has been rigorous investigated in the literature and many different type of observers has been tested, for instance, the Extended Kalman filter (EKF) [2, 3, 4], the Unscented Kalman filter [5], Luenberger Observer [6], Sliding Mode Observer [7] and H_∞ Observer [8].

Moreover, observers that estimate the parameters in equivalent circuit models have also been investigated in the literature. For instance, in [1] and [9], the estimation of the ohmic resistance is separated from the estimator for the time constant. The ohmic resistance is then estimated with recursive least square algorithms, and a Kalman filter is used to estimate the time constant. In [10] it is proposed to use particle filters for the parameter estimation. Furthermore, in [11, 12, 13], it is suggested to use various Kalman filters for capacity estimation.

Most of the existing research is, however, focused on algorithms developed for high accuracy on cell level. In an electrified vehicle a battery pack may consist of several hundreds of cells, making it, both computationally and financially, expensive to implement these algorithms on all the cells. This is a motivation for also considering the computational cost of the algorithms.

There are some previous work that takes computational efficiency into account. One approach is to view the battery pack as "one big cell", and then estimate the states on pack level [14]. Even though this approach is computationally efficient, there is a major drawback. Many cells in a battery pack are connected in series, which means that if one cell is fully (dis)charged it is impossible to (dis)charge any of the other cells even though they have capacity available. Hence, only a few cells are determining the usable capacity of the entire pack, and this information is not obtained using this method. A variant of this method is to group cells that are similar, and do the estimation on group level instead of pack level [15]. This method alleviates the drawback, but does not eliminate it.

Some research has also been focusing on obtaining methods which can obtain SoC estimates for all of the cells in the battery pack. It is, for instance, proposed that SoC could be estimated for one reference cell, and then calculate how much other cells deviate from this reference cell [16, 17]. In this work we will use another approach. We will use methods that are developed for high accuracy on cell level, and investigate how these can be computationally efficient implemented on multiple cells in a battery pack.

1.3 Aim

The aim of this work was to evaluate dual estimation of the states and the parameters in equivalent circuit models of battery cells. Moreover, the computational cost of the algorithms when implemented on multiply battery cells has been investigated.

1.4 Outline of the thesis

The rest of this thesis is organized as follows. In Chapter 2 an equivalent circuit of the battery cell is presented. Offline identification and online state and parameter estimation are presented in Chapter 3 and 4, respectively. Furthermore, in

Chapter 5 it is proposed how the estimation algorithms can be implemented in a computational efficient manner for multiple battery cells. This work is evaluated in a simulation study that is based on laboratory data in Chapter 6, and in Chapter 7 some conclusions and suggested topics for future work are presented.

2

Modelling of Lithium-Ion Batteries

Battery models can be divided into two categories; electrochemical models and equivalent circuits models. In online applications the equivalent circuit models are often the preferred choice, since they are less computationally heavy to run compared to the electrochemical models [18].

2.1 Equivalent circuit models

The remaining capacity, $Q(t)$, is the number of ampere-hours that can be drawn from a battery cell until it is fully discharged. The nominal capacity, $Q_{nom}(t)$, is the number of ampere-hours that can be drawn from a fully charged cell until it is fully discharged. Using these, we define State of Charge (SoC) as the ratio between the remaining capacity and the nominal capacity of the cell, i.e.,

$$z_{\text{SoC}}(t) = \frac{Q(t)}{Q_{nom}(t)} \tag{2.1}$$

The most straightforward way to obtain SoC is to integrate the current through the cell, i.e.,

$$z_{SoC}(t) = z_{SoC}(t_0) + \frac{\eta_i}{Q_{nom}(t)} \int_{t_0}^{t} i_{cell}(\tau) d\tau, \tag{2.2}$$

where η_i is Columbic efficiency, and t_0 is the initial time of the integration.

By measuring the open circuit voltage (OCV) for different SoC, a curve relating those quantities can be obtained. This curve is an important part of equivalent circuit models, and is often referred to as the OCV-curve.

The dynamical effects of the cell are often captured by a series resistor and two parallel resistor-capacitor networks, as shown in Figure 2.1.

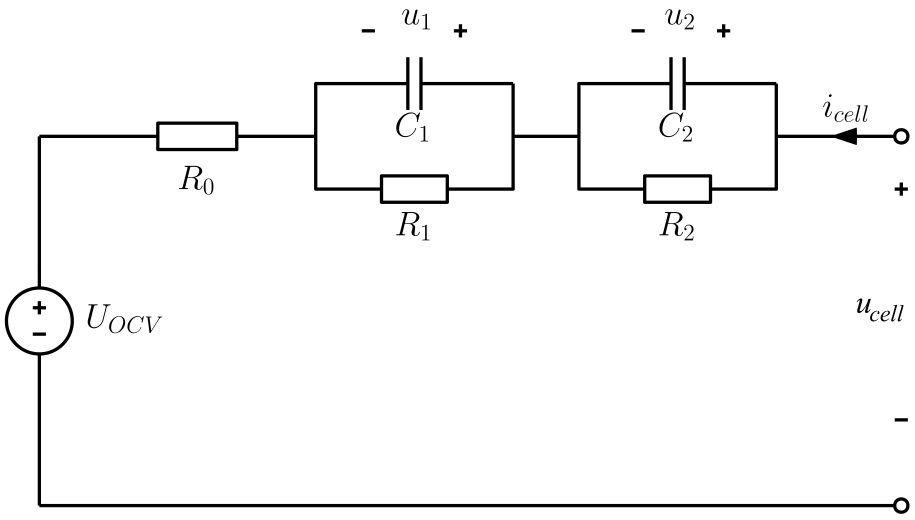

Figure 2.1: Equivalent circuit model with two RC-links

From the circuit in Figure 2.1, (2.2) and the OCV-curve, one of the most commonly used equivalent circuit models are obtained. By using a zero-order hold discretization, this model can be expressed on state space form as,

$$\begin{bmatrix} u_1(k+1) \\ u_2(k+1) \\ z_{\text{SoC}}(k+1) \end{bmatrix} = \begin{bmatrix} e^{-\frac{\Delta t}{\tau_1}} & 0 & 0 \\ 0 & e^{-\frac{\Delta t}{\tau_2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1(k) \\ u_2(k) \\ z_{\text{SoC}}(k) \end{bmatrix} + \begin{bmatrix} R_1(1-e^{-\frac{\Delta t}{\tau_1}}) \\ R_2(1-e^{-\frac{\Delta t}{\tau_2}}) \\ \frac{\eta_i \Delta t}{Q_{\text{nom}}} \end{bmatrix} i_{cell}(k) \tag{2.3a}$$

$$u_{cell}(k) = U_{\text{OCV}}(z_{\text{SoC}}(k)) + u_1(k) + u_2(k) + R_0 i_{\text{cell}}(k) \tag{2.3b}$$

where i_{cell} is the input current and u_{cell} is the voltage over the cell. Moreover, u_1 and u_2 represents the voltages over the capacitors with capacitance C_1 and C_2, connected in parallel with the resistors with resistance R_1 and R_2. R_0 represent the ohmic resistance and $\tau_1 = R_1C_1$, $\tau_2 = R_2C_2$ are the time constants. Δt is the sampling time and U_{OCV} is the OCV-curve.

3

Offline Parameter Identification

The main topic of this work is online estimation of the states and the parameters in the model (2.3), but it is still necessary to do offline identification of the parameters that are not estimated online, and to obtain initial values for the parameters estimated online. In this chapter it is presented how the parameters can be identified offline from measurements.

3.1 Ohmic resistance

Consider the state space model (2.3) of a battery cell. The voltages U_{OCV}, u_1 and u_2 have dynamical behavior and they can therefore, for a sufficiently short sampling interval, be approximated to be constant between two adjacent sampling instances. R_0 can then, from (2.3b), be estimated as

$$R_0 = \frac{u_{cell}(k) - u_{cell}(k-1)}{i_{cell}(k) - i_{cell}(k-1)} = \frac{\Delta u_{cell}(k)}{\Delta i_{cell}(k)}, \tag{3.1}$$

given that the denominator is non-zero. The accuracy of the estimate increases with shorter sampling time and with larger change in the applied current. Hence, a suitable experiment could be to apply a large step in the input current and observe the corresponding change in the cell voltage, see Figure 3.1. It should be mentioned that SoC, aging and temperature are affecting R_0 [19], meaning that the experiment needs to be performed for different combinations of these.

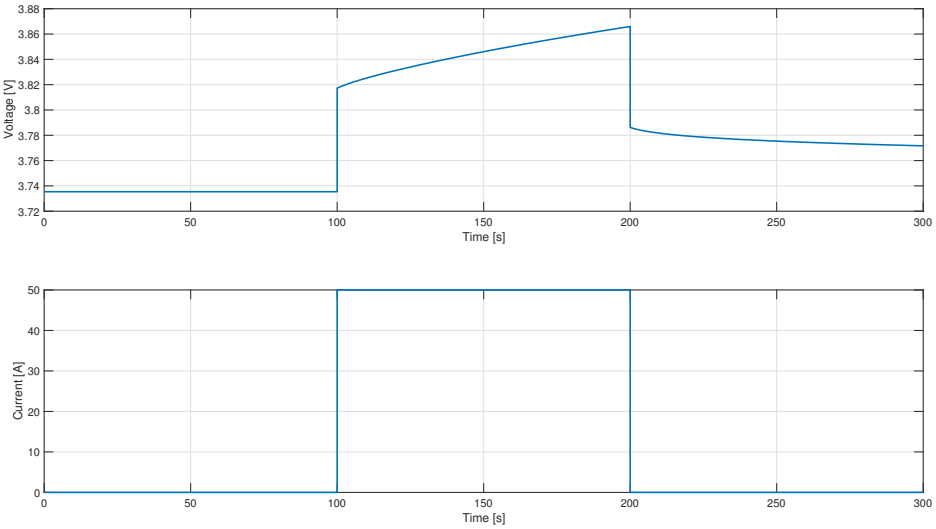

Figure 3.1: Example of an experiment used to determine the ohmic resistance. In this case the ohmic resistance is calculated to $R_0 = \frac{3.82-3.74}{50} = 1.6\text{m}\Omega$

3.2 Identification of time constant

From (2.3), it can be seen that the model has two time constants. In this work, the fast time constant is estimated online, and the slow time constant is identified offline, and is then kept constant.

The slow time constant can be identified from an impulse response. The reason for using an impulse response, instead of a step response, is that it is easier to identify the time constant when the current is zero, otherwise will SoC change during the experiment. An example of an impulse response is seen in Figure 3.2. The immediate drop in the cell voltage is due to the ohmic resistance, while the behavior after 100 seconds is determined by the time constants. The behavior after 100 seconds is however dependant on both the time constants, but assuming that the slow time constant is much greater than the fast time constant, the behavior after 100 seconds will be dominated by the slow time constant. Thus, the slow constant is approximately equal to the time elapsed until 63% of the transients final value is reached [20]. In Figure 3.2 are the voltages, in which between the time constant is calculated, marked with two crosses.

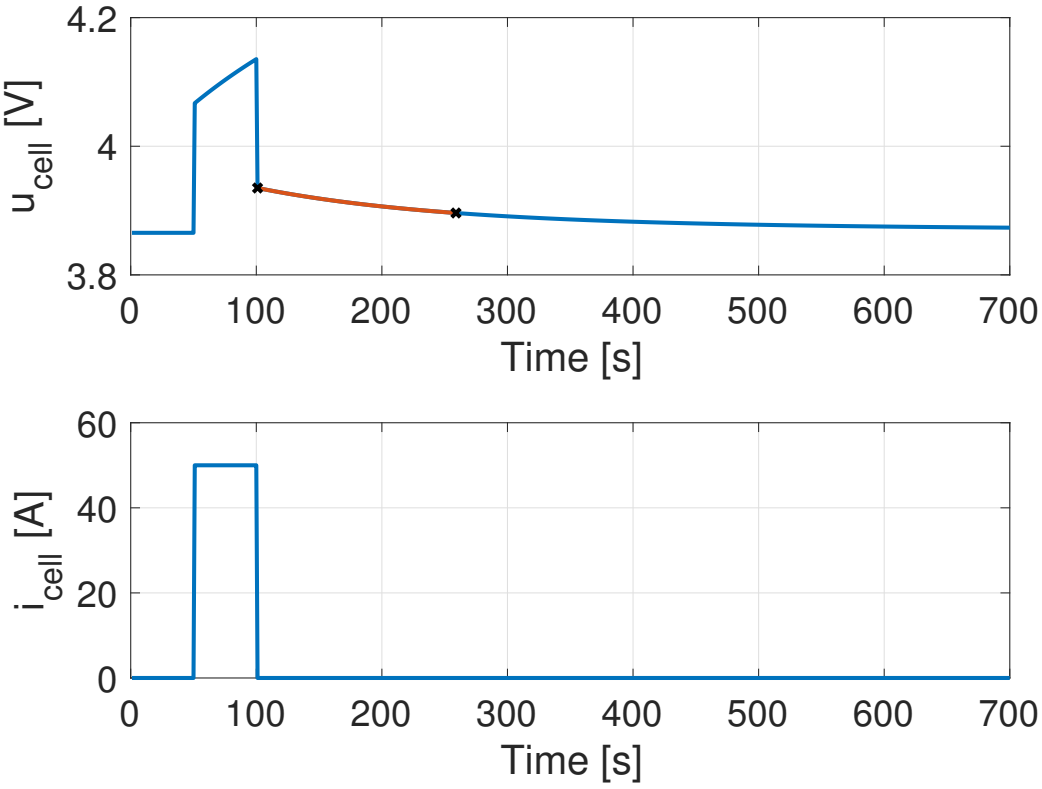

Figure 3.2: Example of an experiment used to determine the slow time constant.

3.3 Identification of capacity

The nominal capacity is the amount of ampere-hours that can be drawn from a fully charged battery cell until it is fully discharged. It could, therefore, be identified by integrating the current when the cell is fully charged until it is fully discharged (or the opposite). It is, however, important to use an unbiased sensor with low variance then.

3.4 Open circuit voltage curve

When the battery is at rest, i.e. when all dynamical transients have disappeared, the OCV can be measured directly. An example of a test cycle used to determine the OCV-curve is shown in Figure 3.3, and in Figure 3.4 the corresponding obtained data values to the OCV-curve is shown. The SoC is obtained from (2.2). Also here it is important to use an unbiased sensor with low variance for the current measurement.

However, for some chemistries there can be a hysteresis effect in the OCV, meaning that the OCV value, for the same SoC, also depends on whether the cell is being charged or discharged. If this effect is small, the average of the OCV-curves obtained

at charging and discharging can be used, otherwise, to prevent biased estimates, two separate curves have to be identified [21].

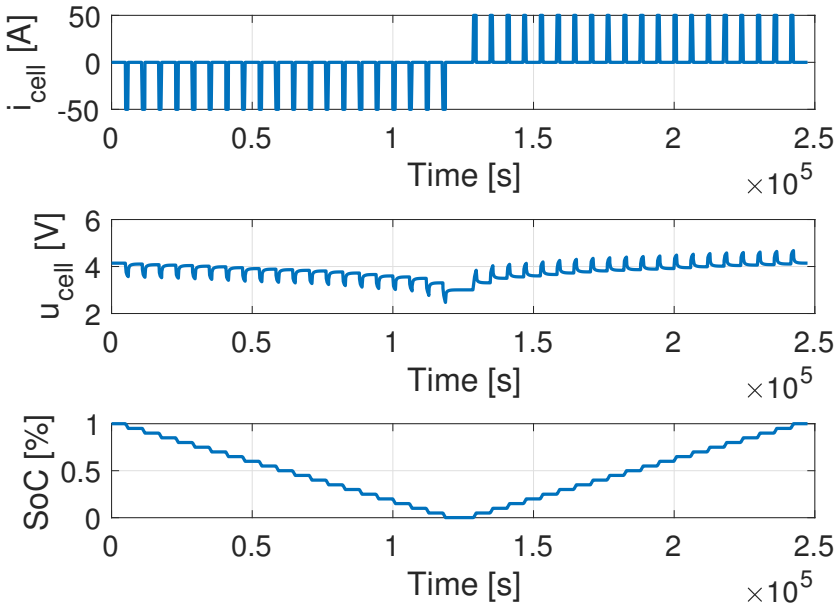

Figure 3.3: Example of test cycle to determine the OCV-curve

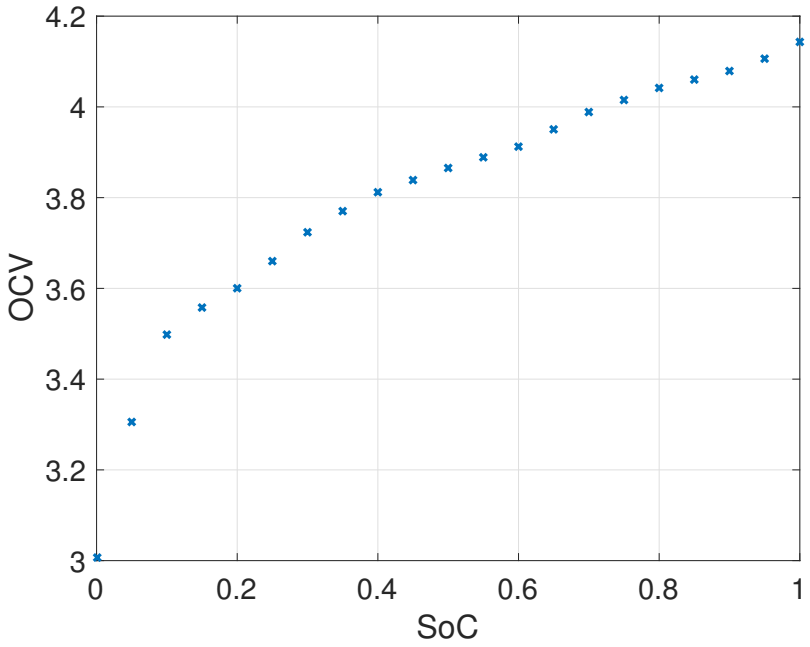

Figure 3.4: Data values to OCV-curve obtained from the test cycle in Figure 3.3

4
Battery Estimation

To maintain accuracy of the model, both the states and some of the parameters in (2.3) need to be estimated online. One way of doing this, that is common in the battery community, is to separate the estimation of the states and the parameters in different estimation schemes [9, 10]. In this work we estimate the capacity, the ohmic resistance, the fastest time constant, and the states. How the different estimators depend on each other is visualized in Figure 4.1.

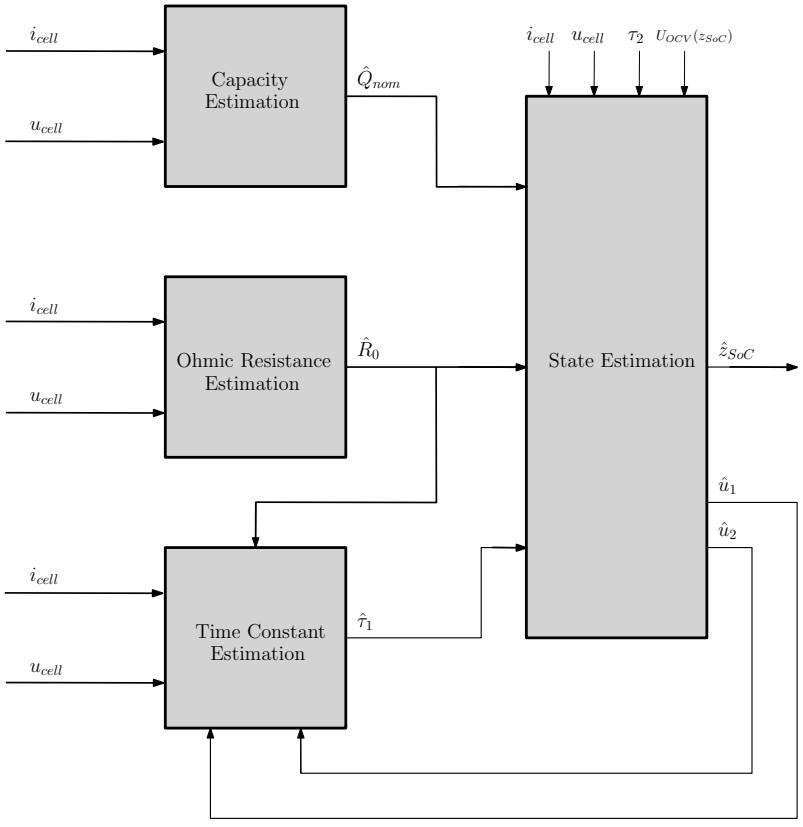

Figure 4.1: Block diagram over the estimation scheme.

4.1 State estimation

For simplicity, we introduce the following notation for (2.3),

$$x(k+1) = A_k x(k) + B_k u(k) + w(k) \tag{4.1a}$$

$$y(k) = h_k(x(k), u(k)) + v(k) \tag{4.1b}$$

where $x(k) = [u_1(k) \quad u_2(k) \quad z_{SoC}(k)]^T$, $u(k) = i_{cell}(k)$, $y(k) = u_{cell}(k)$, and A_k, B_k and h_k correspond to their counterparts in (2.3). Notice that the model is time-variant since the parameters are continuously updated. Furthermore, w and v are white and Gaussian variables, with zero means and covariance Q_S and R_S, respectively.

The state values are estimated using an EKF, which is an extension of the regular Kalman filter for nonlinear systems [22]. The nonlinearities are then, in every time step, approximated with a linearization around the state estimate, and then the regular Kalman filter equations are applied on the linearized system. Hence,

$$\hat{x}^-(k) = A_k\hat{x}(k-1) + B_k u(k-1) \tag{4.2a}$$
$$P_S^-(k) = A_k P_S(k-1)A_k^T + Q_S \tag{4.2b}$$
$$\hat{v}(k) = y_k - h_k(\hat{x}^-(k), i_{cell}(k)) \tag{4.2c}$$
$$S(k) = H_k(\hat{x}^-(k))P_S^-(k)H_k^T(\hat{x}^-(k)) + R_S \tag{4.2d}$$
$$K_S(k) = P_S^-(k)H_k^T(\hat{x}^-(k))S^{-1}(k) \tag{4.2e}$$
$$\hat{x}(k) = \hat{x}^-(k) + K_S(k)\hat{v}(k) \tag{4.2f}$$
$$P_S(k) = P_S^-(k) - K_S(k)S(k)K_S^T(k) \tag{4.2g}$$

where $\hat{x}^-$ and $\hat{x}$ are the estimated *a priori* and *a posterori* state vectors, respectively, and P_S^- and P_S are there corresponding covariance matrices. Moreover, $\hat{v}$ and S are the innovation and its corresponding covariance, K_S is the Kalman gain, and $H_k(\cdot)$ is the Jacobian matrix of h,

$$H_k = \left.\frac{\partial h_k(x(k), i_{cell}(k))}{\partial x(k)}\right|_{x(k)=\hat{x}(k)} \tag{4.3}$$

4.2 Ohmic resistance estimation

In Section 3.1 it was explained how $\hat{R}_0$ could be estimated from a step response. In online applications, however, it is often not possible to manipulate the input to obtain an estimate, which makes that method inapplicable. However, Fridholm et. al. propose that (3.1) could be calculated when Δi_{cell} exceeds a certain threshold, and the estimates could then be weighted together by a Recursive Least Squares (RLS) algorithm [1], i.e.,

$$\hat{R}_0(k) = \hat{R}_0(k-1) + K_R(k)(\Delta u_{cell}(k) - \hat{R}_0(k-1)\Delta i_{cell}(k)) \tag{4.4a}$$
$$K_R(k) = \begin{cases} \frac{P_R(k-1)\delta i_{cell}(k)}{\lambda + \Delta i_{cell}^2(k)P_R(k-1)}, & \text{for } |\Delta i_{cell}| > \delta_{\text{deadzone}} \\ 0, & \text{otherwise} \end{cases} \tag{4.4b}$$
$$P_R(k) = \frac{(1 - K_R(k)\Delta i_{cell})P_R(k-1)}{\lambda} \tag{4.4c}$$

where K_R is the observer gain, P_R is the estimated covariance and λ is a forgetting factor. The threshold, $\delta_{deadzone}$, is introduced to make sure that the system is sufficiently excited before using the data for the estimation update.

4.3 Time constant estimation

In this section, we present an algorithm to estimate the fast time constant, τ_1, that was introduced in [1]. For convenience, the algorithm is also briefly presented here. First we introduce the following notation:

$$\alpha = e^{-\frac{\Delta t}{R_1 C_1}} \tag{4.5a}$$

$$\beta = R_1(1 - e^{-\frac{\Delta t}{R_1 C_1}}). \tag{4.5b}$$

It should be mentioned that both R_1 and C_1 are time dependent, but the notation for this has been omitted to match the notation in other sections. With the notation in (4.5), the first row in (2.3a) shifted one time step becomes,

$$u_1(k) = \alpha u_1(k-1) + \beta i_{cell}(k-1) + e(k-1) = \varphi^T(k)\theta(k), \tag{4.6}$$

where we have added a noise term, $e(k-1)$, to capture errors in the voltage prediction, and introduced $\theta(k) = [\alpha \quad \beta \quad 1]^T$ and $\varphi(k) = [u_1(k-1) \quad i_{cell}(k-1) \quad e(k-1)]^T$.

Fridholm et. al. propose that the parameter vector, θ, can be modelled as a random walk, i.e.,

$$\theta(k+1) = \theta(k) + w(k), \tag{4.7}$$

where $w(k) \sim \mathcal{N}(0, R_w)$. Rearranging (2.3b), we obtain,

$$u_1(k) = u_{cell}(k) - U_{OCV}(\hat{z}_{SoC}(k)) - \hat{u}_2(k) - \hat{R}_0 i_{cell}(k) + v(k), \tag{4.8}$$

where $\hat{z}_{SoC}$, $\hat{u}_2$ and $\hat{R}_0$ are other estimates as seen in Figure 4.1, and $v(k) \sim \mathcal{N}(0, R_v)$, where R_v is a tuning parameter.

By using (4.6) as our measurement model, (4.8) as the measurement, and (4.7) as the transition model, the estimate can be obtained from the following recursive Kalman filter:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)(u_1(k) - \varphi^T(k)\hat{\theta}(k)) \tag{4.9a}$$

$$K_\tau(k) = P(k-1)\varphi(k)(R_v + \varphi^T(k)P(k-1)\varphi(k))^{-1} \tag{4.9b}$$

$$P_\tau(k) = P(k-1) - K(k)\varphi^T(k)P(k-1) + R_w(k), \tag{4.9c}$$

where K_τ is the Kalman gain, P_τ is the estimated covariance matrix and R_w is modelled as,

$$R_w(k) = \frac{P_d\varphi(k)\varphi^T(k)P_d}{R_v(k) + \varphi^T(k)P_d\varphi(k)}. \tag{4.10}$$

This selection of R_w will drive the covariance matrix, P_τ, towards a pre-selected covariance matrix P_d in order to avoid covariance windup [1].

From $\hat{\theta}$, we can extract

$$\hat{R}_1 = \frac{\hat{\theta}_2}{1-\hat{\theta}_2} \tag{4.11a}$$

$$\hat{C}_1 = \frac{\Delta t(\hat{\theta}_1 - 1)}{\hat{\theta}_2 \log(\hat{\theta}_1)}. \tag{4.11b}$$

Moreover, in practical applications, it could sometimes be desirable to remove high frequency content in the measured current and voltage with low pass filters.

4.4 Capacity estimation

One of the most commonly used methods to estimate the capacity is to integrate the current and divide by the corresponding change in SoC [13]. By rearranging (2.2) we get

$$Q_{nom} = \frac{\eta_i \int_{t_0}^{t} i_{cell}(\tau) d\tau}{z_{SoC}(t) - z_{SoC}(t_0)}. \tag{4.12}$$

However, a practical estimator is

$$\hat{Q}_{nom} = \frac{\eta_i \Delta t \sum_{k_0}^{k_1} i_m(k)}{\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0)}, \tag{4.13}$$

where $\hat{z}_{SoC}(k_0)$ and $\hat{z}_{SoC}(k_1)$ are estimates of SoC, $i_m(k)$ is the measured current that not necessarily is the same as $i_{cell}(k)$, and the integral is discretized by zero order hold where k_0 and k_1 are the initial and final time for the integration, respectively.

One way of obtaining SoC estimates is to measure the OCV when the cell is at rest and then use the OCV-curve to obtain the estimate. By assuming Gaussian noise on the voltage sensor, $\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0)$ will be Gaussian, i.e. $\Delta z_{SoC} \sim \mathcal{N}(0, \sigma^2_{SoC})$. Moreover, if $i_m(k) = i_{cell}(k) + v(k)$, where $v(k) \sim \mathcal{N}(0, \sigma_I^2)$, then (4.13) becomes a quotient of two Gaussian variables. This quotient is not Gaussian, but can under certain conditions be rather well approximated by a Gaussian distribution [23]. The expected value of the quotient can be expressed as,

$$E[\hat{Q}_{nom}] = Q_{nom} + \frac{\sigma^2_{SoC} \cdot \eta_i \Delta t \sum_{k=k_0}^{k_1} i_m(k)}{(\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^3} - \frac{\rho \cdot \sigma_{SoC} \cdot \sqrt{k_1 - k_0} \cdot \sigma_I}{(\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^4}, \tag{4.14}$$

where ρ is the correlation between the numerator and the denominator in (4.13). However, since the purpose of (4.13) is to estimate Q_{nom}, the two last terms in (4.14) may cause biased estimates. Fortunately, by estimating SoC via the OCV-curve, $\rho = 0$ and $\sigma^2_{SoC} = (\frac{\partial z_{SoC}}{\partial OCV(z_{SoC})})^2 \sigma_u^2$, where σ_u^2 is variance of the voltage measurement that is normally known. Consequently, the bias can be compensated for by,

$$\hat{Q}_{nom,bc} = \hat{Q}_{nom} - \frac{\sigma^2_{SoC} \cdot \eta_i \Delta t \sum_{k=k_0}^{k_1} i_m(k)}{(\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^3} + \frac{\rho \cdot \sigma_{SoC} \cdot \sqrt{k_1 - k_0} \cdot \sigma_I}{(\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^4}, \tag{4.15}$$

which will have Q_{nom} as expected value. The variance for this estimate is given by,

$$Var[\hat{Q}_{nom,bc}] = \frac{\sigma_{SoC}^2(\eta_i \Delta t \sum_{k=k_0}^{k_1} i_m(k))^2}{\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^4} + \frac{(k_1 - k_0)\sigma_I^2}{\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^2} - \\ - \frac{2\rho\sqrt{k_1 - k_0} \cdot \sigma_I \sigma_{SoC} \eta_i \Delta t \sum_{k=k_0}^{k_1} i_m(k)}{\hat{z}_{SoC}(k_1) - \hat{z}_{SoC}(k_0))^3}. \tag{4.16}$$

In [11] it is proposed that prior information can be utilized, via a Kalman filter, to reduce the variance of the estimates. It is also proposed that (4.15) and (4.16) can be calculated every time the battery is charged, and it is therefore convenient to update the estimator when the new information arrives. It is well known that a battery cell can be cycled many times before the capacity is significantly reduced, and it can therefore be modeled as a random walk in this time-scale, i.e.,

$$\hat{Q}_{nom,KF}(k_c + 1) = \hat{Q}_{nom,KF}(k_c) + e(k_c), \tag{4.17}$$

where k_c is the cycle number, and $e(k_c) \sim \mathcal{N}(0, R_Q)$. By using a Kalman filter, (4.15) and (4.17) can be weighted into the estimate,

$$P_{k_c}^- = P_{k_c - 1} + R_Q \tag{4.18a}$$

$$\hat{Q}_{nom,KF}(k_c) = \hat{Q}_{nom,KF}(k_c - 1) + \\ + \frac{P_{k_c}^-}{P_{k_c}^- + Var[\hat{Q}_{nom,bc}]}(\hat{Q}_{nom,bc}(k_c - 1) - \hat{Q}_{nom,KF}(k_c - 1)) \tag{4.18b}$$

$$P_{k_c}^+ = \Big(1 - \frac{P_{k_c}^-}{P_{k_c}^- + Var[\hat{Q}_{nom,bc}]}\Big) P_{k_c}, \tag{4.18c}$$

where $\hat{Q}_{nom,KF}$ is the estimated capacity, $P_{k_c}^-$ and P_{k_c} are the *a priori* and *a posteriori* covariances, respectively. Also, it should be noted that how much $\hat{Q}_{nom,bc}$ is trusted depends on its variance.

5

Computational Efficient Implementation

In many practical applications, as for electrified vehicles, a battery pack may consist of several hundreds of cells. Since the algorithms, presented in Chapter 4, are developed on cell-level, they need to be implemented for all cells in the pack. This can be computationally (and financially) expensive to do in practice. In this chapter it is therefore presented how these algorithms can be implemented on multiple battery cells with much reduced computational cost.

5.1 Computational cost

The computational cost of an algorithm can be expressed as the number of floating-point operations (flops). A flop is defined by one addition, subtraction, multiplication or division between two floating-point numbers [24].

In linear algebra, the number of flops depends on the dimensions of the vectors and matrices involved in the operation. For instance, an inner product between two vectors $x, y \in \mathbb{R}^n$ are defined as $x^T y = \sum_{i=1}^{n} x_i y_i$. Hence, this operation requires n multiplications and $n-1$ additions, or in total $2n-1$ flops. Similarly, the matrix multiplication $C = AB$, where $A \in \mathbb{R}^{nxp}$ and $B \in \mathbb{R}^{pxm}$ requires $2mnp$ flops.

However, the computational cost can be significantly reduced if symmetries and/or structures in the matrices are considered. For instance, if A is diagonal and $m = n$, $y = Ax$ can be calculated in n flops instead of $2n^2$. Moreover, if also $m = p$ and we know that C is symmetric, it is enough to calculate the diagonal and lower (or upper) triangular part, which almost halves the cost [24].

5.2 State estimation

The state estimator (4.2) includes matrix operations, meaning that the computational cost can be reduced by utilizing structures and symmetries in the matrices. To emphasize the saving potential, we compare two different implementations of (4.2), one where both symmetries and structures are utilized, and one where they are not.

The cost for the implementation that are not utilizing symmetries and structures is straight forward to calculate from the information in Section 5.1 and the sizes of the matrices in (4.2). In Table 5.1 it can be seen that (4.2b) is especially costly since it requires two matrix multiplications, and that the total cost is 216 flops.

In the other implementation, we utilize that A_k and Q_S are diagonal, $\hat{P}_k^-$ and $\hat{P}_k^+$ are symmetric, and H_k contains two elements that are equal to one. To exemplify, we consider (4.2b),

$$P_k^- = A_k P_{k-1} A_k^T + Q_S = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{12} & p_{22} & p_{23} \\ p_{13} & p_{23} & p_{33} \end{bmatrix} \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix} =$$
$$= \begin{bmatrix} \alpha_1^2 p_{11} + q_1 & \alpha_1 \alpha_2 p_{12} & \alpha_1 p_{13} \\ \alpha_1 \alpha_2 p_{12} & \alpha_2^2 p_{22} + q_2 & \alpha_2 p_{23} \\ \alpha_1 p_{13} & \alpha_2 p_{23} & p_{33} + q_3 \end{bmatrix} \tag{5.1}$$

By calculating the flops in (5.1) it is realized that they can be reduced from 117 to 11. In the same manner, flops can be saved in most of the equations in (4.2). In Table 5.1 it is seen that the total number of flops can be reduced from 216 to 70, which corresponds to a reduction by 64%. How the number of flops differs between the two implementations when the algorithm is implemented on many cells is visualized in Figure 5.1.

Table 5.1: Number of flops for state estimation.

Equation	Without	With
(4.2a)	24	8
(4.2b)	117	11
(4.2c)	4	4
(4.2d)	22	13
(4.2e)	21	12
(4.2f)	7	7
(4.2g)	21	15
Total	216	70

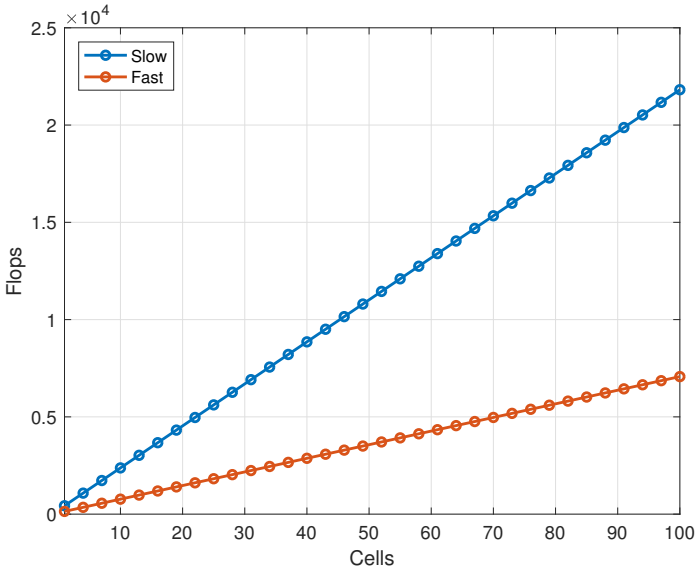

Figure 5.1: Number of flops required to implement the EKF for many cells. "Fast" refers to when symmetries and structures are utilized, and "slow" refers to when they are not.

5.3 Ohmic resistance

By inspecting (4.4a) and (4.4b) it can be understood that all cells connected in series will have the same $K_R(k)$ and $P_R(k)$. Hence, it is enough to calculate them for the first cell, and then re-use them for rest of the series connected cells. Hence, as seen in Table 5.2, the first cell require 13 flops, while the rest of the series connected cells only require 4 flops. This corresponds to a reduction of 69.2% compared to if none of the calculations were re-used. How the number of flops differs between the two implementations when the algorithm is implemented on many cells is visualized in Figure 5.2.

Table 5.2: Number of flops for ohmic resistance estimation.

Equation	First cell	Rest of the cells
(4.4a)	4	4
(4.4b)	5	-
(4.4c)	4	-
Total	13	4

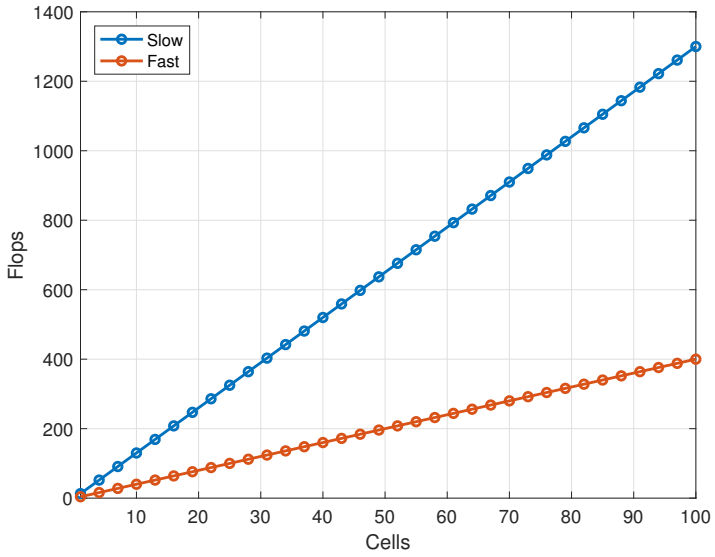

Figure 5.2: Number of flops preformed in the ohmic resistance estimator. "Fast" refers to when calculations are re-used for cells connected in series, and "slow" refers to when no calculations are re-used.

5.4 Capacity estimation

For the capacity estimation in Section 4.4 we can reduce the number of flops by utilizing the same property as for the ohmic resistance, i.e. the observer gain can be re-used for cells that are connected in series. In Table 5.3, the number of flops that are required to compute (4.17) and (4.18) are shown for the first cell and for the rest of the series connected cells. It should be mentioned that the table only includes flops required to be calculated online, thus, for instance the product $\eta_i \Delta t$ can be calculated offline and is therefore not included as a flop in the table.

Normally, most flops can be saved for (4.17). However, since that expression includes a summation of the current, only one flop will be added in each time step. Therefore, it is not as important to minimize those flops as the ones in (4.18). However, by only considering (4.18), the number of flops can be reduced from 9 to 3 which corresponds to a reduction of almost 67%. How the number of flops differs for the two implementations when the algorithm is implemented on many cells is visualized in Figure 5.3.

Table 5.3: Number of flops for capacity estimation.

Equation	First cell	Rest of the cells
(4.17)	$(k_1 - k_0) + 2$	2
(4.18a)	1	-
(4.18b)	5	3
(4.18c)	3	-
Total	$(k_1 - k_0) + 11$	5

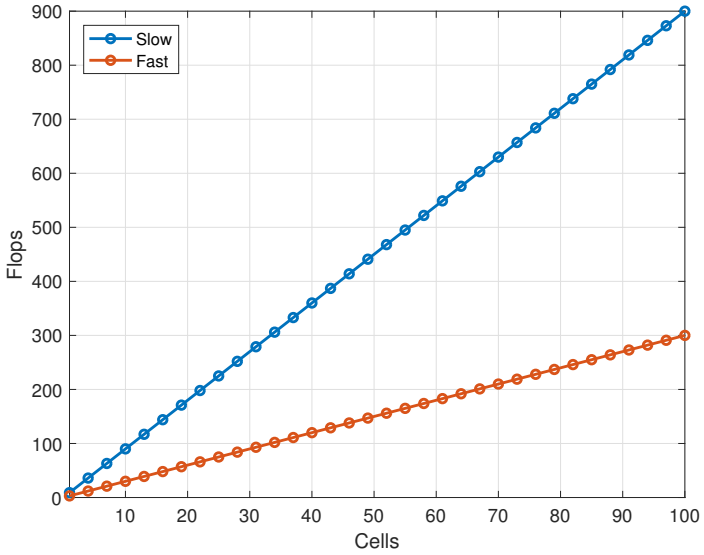

Figure 5.3: Number of flops preformed in the capacity estimator. "Fast" refers to when calculations are re-used for cells connected in series, and "slow" refers to when no calculations are re-used.

5.5 Computational cost of OCV-curve

The approximation of the OCV-curve is also affecting the computational cost. Two ways of doing the approximation are to store data values in a look-up table and then use linear interpolation between the values, or to use a polynomial function fitted to the data values. When it comes to polynomial functions, the number of flops depends on its degree. For the look-up table, however, it is not as easy to calculate the flops because a part of the method is to find where in the look-up table we operate. Therefore, we will consider computational time instead of flops when comparing the methods.

In Matlab, there are two built-in functions that linearly interpolates between data values in a table, `interp1` and `interp1q`. Both of them are rather slow, and we will therefore also consider a self-programmed function, `interpolation`, with the same functionality, see Appendix A for details. They are compared to a polynomial function of degree ten that gave a good fit to the data, as seen in Figure 5.4

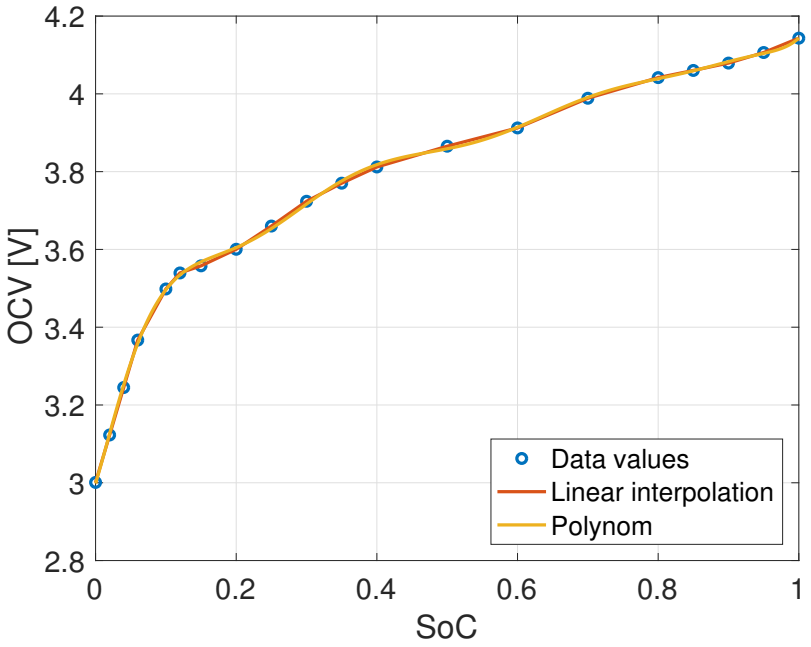

Figure 5.4: OCV-curve approximated with linear interpolation and a 10:th order polynomial function.

The mean and maximum computational time of obtaining 10000 values for the different methods are shown in Table 5.4. It can be seen that the polynomial function is the fastest.

Table 5.4: Computational time for obtaining values from the OCV-curve

Method	Max [s]	Mean [s]
`interp1`	0.8744	0.4594
`interp1q`	0.1507	0.0829
`interpolation`	0.0381	0.0219
Polynomial	0.0231	0.0136

6

Simulation Study

The simulation study consists of two parts; simulation of convergence properties of combined estimation of the states and the parameters, and an investigation of the computational cost. In both parts, data generated from a real battery cell in a laboratory have been used. The parameters that are identified in the offline experiments, as described in Chapter 3, are shown in Table 6.1.

Table 6.1: Parameter values used in the simulation.

Parameter	Value	Unit
R_0	1	mΩ
Q_{nom}	26	Ah
R_1	1	mΩ
R_2	1.5	mΩ
C_1	10000	F
C_2	200000	F
η_i	1	-
Δt	1	s

6.1 Simulation of convergence properties

To evaluate the robustness of the algorithms, convergence for different initial states and parameter values are investigated. The algorithms investigated are presented in Chapter 4, and the initialization of the SoC and the estimated parameter values are randomly selected from an interval of ±20% from the values in Table 6.1. In Figure 6.1 and 6.2, it can be seen that even for rather large initial error in both the SoC and in the parameter values the estimates converge. It should, however, be mentioned that the capacity estimation is evaluated on data generated from differential equations describing the physical reactions in the battery cell [25], since long data series are required to properly show the effect of the algorithm.

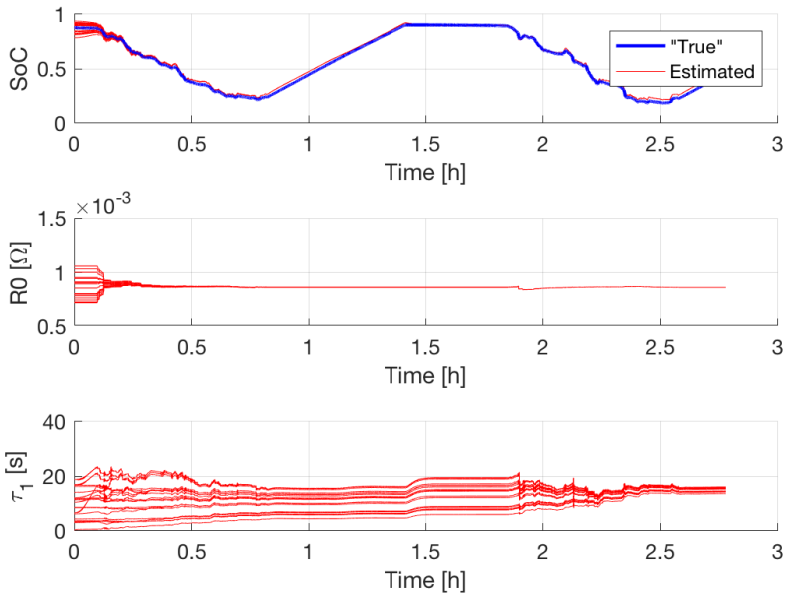

Figure 6.1: Estimates of z_{SoC}, R_0 and τ_1 with randomly selected initial values.

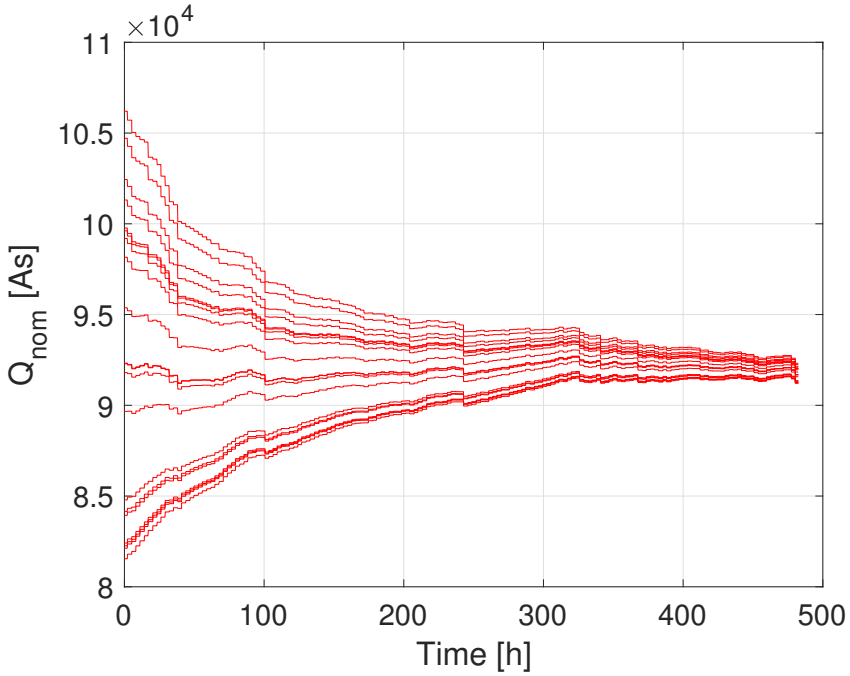

Figure 6.2: Estimates of Q_{nom} with randomly selected initial values.

6.2 Computational cost

In Chapter 5, computationally efficient implementations of the algorithms were presented. In this section, a similar investigation is made, but computational time is

used as the measure instead of number of flops. Moreover, the approximation of the OCV-curve is included in the cost of the algorithms.

For the state estimation, we consider two implementations of the EKF. In the first implementation, both symmetries and structures in the matrices are utilized and the OCV-curve is approximated with a polynomial function of degree ten. In the second implementation, neither symmetry nor structure are utilized and the OCV-curve is implemented using `interp1`. The mean computational time for one iteration is $9.97 \cdot 10^{-6}$ seconds for the first implementation, and $1.16 \cdot 10^{-4}$ seconds for the second. This corresponds to a reduction in computational time by 91.4%.

For the parameter estimators, we run the algorithms on 100 cells connected in series and compare two implementations, one where calculations are re-used between cells, and one when they are not. For the ohmic resistance estimation, the mean computational time for obtaining one estimate on each cell is $3.25 \cdot 10^{-5}$ seconds if calculations are re-used, and $5.20 \cdot 10^{-5}$ seconds if no calculations are re-used. This is a reduction of 37.5%. Similarly for the capacity estimation, the mean computational time is $2.2 \cdot 10^{-4}$ seconds when calculations are re-used, and $1.1 \cdot 10^{-3}$ seconds when no calculations are re-used. This corresponds to a reduction of 78.9%.

The most computationally demanding time steps are those where the states, the resistance, and the time constant are updated simultaneously. Therefore, we evaluate the computational time for this situation by comparing two different implementations. In the first implementation, we use the first state implementation described above, and re-use observer gains in the parameter estimators. In the second implementation, the second state implementation is used, and no calculations are re-used for the parameter estimation. The mean computational time of obtaining estimates for 100 cells connected in series is $3.2 \cdot 10^{-3}$ seconds for the first implementation, and $1.38 \cdot 10^{-2}$ seconds for the second implementation. Thus, the first implementation reduces the computational time by 77% compared to the second implementation.

7
Conclusion

Here we present some concluding remarks of the work, and some suggested topics on future work.

7.1 Discussion

In Section 6.2 an investigation of the computational time of the algorithms was presented. In that investigation, the results were presented in mean computational time, despite that the hardware is usually dimensioned based on maximum computational time. However, it is difficult to isolate computations in Matlab from other activities in the computer. Therefore, mean computational time was selected based on its robustness properties.

The most computationally demanding time steps are when the states, the resistance and the time constant are estimated simultaneously. One way of reduce the computational burden is to skip to estimate the time constant when the resistance is estimated. Since they are estimated from the same signals, high excitation of one of them will probably mean low excitation of the other. This must, however, be investigated before any conclusions can be drawn.

7.2 Future Work

The algorithms have only been evaluated on a short time series of data. In future work, it would be interesting to evaluate the algorithms more extensively. In particular, it would be interesting to investigate if the parameters that were identified offline could be assumed to be constant also for long time series.

Bibliography

[1] B. Fridholm, T. Wik, M. Nilsson, "Robust recursive impedance estimation for automotive lithium-ion batteries", *Journal of Power Sources*, vol. 304, pp 33-41, Feb 2016.

[2] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery pack: Part 3. State and parameter estimation", *Journal of Power Sources*, vol. 134, no. 2, pp. 277-292, 2004.

[3] J. Lee, O. Nam, B. H. Cho, "Li-ion battery soc estimation method based on the reduced order extended Kalman filtering", *Journal of Power Sources*, vol. 174, no. 1, 2007.

[4] A. Klintberg, T. Wik, B. Fridholm, "Theoretical bound on the accuracy of state and parameter estimation in batteries" *American Control Conference (ACC), 2017*, IEEE 2017.

[5] G. L. Plett, "Sigma-point kalman filtering for battery management systems of lipb-based HEV battery packs part 1: Introduction and state estimation," *Journal of Power Sources*, vol. 161, no. 2, pp. 1356–1368, 2006.

[6] X. Hu, F. Sun, Y. Zou, "Estimation of state of charge of alithium-ion battery pack for electric vehicles using an adaptive luenberger observer," *Energies*, vol. 3, pp. 1586–1603, 2010.

[7] I.-S. Kim, "The novel state of charge estimation method for lithium battery using sliding mode observer," *Journal of Power Sources*, vol. 163, no. 1, pp. 584–590, 2006.

[8] J. Yan, G. Xu, H. Qian, and Y. Xu, "Robust state of charge estimation for hybrid electric vehicles: Framework and algorithms," *Energies*, vol. 3, no. 10, pp. 1654–1672, 2010.

[9] C. Zhang, W. Allafi, Q. Dinh, P. Ascencio, J. Marco, "Online estimation of battery equivalent circuit model parameters and state of charge using decoupled least squares technique", *Energy*, vol. 142, pp 678-688, Jan 2018.

[10] M.F. Samadi, S.M.Mahdi Alavi, and M. Saif, "Online state and parameter estimation of the Li-ion battery in a Bayesian framework", *2013 American Control Conference (ACC)*, USA, 2013

[11] A. Klintberg, B. Fridholm, C. Zou, T. Wik, "On State of Health Estimation for Batteries Using Coulomb Counting", *To be submitted*.

[12] S. Lee, J. Kim, J. Lee, B. H. Cho, "State-of-charge and capacity estimation of lithium-ion battery using a new open-circuit voltage versus state-of-charge", *Journal of Power Sources*, vol. 185, pp. 1367-1373, 2008.

[13] X. Tang, X. Mao, J. Lin, B. Koch, "Capacity Estimation for Li-ion Batteries", *American Control Conference*, USA, June 2011.

[14] L. Liu, L. Y. Wang, Z. Chen, C. Wang, F. Lin, and H. Wang, "Integrated system identification and state-of-charge estimation of bat- tery systems," *IEEE Transactions on Energy Conversion*, vol. 28, no. 1, 2013.

[15] R. Xiong, F. Sun, X. G. Hongwen He, "Adaptive state of charge estimator for lithium-ion cells series battery pack in electric vehicles," *Journal of Power Sources*, vol. 241, pp. 699–713, 2013.

[16] G. L. Plett, "Efficient battery pack state estimation using bar-delta filtering," in *EVS24 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium*, 2009.

[17] F. Sun, R. Xiong, and H. He, "A systematic state-of-charge estimation framework for multi-cell battery pack in electric vehicles using bias correction technique," *Applied Energy*, vol. 162, no. 15, pp. 1399– 1409, 2016.

[18] N. A. Chaturvedi, R. Klein, J. Christensen, J: Ahmed, A. Kojic, "Algorithms for Advanced Battery-Management Systems; Modelling, Estimation and Control Challanges for Lithium-ion Batteries", *IEEE Control Systems*, vol. 30, no. 3, pp 49-68, June 2010.

[19] B. Fridholm, T. Wik, M. Nilsson, "Kalman filter for adaptive learing of look-up tables with application to automotive battery restiance estimation", *Control Engineering Practice*, vol. 48, pp 78-86, Mar 2016.

[20] S. Jiang, "A Parameter Identification Method for a Battery Equivalent Circuit Model", *SAE International*, 2011.

[21] R. Tjandra, S. Thanagasundram, K. J. Tseng, A. Jossen, "Improved Lithium-Ion Battery Model with Hysteresis Effect", *IEEE Transportation Electrification Conference*, USA 2014.

[22] S. Sarkka, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013, vol. 3.

[23] J. Hayya, D. Armstron, N. Gressis, "A note on the ratio of two normally distributed varibles", Management Science 21 (11) (1975) 1338-1341.

[24] S. P. Boyd, L. Vandenberghe, *Convex Optimization*, New York, USA, Cambridge University Press, 2004.

[25] M. Torchio, L. Magni, R. B. Gopaluni, R. D. Braatz, D. M. Raimondo, "LIONSIMBA: A Matlab Framework Based on a Finite Volume Model Suitable for Li-Ion Barrery Design, Simulation, and Control", *Journal of The Electrochemical Society*, vol. 163, no. 7 pp 1192-1205, Apr 2016.

A

Appendix A

The attached Matlab function is a self-programmed function that linearly interpolates between values in a table. It can be seen that the function calculates OCV from SoC, which is the causality used in the state estimator. The opposite causality, used in the capacity estimation, can easily be obtained by changing the order of the columns in `OCVcurve`.

```
function OCV = interpolation(SoC)
%SoC - [1x1]. Input value of State of Charge
%OCV - [1x1]. Output value of Open Circuit Voltage

    %OCV-curve
    OCVcurve = [0    3.0009
        0.02 3.1229
        0.04 3.2449
        0.06 3.3669
        0.10 3.4983
        0.12 3.5395
        0.15 3.5578
        0.20 3.6004
        0.25 3.6601
        0.30 3.7238
        0.35 3.7703
        0.40 3.8120
        0.50 3.8654
        0.60 3.9124
        0.70 3.9887
        0.80 4.0418
        0.85 4.0602
        0.90 4.0791
        0.95 4.1063
        1.00 4.1432]; % First column: SoC, second column: OCV

    %Find index in OCV-curve which is below SoC-input
    loweridx = find(OCVcurve(:,1) <= SoC, 1, 'last' );

    if loweridx == 20 %Avoid NaN when loweridx == 20.
        upperidx = [];
```

```
    else %Otherwise, Upperidx = loweridx + 1.
        upperidx = loweridx + 1;
    end
    %Calculate linear interpolation to obtain OCV-value.
    %Explanation:
    %OCV = LowerValue + scale * (UpperValue-LowerValue)
    OCV = OCVcurve(loweridx,2) +  (SoC-OCVcurve(loweridx,1))/...
    (OCVcurve(upperidx,1)-OCVcurve(loweridx,1))*...
    (OCVcurve(upperidx,2)-OCVcurve(loweridx,2));

end
```