UNIVERSITY OF TURKU

Comparison of different battery models and equivalent circuit model optimization

Master of Science in Technology Thesis

University of Turku

Department of Mechanical and Materials Engineering

Materials Engineering

Materials of Energy Technology

Author:

Ville Koponen

Supervisors:

Dr. Jerzy J. Jasielec, PhD (Tech)

12.6.2025

Turku

Master of Science in Technology Thesis
University of Turku
Department of Mechanical and Materials engineering
Materials Engineering

Subject: Materials Engineering
Author: Ville Koponen
Title: Different battery models and equivalent circuit model optimization
Supervisors: Dr. Jerzy J. Jasielec
Number of pages: 47 pages + 22 pages of appendices
Date: 12.6.2025

Abstract

The world is moving in a direction where the use of renewable energy is intended to be increased in order to reduce environmental burden and move away from the use of fossil fuels. Batteries allow energy to be stored and released later when needed. Lithium-ion batteries (LIBs) are lightweight and energy-dense, which makes them very attractive for applications in electric cars, in portable devices such as phones and laptops, and in stationary energy storage.

LIBs can be described using a variety of models, including electrochemical, data-driven, empirical, and equivalent circuit models. A good battery model is accurate, not complicated and can be used in many different applications. In particular, the equivalent circuit models (ECM) can accurately describe the operation of batteries, are not complex, and can be used in various applications such as in battery management systems. Parameters of the ECM need to be established towards individual battery chemistries.

In this work the models for used for the description of LIBs are summarized. Four equivalent circuit models are selected and optimized for two most ubiquitous types of LIBs, namely lithium nickel manganese cobalt oxide (NMC) and lithium iron phosphate (LFP) batteries. The accuracy of the models is examined and discussed.

Key words: lithium-ion batteries, battery models, equivalent circuit model, optimization.

Table of contents

Abbreviations List

Abbreviation	Explanation
AC	alternating current
BMS	battery management system
DDBM	data-driven battery model
ECM	equivalent circuit model
EM	empirical aging model
EVs	electric vehicles
LAM	loss of active material
LFP	lithium iron phosphate
LIBs	lithium-ion batteries
LLI	loss of lithium inventory
ML	machine learning
NMC	lithium nickel manganese cobalt oxide
NN	neural network
OCV	open-circuit voltage
P2D	pseudo-two-dimensional
PNGV	the partnership for a new generation of vehicles model
Rint	the internal resistance model
SOC	state of charge
SOH	state of health
SPM	single-particle model
SVM	support vector machines
redox	oxidation-reduction

1 Introduction

Batteries are now a part of everyday life and are used in a wide variety of applications. The various needs of different industries have led to the development of a wide variety of batteries with certain characteristics. In electric vehicles (EVs), it is important that the batteries are light, can quickly produce energy under dynamic charge/discharge profiles and are functional in different weather conditions. In the storage of renewable energy, batteries with high capacity are needed to balance the supply and demand of electricity. Batteries can store excess energy generated during periods of overproduction and release it when the demand for electricity is increased.

Batteries enable the use of electricity when it is not directly available from the power grid. They ensure the availability of energy even in situations where energy cannot be produced directly on site and there is a constant need for energy supply. Batteries play a key role in today's world in many different areas, such as transport, communication and energy management. In EVs, the chemical energy stored in the batteries serves as the vehicle's energy source. EVs are non-polluting when driving compared to old combustion engine powered vehicles. Thus, the development of batteries for EVs helps to reduce dependence on fossil fuels and promotes sustainable development [1]. Almost all mobile devices, such as smartphones, tablets and laptops, run on batteries. Batteries enable the use of devices without the need for constant connection to the grid. This enables devices with large capacity batteries to be portable for a long time before the battery needs charging. Batteries can also be used in the management of renewable energy. They store energy when there is plenty of production and release it according to the consumption needs. In this way, energy production can adapt to fluctuating demand [2].

1.1 Operating principle of the lithium-ion batteries (LIBs)

Lithium-ion batteries (LIBs) are rechargeable energy storage devices that operate based on oxidation-reduction (redox) reactions occurring between the anode and the cathode. Processes of intercalation and de-intercalation, where Li+ is inserted/removed into electrode material’s structure, are coupled with reduction and oxidation reactions, respectively. LIBs are widely used in portable electronics, EVs and stationary energy storage. The advantages of LIBs are high power and energy densities in both gravimetric and volumetric terms, which are the most important parameters for applications in portable electronics. The price of LIBs has decreased,

which is why they also compete against the aqueous redox flow batteries for large-scale (grid) energy storage applications. [4] LIB consists of two electrodes, a positive cathode (+) and a negative anode (-), separated by an electrolyte. The electrolyte allows the movement of ions from the anode to the cathode and vice versa but prevents the movement of electrons directly between them.

During charging (schematics shown in figure 1), the battery is supplied with power from an external source, forcing the electrons to move through an external circuit from the cathode (+) to the anode (-). At the same time, lithium ions (Li^+) move from cathode (+) through the electrolyte to the anode (-), where they intercalate into the graphite layers [5]. In the charging state of the battery, a reduction reaction takes place at the anode and an oxidation reaction takes place at the cathode [5]:

$$\text{Anode: } C_6 + Li^+ + e^- \rightarrow \text{Li}C_6 \text{ (reduction)}$$

$$\text{Cathode: } LiCoO_2 \rightarrow CoO_2 + Li^+ + e^- \text{ (oxidation)}$$

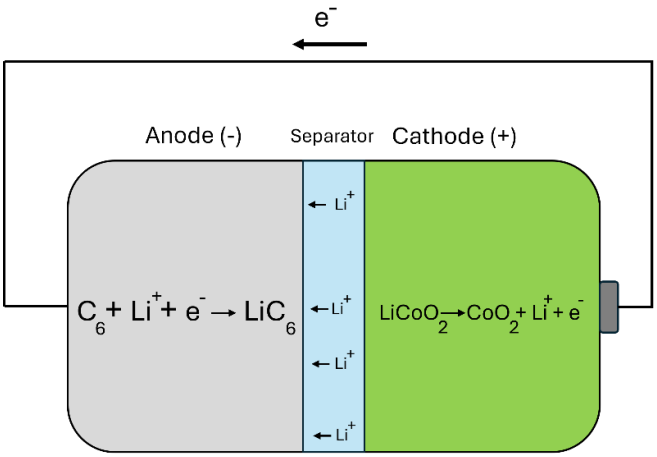


Fig 1. Schematics of Li-ion battery charge process.

When the battery is in use and energy is taken from it, the process reverses: lithium ions (Li^+) move from the anode to the cathode through the electrolyte and release energy. In the discharging (use of the battery) state of the battery, a reduction reaction takes place at the cathode and an oxidation reaction takes place at the anode [5]:

$$\text{Anode: } \text{Li}C_6 \rightarrow C_6 + Li^+ + e^- \text{ (oxidation)}$$

$$\text{Cathode: } CoO_2 + Li^+ + e^- \rightarrow LiCoO_2 \text{ (reduction)}$$

The released electrons move through an external circuit from the anode (-) to the cathode (+), generating a current that can be utilized in various applications [5]. The process of discharging is depicted in figure 2.

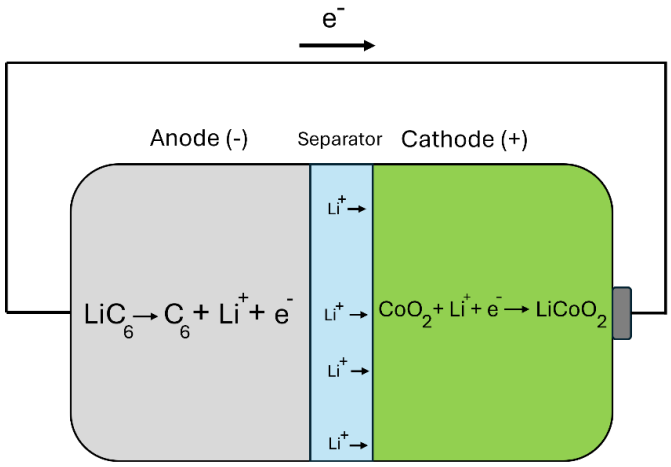


Fig 2. Schematics of Li-ion battery discharge process.

1.2 Types of LIBs

There are different types of LIBs, and each has its own advantages and disadvantages. These different types of LIBs differ by the cathode material that significantly affects the battery properties. The cathode materials include. lithium cobalt oxide, lithium manganese oxide, lithium titanate oxide, lithium nickel cobalt aluminum oxide, lithium nickel manganese cobalt oxide (NMC), and lithium iron phosphate (LFP) [3].

In this master's thesis, NMC and LFP batteries were chosen due to their wide applicability and attractive properties, and described using battery modeling. Advantages of NMC batteries include high energy density, balance between energy and power density, safety, cost, life span and performance. However, on the downside thermal stability and cycle life are worse compared to LFP. NMC also contains environmentally harmful materials, nickel and cobalt. The advantages of LFP are thermal stability, which increase safety, longer life span, lower cost, and that it does not contain materials harmful to the environment. However, the disadvantages are lower energy density, which means bulkier sizes, and lower nominal voltage compared to NMC and other technologies mentioned above. [3]

1.3 Purpose of this work

The purpose of this thesis is to introduce different battery modeling methods, including electrochemical, data-driven, empirical and equivalent circuit models. The second purpose is to delve deeper into equivalent circuit models, which are attractive due to their simplicity and versatility. Four different equivalent circuit models are optimized, and the accuracy of the models is examined. Optimization is important because it allows for more accurate estimation of battery voltage under various load and environmental conditions. The model parameters for the LFP and NMC batteries are established.

The developed models and their parameters can be used in the future research, to predict the battery efficiency in various battery management systems. In particular, they will be used in the ongoing "Home Energy Management Systems" project, cofounded by the University of Turku and the city of Salo.

This thesis answers the following research questions: How well different equivalent circuit models are able to model the voltage of LFP and NMC batteries as a function of SOC. Are there any differences between the models and what is the accuracy of the models? Does the use of different optimization methods allow to find different sets of optimal parameters? Or in other words: is the error function, a function with multiple minima?

2 Categorization of battery models

Along with the development of different batteries, various battery models have also been developed. The purpose of these models is to demonstrate the operation of the batteries and to give a better picture of the basic features of the batteries. They can be used to describe the physical limits of batteries and predict batteries behavior under different conditions. Numerous models have been developed for different purposes. Each model has its own advantages and disadvantages. Some are suitable for use in a wide variety of applications, while others are only suitable for a specific application. Electrochemical models are very accurate, but their disadvantage is complexity and the need for high computing power. Manufacturers use electrochemical models for development and research purposes. Less accurate but much simpler equivalent circuit models have been developed to simulate electric systems. [6]

Many of the models can be used to model important battery properties such as state of charge (SOC) and state of health (SOH) [7]. SOC quantifies the remaining capacity of the battery in real-time, and SOH indicates the degradation and aging of the battery over time compared to its ideal conditions. When starting to look at the application, the first thing to do is find out what is the purpose that the modeling is needed. Each application of the model needs a slightly different approach and parameters. Battery models can be divided by different criteria. The general division is presented in table 1.

Table 1. Classification of the battery models [6].

Perspective of modeling	**Level (depth) of modeling**	**Technique or approach of modeling**	**Time scale of the models**
Electrochemical Electrical Thermal Mechanical Molecular Combination (electro-thermal, etc.)	System Pack Stack and module Full cell Half cell Material	Physical based Empirical Analytical/Mathematical Equivalent circuit Stochastic Hybrid	Short term (partial charge) Medium term (full cycle) Long term (multiple cycles)

2.1 Electrochemical models

The Electrochemical models are based on the chemical processes that take place inside the battery. These models describe the battery processes in great detail. This is why electrochemical models are the most accurate battery models [8]. Electrochemical models include pseudo-two-dimensional model (P2D), as well as single-particle model (SPM) and other simplified P2D models[9].

2.1.1 Pseudo-two-dimensional model (P2D)

The electrodes of LIBs are porous in structure. This increases the interfacial area between the solid and the electrolyte solution. Because the structure is complex, it has been difficult to develop a reliable model for the battery. In 1975, Newman and Tiedemann used macroscopic approach, to develop the porous electrode theory, specifically for battery applications [10] In 1993, Doyle, Fuller and Newman published P2D model for LIBs [11]. The P2D model combined porous electrode theory and concentrated solution theory. The model was developed to examine the charging and discharging processes of the battery accurately by taking into account the different parts of the structure, the electrodes, the electrolyte and the separator. The P2D model uses a 1D cross section of the battery to simulate the flux of ions during charging or discharging. This is demonstrated in figure 3. The model is called pseudo because the second dimension used to describe radial diffusion inside the electrode material particles is along the radius of the particles and not truly a second dimension. The model contains multiple systems of nonlinear partial differential equations. By solving the equations, the solution is the voltage as functions of time, potentials in the electrolyte, electrolyte phases, salt concentration, current density in the electrolyte as functions of time and position in the cell [8]. The main equations are shown in table 2. 1) The disadvantage of the model is that it cannot show the distribution of current density on the surface of the electrode [12].

The P2D model is suitable for lithium-ion battery design and optimization. It is especially useful for understanding spatial concentration and potential distributions. With the help of the model, it is possible to encompass cell design, estimate the SOC, and with extended version of the model perform thermal analysis and degradation modeling. However, the model has its limitations. It is very complex and requires a lot of computing power. It is not suitable for real-

time applications, such as battery management systems (BMS), which require rapid calculations. [13]

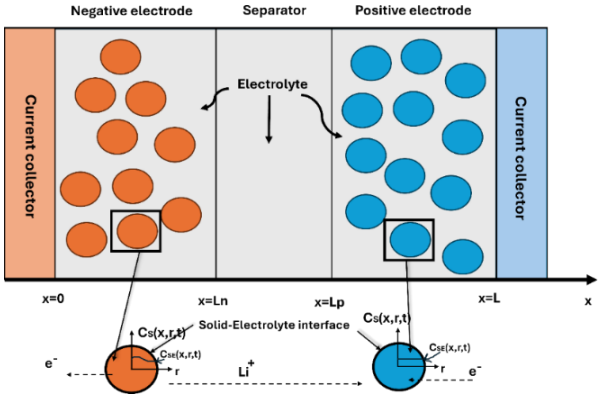


Fig 3. Demonstration of P2D Model of the battery. Model predicts the evolution of lithium concentration in the solid $c_s(x,r,t)$ [12].

Table 2. Main equations of the P2D model [14].

Region	Eq. no.	Equations
Electrodes (k = n,p)	(1)	$\frac{\partial c_{s,k}(x,r,t)}{\partial t} = \frac{D_{s,k}}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c_{s,k}(x,r,t)}{\partial r}\right)$
	(2)	$\epsilon_k\frac{\partial c_{e,k}(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(D_{eff,k}\frac{\partial c_{e,k}(x,t)}{\partial x}\right) + a_k(1-t_+)J_k(x,t)$
	(3)	$\sigma_{eff,k}\frac{\partial^2\Phi_{s,k}(x,t)}{\partial x^2} = a_k F J_k(x,t)$
	(4)	$-\sigma_{eff,k}\frac{\partial\Phi_{s,k}(x,t)}{\partial x} - k_{eff,k}\frac{\partial\Phi_{e,k}(x,t)}{\partial x} + \frac{2k_{eff,k}RT}{F}(1-t_+)\frac{\partial \ln c_{e,k}}{\partial x} = I$
	(5)	$J_k(x,t) = K_k\left(c_{s,k}^{\max} - c_{s,k}^{surf}\right)^{0.5}\left(c_{s,k}^{surf}\right)^{0.5}\left(c_{e,k}\right)^{0.5}\left[\exp\left(\frac{0.5F\mu_{s,k}(x,t)}{RT}\right) - xp\left(\frac{0.5F\mu_{s,k}(x,t)}{RT}\right)\right]$ $\mu_{s,k}(x,t) = \Phi_{s,k}(x,t) - \Phi_{e,k}(x,t) - U_k; \quad V_{cell}(t) = \Phi_{s,k}(0,t) - \Phi_{s,k}(L,t)$
Separator (k = s)	(6)	$\epsilon_k\frac{\partial c_{e,k}(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(D_{eff,k}\frac{\partial c_{e,k}(x,t)}{\partial x}\right)$
	(7)	$-k_{eff,k}\frac{\partial\Phi_{e,k}(x,t)}{\partial x} + \frac{2k_{eff,k}RT}{F}(1-t_+)\frac{\partial \ln c_{e,k}}{\partial x} = I$

1) The solid-state Li^+ ions concentration ($\mathbf{c_s}$) in the electrodes is derived from Fick's law of diffusion for spherical particles.

2) The liquid-phase Li^+ ions concentration ($\mathbf{c_e}$) in the electrolyte and in the separator is based on the conservation of Li^+ ions.

3) The solid-state potential ($\mathbf{\Phi_s}$) in the electrodes is derived from Ohm's law.

4) The liquid-phase potential ($\mathbf{\Phi_e}$) in the electrolyte and in the separator is calculated using Kirchhoff's and Ohm's laws.

5) The pore wall flux of Li^+ ions ($\mathbf{J}$) in the electrodes is described by the Butler-Volmer kinetics equation. [10]

In above equations $\boldsymbol{D_{s,k}}$ and $\boldsymbol{D_{eff,k}}$ denote the diffusion coefficients in solid and electrolyte, $\boldsymbol{\epsilon_k}$ is porosity of region, $\boldsymbol{a_k}$ denotes specific surface area of electrode, $\boldsymbol{\sigma_{eff,k}}$ is effective electronic conductivity of the solid phase of electrode, $\boldsymbol{F}$ denotes Faraday's constant, $\boldsymbol{k_{eff,k}}$ is effective ionic conductivity of the electrolyte in region, $\boldsymbol{R}$ is universal gas constant, $\boldsymbol{T}$ is absolute temperature, $\boldsymbol{t_+}$ denotes Li^+ transference number in the electrolyte, $\boldsymbol{I}$ is applied current density, $\boldsymbol{K_k}$ is reaction rate constant of electrode, $\boldsymbol{\mu_{s,k}}$ denotes over potential of electrode, $\boldsymbol{n}$ represents negative electrode and $\boldsymbol{p}$ denotes positive electrode, $\boldsymbol{U_k}$ is open-circuit potential of electrode, $\boldsymbol{L}$ denotes total thickness, and $\boldsymbol{t}$ is time.

2.1.2 Single particle model (SPM)

Single particle model (SPM) is a simplified 1D electrochemical model, used to study and simulate the performance of LIBs. SPM retains the most basic features that are most important for many applications. It simplifies the ionic diffusion inside the battery particles to make the simulations computationally efficient. The model has the following simplifications. In SPM, it is assumed that the electrolyte concentration is uniform and homogenous. Electrolyte concentration varies only with time and not with respect to location. This helps in solving the diffusion equations, which makes mathematical modeling easier. The active material in the electrodes is represented as a single spherical particle, which reduces the complexity of the calculation. Demonstration of the model is shown in Figure 4. The diffusion coefficient in the solid phase of the electrode material is assumed to be constant. This makes it easier to model the movements of ions in a solid material and enables simpler calculations. Governing equations of the model are presented in table 3. [13]

The SPM is useful for studying the fundamental behavior of LIBs. It is especially used to estimate the SOC of the battery. Due to the simplicity, the calculation is more efficient than for P2D, which makes the model suitable for real-time applications such as battery management systems. However, the model also has limitations. For simplification, many important phenomena such as electrolyte transport, electrolyte concentration gradient and thermal effect are not taken into account. Therefore, the use of the SPM may be limited in applications where these factors have a significant impact, such as heat-hardened analysis or comprehensive life cycle assessment. [13]

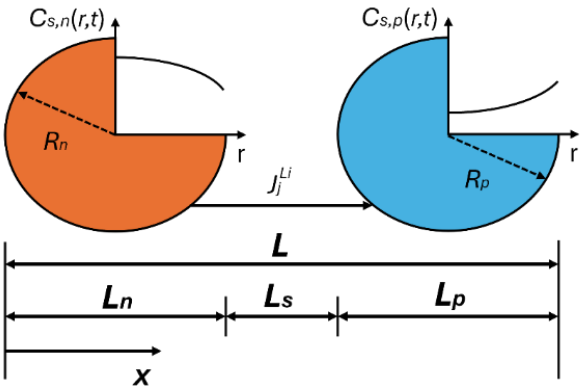


Fig 4. Demonstration of the single particle model (SPM) [15].

Table 3. Governing equations of the SPM model [13].

	Explanation	Equation
Solid phase diffusion equation	Fick's second law of diffusion: Simulate the movement of lithium ions within the particles of solid active material.	$\frac{\partial c_s}{\partial t} = \frac{D_s}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial c_s}{\partial r}\right)$
Overpotential	The difference between the actual applied voltage and the voltage that, according to thermodynamics, would be necessary for a given electrochemical reaction to occur. Over-potential promotes unwanted battery processes such as lithium plating, where lithium ions are released from the electrolyte and accumulate on the surface of the anode when the cell is charged at high current, permanently degrading the battery's performance.	$\eta = \left(\phi_{s,j} - \phi_{e,j}\right) - U(c_s)$
Overall cell voltage	Deducting the anode voltage from the cathode voltage in the solid phase and taking into account overpotentials brought on by reaction kinetics and ohmic losses in the electrolyte.	$U_{batt} = \left(\phi_{s,p} - \phi_{s,n}\right) - I \cdot R_{cell}$
Electrode voltage equations	Each electrode's voltage is determined by the lithium content within the particles of the active material.	Measured empirically and incorporated into the model are the open-circuit voltage (OCV) relations.

$t =$ time

$r =$ radial distance from the center of single particle model

$C_s =$ concentration of lithium ions at "r" distance from the center of the single particle model

$\frac{\partial c_s}{\partial t} =$ time derivative of the concentration

$D_s =$ the diffusion coefficient of Li ions in the solid phase

$\phi_{s,j} =$ the solid phase potential and j is either an anode or cathode

$\phi_{e,j} =$ the electrolyte phase potential and j is either an anode or cathode

$U(c_s) =$ the Open Circuit Voltage

$U_{batt} =$ battery(cell)voltage

2.2 Data-driven models

Data-driven battery models (DDBMs) use statistical, machine learning (ML), and data analytics techniques to predict battery performance, state, and behavior based on empirical data. Unlike physics-based models, which rely on electrochemical equations, these models use patterns in historical and real-time data to deduce relationships without requiring a detailed understanding of the underlying physics. These types of models have grown in popularity because they can accurately capture complex non-linear behaviors, and they can flexibly utilize different data types. The models use machine learning and large datasets of battery measurements to understand the relationships between input variables (e.g. current, SOC and temperature) and output variables (e.g. capacity and voltage) [16]. Data-driven models are able to capture the dynamic behavior of the batteries, so they can be used for accurate state estimation and prediction. Traditional physical battery models often use simplified assumptions, which is why they are not always able to describe the entire complex operation of the Battery. Because data-driven models use large datasets and are dynamic, they can adapt to changes in battery behavior over time. DDBMs can handle data types such as time-series data or images of electrode microstructures, enabling the integration of multiple data sources to achieve a more comprehensive understanding of battery operation [16]. Various data-driven techniques include neural networks, support vector machines, fuzzy logic, decision trees and Gaussian processes [16].

The downsides of DDBMs are the large amounts of data they require, and the data must be of high quality. Poor quality data can lead to overfitting or underfitting in the model's decision making. Overfitting and underfitting are visualized in figure 5. Overfitting means the situation when a model learns the training data too well, including noise and irrelevant patterns, making it overly complex. As a result, the model performs exceptionally well on the training data but poorly on new unseen data and it fails in generalization. Underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and test datasets [17]. Overfitting can be avoided, for example, by simplifying the model, using regularization techniques and by increasing the size of the training dataset [18]. The underfitting problem can be corrected by increasing the complexity of the model and training the model for a longer time. Different penalty methods in the case of overfitting and early stopping approaches in the case of over/under fitting are also effective [19].

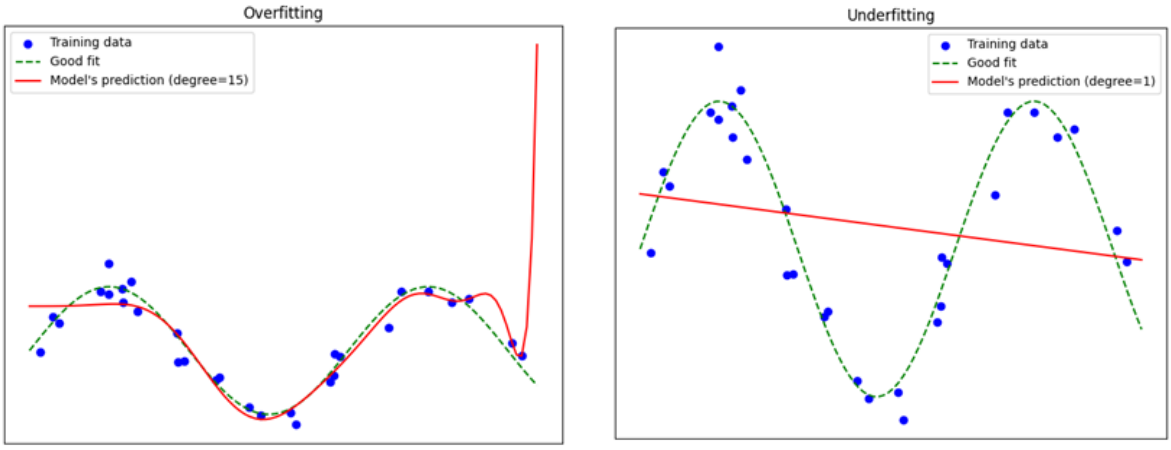


Fig 5. Examples of overfitting (left) and underfitting (right) [20].

2.2.1 Neural network (NN) method

Neural network (NN) is composed of interconnected nodes that process and transmit information. It can handle complex relationships between input and output variables, making them well-suited for battery modeling [16]. The general structure of NN typically consists of three layers, as illustrated in figure 6.

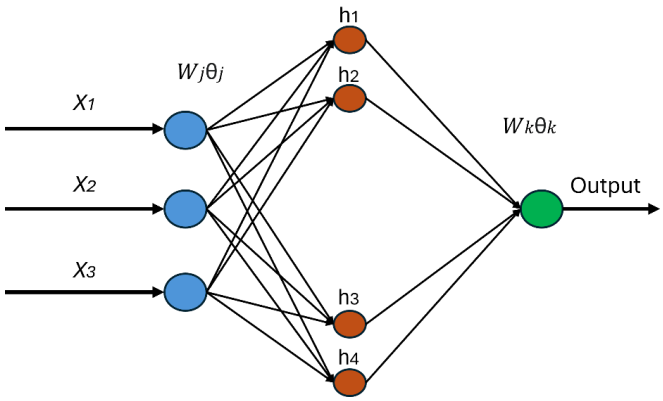


Fig 6. The general architecture of the 3-layer neural network [35].

Estimation of SOC is one of the major applications of NN-based methods in battery modeling. In the training process, a large dataset of input-output pairs is fed into the NN. The input variables describe the operating conditions of the battery (a vector that includes the instantaneous values of current, voltage, and temperature), and the output variable is the corresponding SOC value. NN learns the relationship between the input and output variables and develops a mathematical model that accurately predicts the SOC value from new input data. The link between the input layer and the output layer is established by the appropriate number

of hidden layers, hidden nodes, and an activation function. The SOC in the output layer can be represented as follows [21]:

$$SOC_i = f_i\left\{\sum_k W_{j,k}O_j + \theta_{j,k}\right\}$$

where: f_i is the activation function, $W_{j,k}$ and $\theta_{j,k}$ are the weight and bias from the hidden layer to the output layer, and O_j is the output of the hidden layer.

The model can then be used in real-time applications to estimate SOC when the operating conditions are known. Operation of the batteries can be highly non-linear and can be influenced by a number of factors, such as aging of the battery and temperature. NNs can handle non-linear relationships between input and output variables and for that reason they are quite suitable for SOC estimation in batteries. NNs can be applied to different batteries, regardless of their chemical characteristics. Various battery types have different charge-discharge behavior, and accurate SOC modeling must account for these differences. [16]

2.2.2 Support Vector Machines (SVM)

Support Vector Machines (SVM) are monitored learning algorithms which are designed for both classification and regression tasks. SVM determines the optimal hyperplane that either separates data points into different classes or predicts a continuous output variable [16]. The optimal separation boundary is a hyperplane that maximizes the gap to the closest data points from any class. [21]. This is demonstrated in the figure 7, which shows an example of a hyperplane separating different classes.

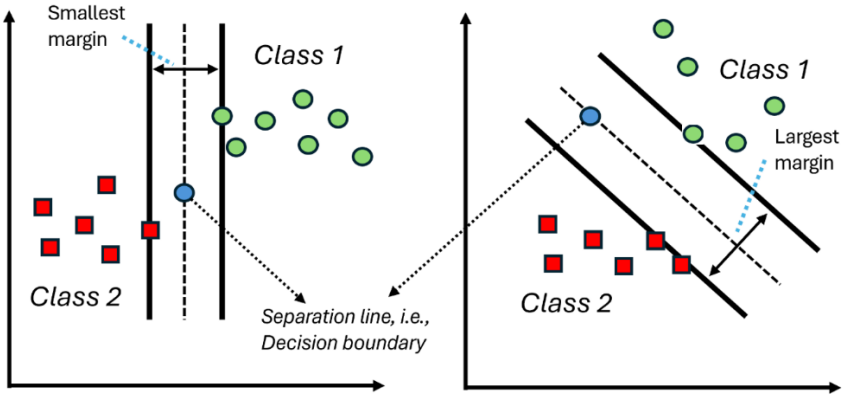


Fig 7. The SVM algorithm aims to create hyperplanes that divide one class from another while maximizing the separation margin. In the image on the right margin is larger (better) and in the image on the left margin is smaller (less acceptable separating line) [22].

SVM are well-suited for battery modeling because they can manage high-dimensional data and non-linear relationships. In battery SOC estimation, SVM are trained on datasets with input variables like voltage, current, temperature, and corresponding SOC values. The SVM finds the optimal hyperplane to separate the data points after which it can accurately predict SOC for new input data. SVM also have strong generalization capabilities, which allow them to perform effectively on data they haven't encountered before. This is important for battery SOC estimation because the model must accurately predict SOC under a wide range of operating conditions, not only those present in the training dataset. SVM can manage noisy data well. Noisy data is often encountered in battery systems due to factors like sensor inaccuracies or variations in battery chemistry. By incorporating noise into the training process, SVM can develop a more reliable model for SOC estimation. [16]

2.3 Empirical and Semiempirical aging models

The purpose of empirical (EMs) and semiempirical aging models is to model battery aging process. With the models, it is possible to study various stress factors, and simple analytical formulas can be obtained by curve-fitting the data. Analytical formulas describe the effect of different stress factors very intuitively, and they are easy to understand. Because semiempirical models are simple, it is possible to use them in a wide variety of applications such as system-level design problems, optimization models, and battery management systems. [23]

However, the models need large test matrices in order to distinguish the effect of individual stress factors from each other. If stress factor interdependence is not understood the model does not give a true picture of the aging process. Other challenges of EMs are that tests are often performed in accelerated conditions with limited equipment. EMs might be also limited only to a specific part of the battery lifetime. EMs tend to oversimplify the aging process of LIBs and the correlation between stress factors. Various stress factors, their mechanisms, resulting degradation modes, and their effects on the battery are presented in figure 8. Loss of lithium inventory (LLI) represents the loss of active lithium ions which are no longer available for cycles. The causes of LLI include parasitic side reactions such as surface film formation, decomposition reactions and lithium plating. LLI is connected to capacity fade. Loss of active material (LAM) represents the structural degradation of the anode or cathode material. Causes of LAM are Surface layer growth or cycling-induced cracks. LAM can cause both capacity fade

and power fade. Conductivity loss means degradation of electrical parts, i.e. current collector corrosion and binder decomposition. [23]

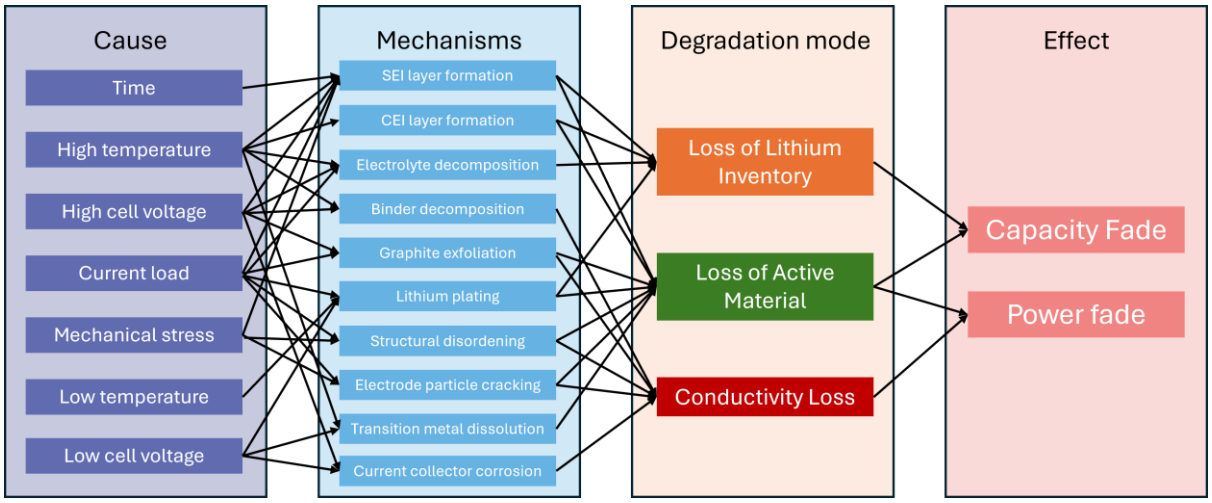
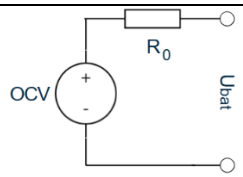
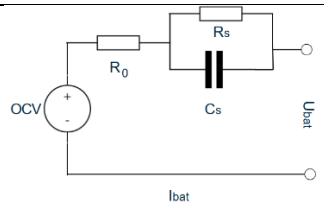
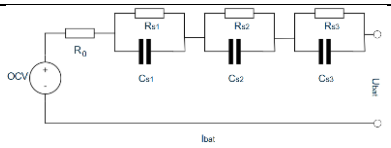
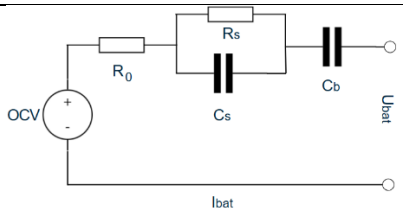
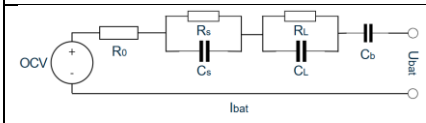
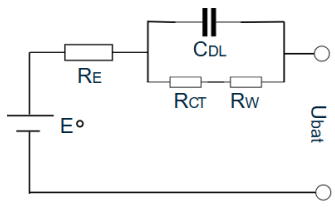


Fig 8. Correlation between stress factors, the corresponding aging mechanisms, aging mode and their effect on LIBs' aging [23].

2.4 Equivalent circuit models

Equivalent circuit is a method used in electrical and electronic design, where a complex electrical component or an entire electrical system is represented by a simplified circuit. In the equivalent circuit models (ECM), batteries are modeled using circuits and components in them. ECMs can be divided into two types: the ECM in the time domain and in the frequency domain. The time-domain ECM simulates the external characteristics such as battery terminal voltage and current through capacitance, inductance, resistor and with other circuit elements. Due to its few parameters and easy identification, it is used in electric vehicle management systems [7]. Different time domain-based models are presented in table 4. Later in this section, each model is discussed in more detail. The frequency-domain ECM establishes the circuit with the same components as the time-domain based ECM. In the frequency domain ECM, response of the sinusoidal excitation at several structural points in the frequency range corresponds to the measured electrochemical impedance spectrum of the battery. So basically, the idea is to simulate the battery's impedance spectrum. In practice, the instrument applies small alternating currents (AC) across the battery over a wide range of frequencies. By measuring the resulting potential across the battery, a Nyquist plot is generated, illustrating the battery's complex impedance across the entire frequency spectrum [7].

Table 4. Various time-domain electrical circuit models [7].

Model Structure	Expression	Model type	Model variables	References.
 Rint model	$U_{bat} = U_{\mathrm{OCV}} - I_{bat} \cdot R_0$ U_{bat} = terminal voltage U_{OCV} = open circuit voltage I_{bat} = discharging current R_0 = Ohm resistance	Analytic function	SOC, T, C-rate	Pathiyil et al. [24], Sibi Krishnan et al. [25]
 Thevenin model	$U_{bat} = U_{\mathrm{OCV}} - U_s - I \cdot R_0$ R_s = *polarization resistance* C_s = *polarization capacitance* U_s = *RC network voltage*	Analytic function	SOC, T, C-rate	Antaloae et al. [26]
 Multiorder Thevenin model	$U_{bat} = U_{\mathrm{OCV}} - U_{s1} - U_{s2}$ $-U_{s3} - I \cdot R_0$	Look-up table	SOC, T, C-rate	Sessa et al. [27], Benato et al. [28]
 PNGV model	$U_{bat} = U_{\mathrm{OCV}} - U_b - U_s -$ $I \cdot R_0$ C_b = *equivalent capacitance*	Look-up table	SOC, T	Zhang et al. [29]
 GNL model	$U_{bat} = U_{\mathrm{OCV}} - U_b - U_s -$ $U_L - I \cdot R_0$ R_s = *concentration polarization resistance* C_s = *concentration polarization capacitance*	Analytic function	SOC, T, C-rate, SOH	Saxena et al. [30], Serrao et al. [31]
 Impedance-based model	$U_{bat} = U_E o - U_{CT} - U_W -$ $-I \cdot R_E$ R_E = *electrolyte and electrode resistance* R_w = *linear ohmic characteristics* R_{CT} = *charge transfer* C_{DL} = *electrochemical double layer effect*	Analytic function	SOC, T	Dai et al. [32], Greenleaf et al. [33]

Battery performance and lifetime depend on several factors, such as temperature, charge and discharge rate, battery charge level, and chemical composition. Equivalent circuit models provide an electrical engineering approach that allows the influence of these factors to be analyzed without the need for full electrochemical simulation. The models allow the evaluation of battery performance under varying conditions, such as under load, at different temperatures, and at different charge levels [9]. With the help of the models, optimal charging and discharging protocols can be designed, which minimize the internal losses of the battery and extend its service life. The SOC and SOH of the battery can be estimated using equivalent circuit models. These are important parameters in the automotive industry when developing new and better EVs [34]. These models can also be used to predict energy consumption and optimization strategies for battery packs, which is especially useful in larger energy storage and industrial applications [35].

2.4.1 The internal resistance model (Rint)

The most widely used battery model is the internal resistance (Rint) model, shown in Table 4. It includes an ideal battery with an open circuit voltage U_{ocv}, a constant equivalent internal resistance R_0, and a terminal voltage U_{bat}. When the battery is fully charged, its open circuit voltage U_{ocv} is higher than when it is discharged. The model is simple and assumes a constant internal resistance and it does not account for how R_0 varies with temperature, SOC, or electrolyte concentration [36]. When fully charged, the terminal voltage U_{bat} can be determined by measuring the U_{ocv}, while R_0 can be calculated by applying a load and measuring the current and terminal voltage. In this model, the SOC of the battery is 100% when the battery is fully charged and 0% when fully discharged. The total charge removed as the SOC decreases from 100% to 0% represents the battery's capacity, Q, which is typically measured in ampere-hours (Ah) or milliampere-hours (mAh). [36] Rint equivalent circuit model has the governing equations [37]:

$$\mathrm{SoC}(t) = \mathrm{SoC}(t_0) - \int_{t_0}^{t} \frac{I(t)\eta}{Q_{\max}}\, dt$$

$$I = \frac{U_0}{R_0}$$

$$U_{bat} = U_{ocv} - R_0 I_0$$

where: U_{ocv} is the open circuit voltage, I is the load current, and R_0 is the ohmic resistance.

2.4.2 Thevenin model

Thevenin model is based on the Rint model. In Thevenin model, a parallel RC network has been added to the Rint model to simulate the battery polarization effect. This model also has a simple structure and has a high simulation accuracy. The advantage of Thevenin model is that it also describes the polarization effect inside the battery, which helps better simulate the dynamic and static characteristics of the battery [38]. There are only few parameters in the model, and its curve fitting process is based on a single exponential function. The subsequent estimation process involves only a few calculations, making it well-suited for SOC estimation in embedded systems and aligning with the application needs of EVs [39]. When charging or discharging the battery, the voltage shows both abrupt and gradual change. In the model, the internal resistance R_0 is used to simulate the characteristics of the abrupt resistance. R_s and C_s are used to simulate the capacitance characteristics of voltage gradual changes [9]. Thevenin model is illustrated in shown in Table 4. Its governing equations are [40]:

$$\mathrm{SoC}(t) = \mathrm{SoC}(t_0) - \int_{t_0}^{t} \frac{I(t)\eta}{Q_{\max}}\, dt$$

$$I = \frac{U_s}{R_s} + C_s \frac{dU_s}{dt}$$

$$U_{bat} = U_{ocv} - R_0 I_0 - R_s \left(1 - e^{\frac{-t}{R_s C_s}}\right)$$

where U_{ocv} is the open circuit voltage, I is the load currentR_0 is the internal ohmic resistance of the battery, R_s is the polarization resistance, and C_s is the polarization capacitance. The terminal voltage can be represented using an exponential term ($IRe^{\frac{-t}{\tau}}$), or calculated from differential equation.

The disadvantages of the model are its lack of applicability in real life applications because all parameters are considered constant and independent on the operating conditions. The model is also unable to simulate capacity fade or the battery runtime due to thermal impacts. However, the model is used and it has been possible to improve it by adding ideal diodes, zener diodes to measure open circuit voltage and internal resistance. [41]

2.4.3 Multiorder Thevenin model

Multiorder Thevenin model is a Rint circuit to which several RC circuits are connected in series. There are usually two or three RC circuits. Third order Thevenin equivalent circuit model, illustrated in Table 4, has the following governing equations [42]:

$$\mathrm{SoC}(t) = \mathrm{SoC}(t_0) - \int_{t_0}^{t} \frac{I(t)\eta}{Q_{\max}}\, dt$$

$$I = \frac{U_{s1}}{R_{s1}} + C_{s1}\frac{dU_{s1}}{dt} = \frac{U_{s2}}{R_{s2}} + C_{s2}\frac{dU_{s2}}{dt} = \frac{U_{s3}}{R_{s3}} + C_{s3}\frac{dU_{s3}}{dt} = \frac{U_0}{R_0}$$

$$U_{bat} = U_{ocv} - R_0 I_0 - R_{s1}\left(1 - e^{\frac{-t}{R_{s1}C_{s1}}}\right) - R_{s2}\left(1 - e^{\frac{-t}{R_{s2}C_{s2}}}\right) - R_{s3}\left(1 - e^{\frac{-t}{R_{s3}C_{s3}}}\right)$$

where U_{ocv} is the open circuit voltage, I is the load current, R_0 is the ohmic resistance, R_{s1} to R_{s3} are the polarization resistances, and C_{s1} to C_{s3} are the polarization capacitances.

When two or three RC circuits are used instead of only one RC circuit in the model, the accuracy of the exponential fitting improves. The polarization effect of a lithium battery is presented equivalently by an RC loop and is equivalent to the first-order zero input response after the battery is discharged. Guo et al. [43] have demonstrated, by plotting exponential curves, that the correction decision coefficients of the double and triple exponential fittings are larger than in single exponential fitting, so the fitting effect is better (closer to 1). Surprisingly, they also found that double exponential fitting gave the most accurate result. The third-order RC loop theoretically provides a more accurate representation of the battery's dynamic characteristics, but it includes one additional RC loop compared to the second-order RC loop. The third-order RC loop introduces two more unknown parameters during the computer-based data fitting process and as a result, the fitting performance of the third-order RC loop is worse than in the second-order RC loop. It is also proved that the second-order RC model is more suitable for modeling LIBs [43].

2.4.4 The Partnership for a New Generation of Vehicles model (PNGV)

The Partnership for a New Generation of Vehicles model (PNGV) is obtained when a capacitor is connected in series to Thevenin model. The circuit diagram of the PNGV model is shown in Table 4. The model is a nonlinear equivalent circuit model. Its parameters depend on voltage, temperature, and SOC. The resistance R_0 represents the change of the battery's internal

resistance. The ideal voltage source and the capacitor C_b capture changes in the open-circuit voltage and capacity, respectively. The parallel RC network characterizes the battery's polarization voltage. [44]

PNGV model has a high accuracy to simulate the transient response process, and it is also suitable for use with large current values [45]. Compared to simpler models, PNGV is better able to handle complex charging and discharging conditions. However, the PNGV model is complex and requires more calculation, which is why it has low real-time performance [9]. PNGV equivalent circuit model has the governing equations [46]:

$$\frac{dU_s}{dt} = \frac{I}{C_s} - \frac{U_s}{R_s C_s}$$

$$\frac{dU_b}{dt} = \frac{I}{C_b}$$

$$U_{bat} = U_{ocv} - U_s - U_b - R_o I$$

where: U_{ocv} is the open circuit voltage, I is the load current, R_0 is the ohmic resistance, R_s is the polarization resistance, C_s is the polarization capacitance, and C_b is the equivalent capacitance.

2.4.5 Other equivalent circuit models

In GNL model, second RC network is connected to the PNGV model [7]. The circuit diagram of the GNL model is shown in Table 4. The model is a nonlinear equivalent circuit model.

Frequency-domain ECM consists of the same components as time-domain ECMs. The circuit diagram of the impedance-based model is shown in Table 4. The principle of impedance-based models is making the response of sinusoidal excitation in a specific frequency range coincide with the measured electrochemical impedance spectrum of the battery. The purpose is to simulate the battery impedance spectrum, not to describe the battery terminal voltage and current. The instrument measures the potential across a battery by applying small AC currents over a wide range of frequencies, and then a Nyquist plot is drawn to show the complex impedance of the battery over the entire frequency range. [7]

3 Optimization of equivalent circuit models

Optimization plays a key role in identifying the best model parameters that ensure a model's predictions closely match experimental data. The purpose of the optimization, presented in this work, is to test how well different equivalent circuit models can predict the voltage change as a function of SOC. A good model is one that can mimic real battery usage, and the error is not very large. In order to test different equivalent circuit models, input data is needed to optimize the parameters of the model. Once the parameters of the equivalent circuit model have been determined, the model is able to mimic the behavior of a real battery.

Central to the optimization process is the goal function, a mathematical expression that defines what needs to be optimized. By analyzing how changes in input variables (parameters of equivalent circuit) affect the value of the goal function, optimization methods identify the set of parameters that yield the most favorable outcome. Optimization for different models was done by minimizing the difference between modelled and measured battery voltages (input data) during charge and discharge cycles with different C-rates. The goal is to find optimal parameters that best fits the battery voltage data across different charging and discharging rates using least squares error minimization.

The goal function for the Rint model takes the form:

$$\sum_{\mathrm{i}} \left(\mathrm{U_{model}}(\mathrm{SOC_i}, R_0) - \mathrm{U_{measured}}(\mathrm{SOC_i})\right)^2$$

where R_0 is the optimized parameter and the value of the modelled voltage is expressed as:

$$\mathrm{U_{model}} = \mathrm{U_{ocv}}(\mathrm{SOC_i}) - \mathrm{R_0 I}.$$

For the Thevenin model, the goal function is expressed as:

$$\sum_{\mathrm{i}} \left(\mathrm{U_{model}}(\mathrm{SOC_i}, \mathrm{R0}, \mathrm{Rs}, \mathrm{Cs}) - \mathrm{U_{measured}}(\mathrm{SOC_i})\right)^2$$

where R_0, R_s and C_s is the optimized parameter and the value of the modelled voltage is:

$$\mathrm{U_{model}} = \mathrm{U_{ocv}}(\mathrm{SOC_i}) - \mathrm{R_0 I_0} - \mathrm{R_s}\left(1 - \mathrm{e}^{\frac{-\mathrm{t}}{\mathrm{R_s C_s}}}\right)$$

For the Second Order Thevenin model, the goal function is expressed as:

$$\sum_{\mathrm{i}} \left(\mathrm{U_{model}}(\mathrm{SOC_i}, \mathrm{R0}, \mathrm{Rs1}, \mathrm{Cs1}, \mathrm{Rs2}, \mathrm{Cs2}) - \mathrm{U_{measured}}(\mathrm{SOC_i})\right)^2$$

where R_0, R_s, C_s, R_{s2}, C_{s2} is the optimized parameter and the value of the modelled voltage is:

$$U_{model} = U_{ocv}(SOC_i) - R_0 I_0 - R_s\left(1 - e^{\frac{-t}{R_s C_s}}\right) - R_{s2}\left(1 - e^{\frac{-t}{R_{s2} C_{s2}}}\right)$$

For the PNGV model, the goal function is expressed as:

$$\sum_i \left(U_{model}(SOC_i, R0, Rs, Cs, Cs2) - U_{measured}(SOC_i)\right)^2$$

where R_0, R_s, C_s, C_{s2} is the optimized parameter and the value of the modelled voltage is:

$$U_{model} = U_{ocv}(SOC_i) - U_s - U_b - R_o I$$

Three different algorithms were used to optimize the parameters, Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Bounds (L-BFGS-B), Sequential Least Squares Programming (SLSQP), and Powell's method.

L-BFGS-B is a quasi-Newton method for solving large-scale optimization problems with box constraints (i.e., variable bounds). It approximates the Hessian matrix (second derivatives) using a limited amount of memory, which makes it suitable for problems with many variables. The advantages of L-BFGS-B are the code is easy to use, and the user need not supply information about the Hessian matrix or the structure of the objective function, the storage requirements are moderate and can be controlled by the user, and the cost of the iteration is low and is independent of the properties of the objective function. As a result, L BFGS-B is recommended for large problems in which the Hessian is large-scale or is difficult to compute. Drawbacks of L-BFGS-B are it is not rapidly convergent, and it can be computationally expensive for large-scale problems, on highly ill-conditioned problems it may fail to obtain high accuracy in the solution, and it cannot make use of knowledge about the structure of the problem to accelerate convergence. [47]

SLSQP performs a quadratic approximation of the objective function at each iteration point, and it uses of both gradient and Hessian matrix information. It is an efficient mathematical optimizer to search fast a local optimum. It finds different local optima with different initial starting trial solutions. A large repetition of running the SLSQP algorithm with random initial trials will yield many local optimal solutions which will show their distribution characteristics [48]. Disadvantages are that the algorithm can be computationally expensive for large-scale problems, and it may fail if the problem is not well-scaled. [49]

The Powell algorithm is a searching algorithm first proposed in 1964 by Powell [50] to solve unconstrained optimization problems. The derivatives do not require calculation because the algorithm only needs to calculate the function values when the function is a continuous one, through constant improvement. The Powell method of least squares is used to search for optimal parameters, and the conjugate equations in the iteration are generated step by step. This is a very effective approach to achieve the minimum value of one function that is not strict with the initial value. The convergence rate is fast, and its partial optimization ability is classical. [51] Disadvantages are that Powell has potential for getting stuck in local optima, it may struggle with non-smooth or discontinuous functions, and it has slower convergence than gradient-based methods. [52]

As mentioned in section 1.3, the purpose of this work is to optimize four selected equivalent circuit models for the description of the two most ubiquitous types of LIBs, namely NMC and LFP. For each type of the battery, all four models mentioned before will be optimized. The data needed for optimization is OCV as function of SOC and voltages as function of SOC with various C-rate values while charging and discharging the battery. When the battery capacity and different C-rate values are known, the current can be calculated during battery discharge and charging with formula below:

$$I = C - rate \cdot Battery\ capacity$$

The Python codes used to perform the optimization can be found in Appendices 2 to 5. Artificial intelligence (OpenAI, ChatGPT v.3.5) has been used to generate the codes.

3.1 LFP Battery

LFP battery used in this thesis is 48V 200Ah 10KW $LiFePO_4$ battery [53]. From the plots provided by the producer, and presented in figure 9, the charge and discharge curves at rates 0.2 C and 0.5 C, were extracted using free online software, Plot Digitizer (https://plotdigitizer.com/app). The OCV potential was calculated as the average between these four curves. The capacity of the battery was 200 Ah, hence 1C rate corresponds to the current I = 200 A.

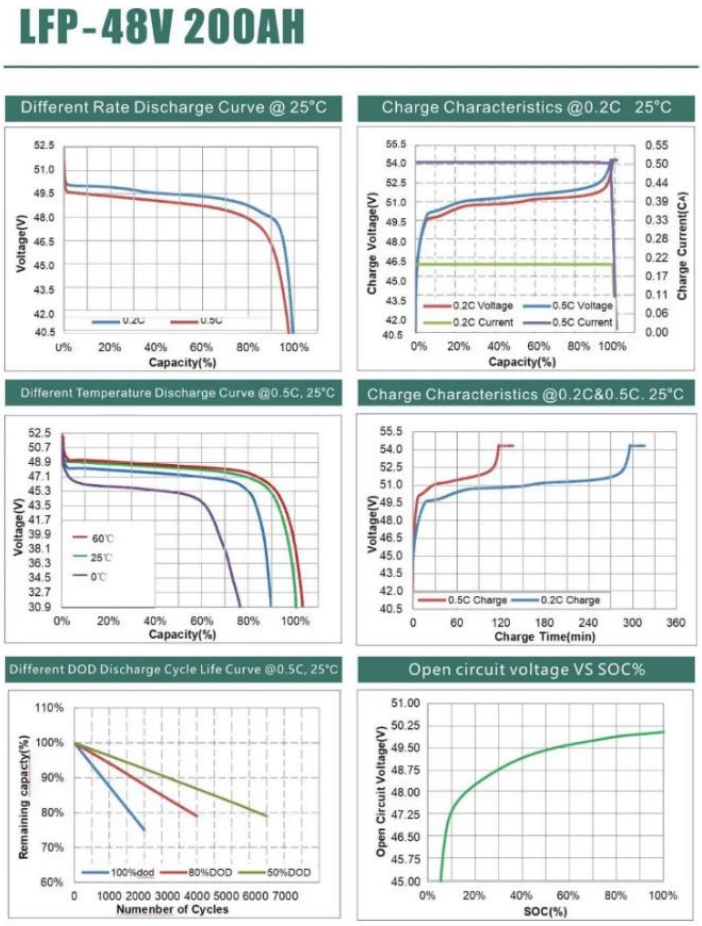


Fig 9. 48V 200Ah 10KW $LiFePO_4$ battery data used to test the accuracy of the models [53].

3.1.1 Rint model

The Rint model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. The R0 parameter value was established to be 0.02078 Ω, and the model was used to calculate voltage values as a function of SOC. Error function value was 76.07 V^2. All methods gave the same minimum error function value and R0 value. The model-predicted and experimental voltage values during charging are depicted in Figure 10. The model-predicted and experimental voltage values during discharge are depicted in Figure 11. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the LFP battery voltage. This is presented in Table 5.

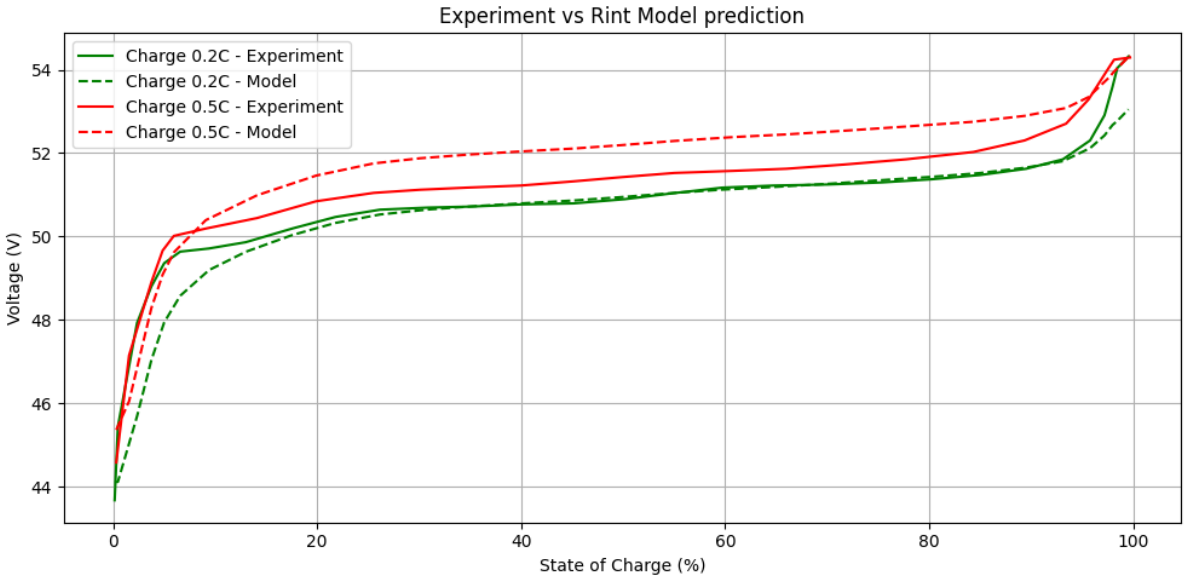


Fig 10. LFP battery voltage and model predicted voltage values as a function of SOC during battery charging.

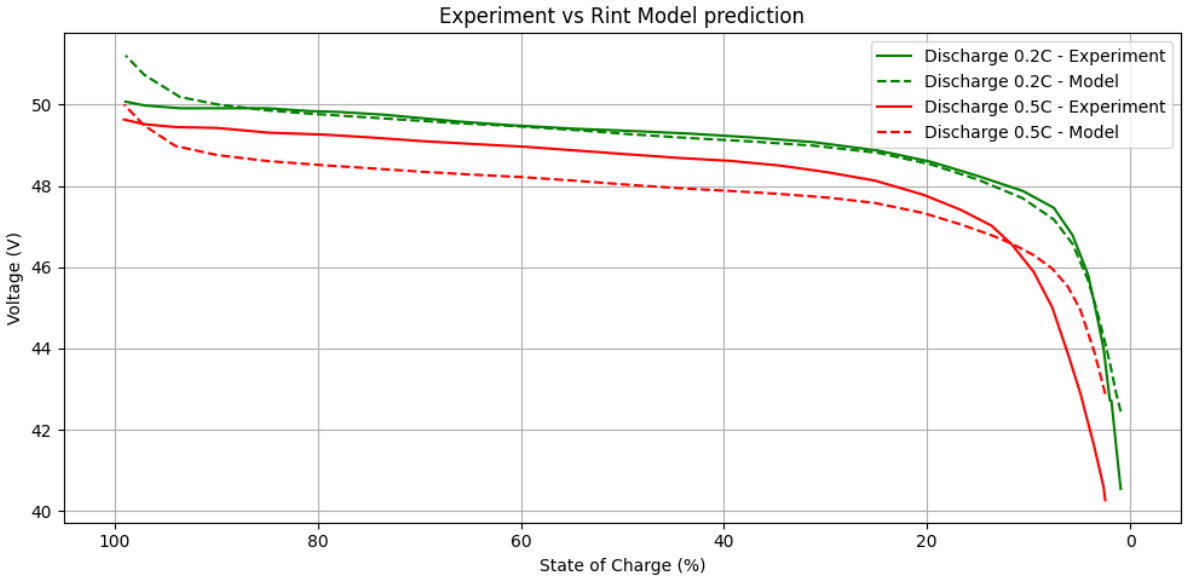


Fig 11. LFP battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 5. Absolute and relative errors between model-predicted and LFP battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.469	0.94
Charging	0.5	0.608	1.20
Discharging	0.2	0.386	0.88
Discharging	0.5	0.870	1.93

3.1.2 Thevenin model

The Thevenin model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. All methods gave the same minimum error function value but different optimized parameters. The optimal parameters found using the selected methods are presented in table 6. The model-predicted and experimental voltage values during charging are depicted in Figure 12. The model-predicted and experimental voltage values during discharge are depicted in Figure 13. The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the LFP battery voltage. Optimization methods resulted in the same absolute and relative errors, as well as the same plotted curves. This is presented in Table 7.

Table 6. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Error Function Value [V^2]
L-BFGS-B	0.01243	0.00835	500	76.07
SLSQP	0.01039	0.01039	1000	76.07
Powell	0.00078	0.02000	1000	76.07

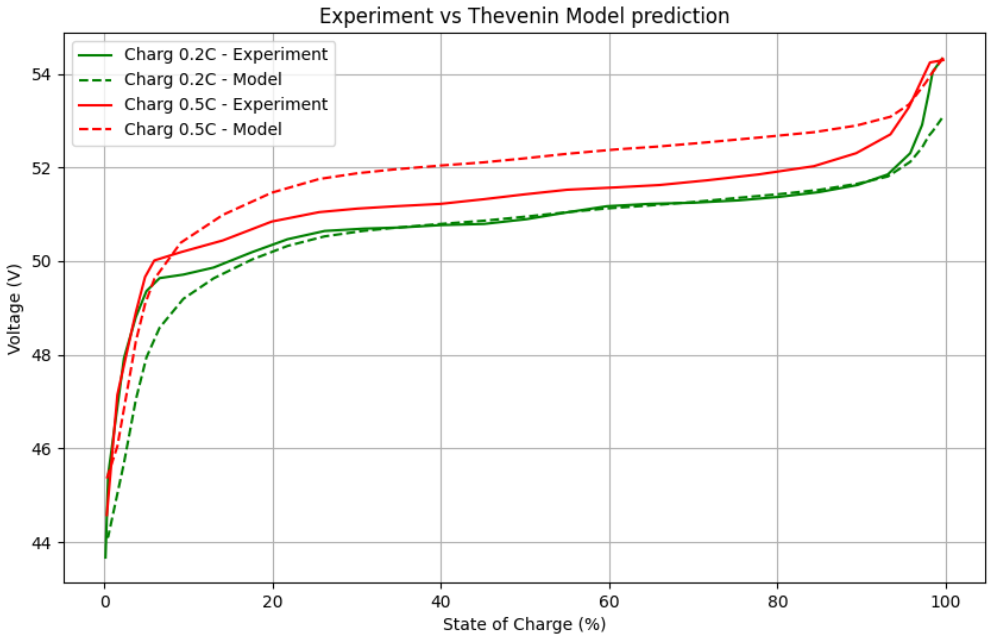


Fig 12. LFP battery voltage and model predicted voltage values as a function of SOC during battery charging.

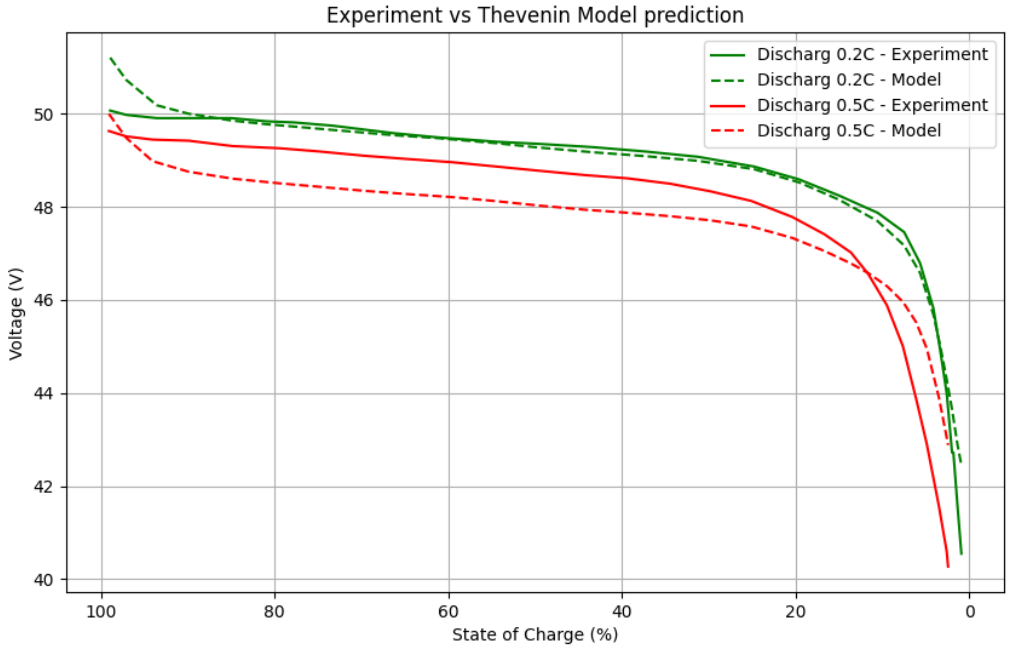


Fig 13. LFP battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 7. Absolute and relative errors between model-predicted and LFP battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.469	0.94
Charging	0.5	0.608	1.20
Discharging	0.2	0.386	0.88
Discharging	0.5	0.870	1.93

3.1.3 Second order Thevenin model

The Thevenin model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. All methods gave the same minimum error function value but different optimized parameters. The optimal parameters found using the selected methods are presented in table 8. The model-predicted and experimental voltage values during charging are depicted in Figure 14. The model-predicted and experimental voltage values during discharge are depicted in Figure 15. The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the LFP battery voltage. Optimization methods resulted in the same absolute and relative errors, as well as the same plotted curves. This is presented in Table 9.

Table 8. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Rs2 [Ω]	Cs2 [F]	Error Function Value [V^2]
L-BFGS-B	0.01013	0.00532	500	0.00532	1000	76.07
SLSQP	0.00635	0.00722	100	0.00722	200	76.07
Powell	0.01278	0.00300	800	0.00500	1500	76.07

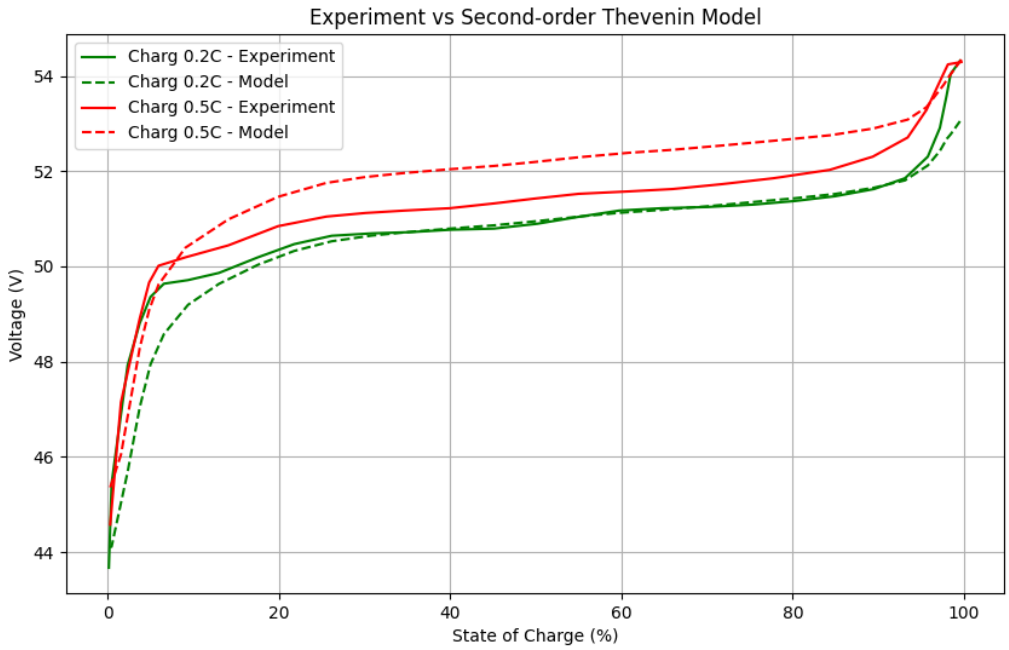


Fig 14. LFP voltage and model predicted voltage values as a function of SOC during battery charging.

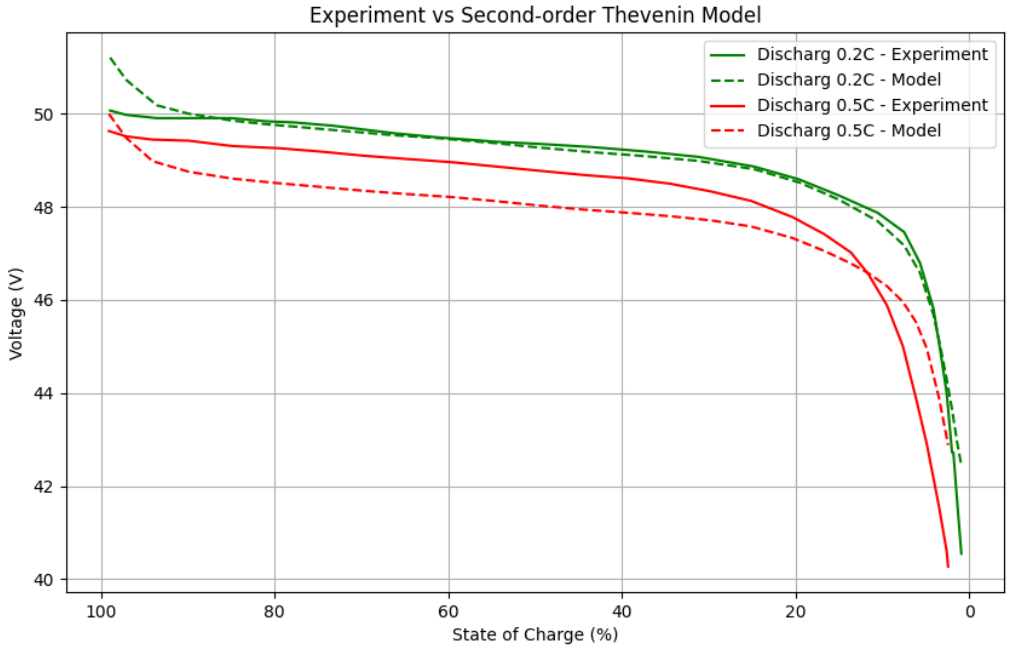


Fig 15. LFP battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 9. Absolute and relative errors between model-predicted and LFP battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.469	0.94
Charging	0.5	0.608	1.20
Discharging	0.2	0.386	0.88
Discharging	0.5	0.870	1.93

3.1.4 PNGV model

The PNGV model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. Powell method gave the best minimum error function value. The optimal parameters found using the selected methods are presented in table 10. The model-predicted and experimental voltage values during charging are depicted in Figure 16. The model-predicted and experimental voltage values during discharge are depicted in Figure 17. The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the LFP battery voltage. Optimization methods resulted in different absolute and relative errors. This is presented in Table 11.

Table 10. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Cb [F]	Error Function Value [V^2]
L-BFGS-B	0.01029	0.00983	1000	2000	73.98
SLSQP	0.01032	0.00981	1000	2000	73.98
Powell	0.00707	0.00986	500	338	69.31

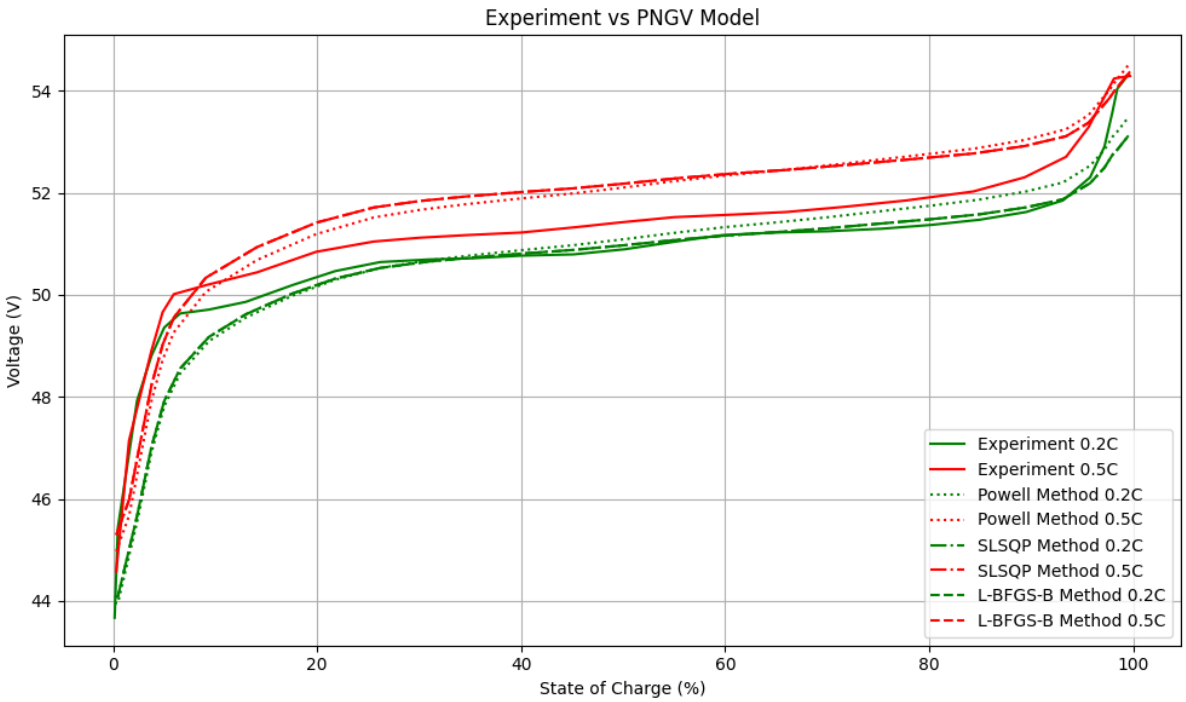


Fig 16. LFP battery voltage and model predicted voltage values as a function of SOC during battery charging.

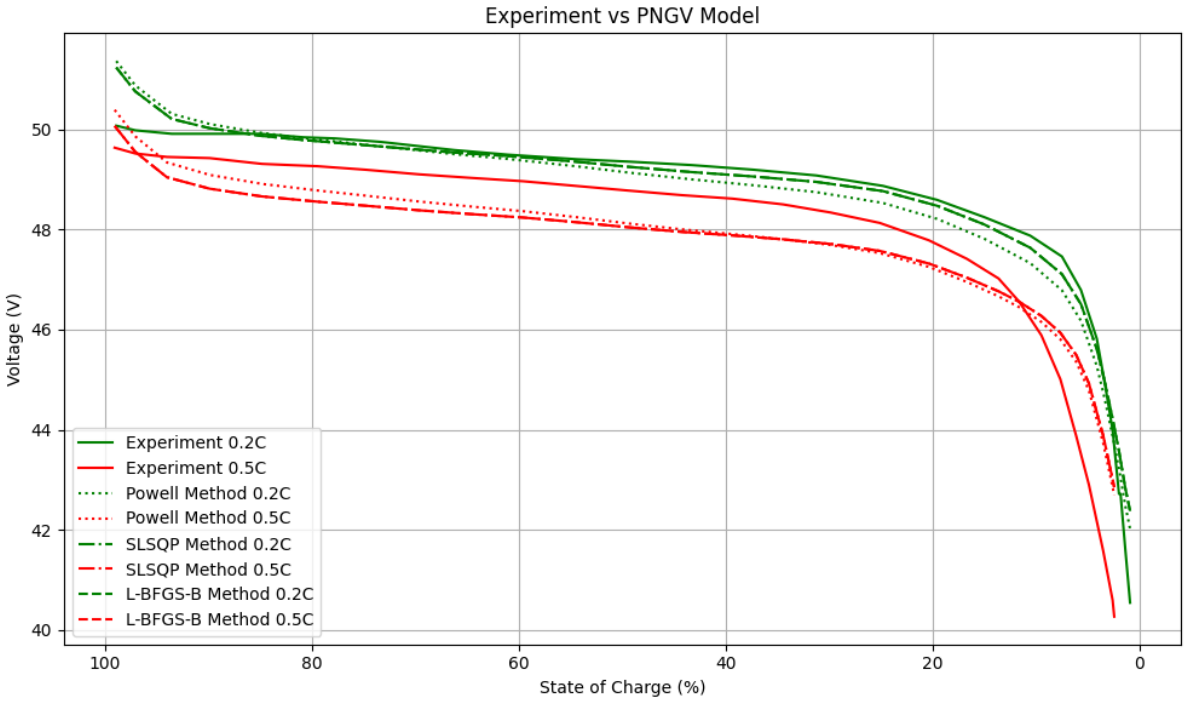


Fig 17. LFP battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 11. Absolute and relative errors between model-predicted and LFP battery voltages.

	Average absolute error (V)			Average relative error (%)		
Optimization method	L-BFGS-B	SLSQP	Powell	L-BFGS-B	SLSQP	Powell
Charging 0.2 C	0.470	0.470	0.529	0.95	0.95	1.06
Charging 0.5 C	0.605	0.605	0.603	1.19	1.19	1.19
Discharging 0.2 C	0.392	0.392	0.450	0.89	0.89	0.99
Discharging 0.5 C	0.853	0.853	0.786	1.89	1.89	1.75

3.2 NMC Battery

The postmortem analysis of commercial 18650 lithium-ion battery, presented by Weisenberger et al. [54], allowed obtaining the general parameters of battery cell (including 1C-rate current and cut-off voltages), as well as the parameters of the electrodes and separator (including thicknesses, active material fractions, porosities and particles radii). Electrolyte conductivity, diffusion coefficients, transference number and thermodynamic factor were taken from the article by Kremer et al. [55]. Conductivities of electrode materials, and equilibrium potential for electrode reactions s as taken from COMSOL Multiphysics materials database. The values of these parameters are shown in table A1, in appendix 1.

These parameters served as the input data for the electrochemical P2D model (see section 2.1.1), The details of the model and its implementation have been described by Jasielec and Peljo [56]. As an output of the model charge and discharge curves at C-rates of 0.2 and 0.5 were generated (1C rate corresponds to the current I = 2.6 A). To remove the effects of the capacity fade at high c-rates (described in detail in [55], [56]), curves were rescaled to cover the SOC range from 0 to 1. The OCV was taken from Comsol database, as difference between SOC-dependent potentials of cathode and anode.

3.2.1 Rint model

The Rint model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. The R0 parameter value was established to be 0.04458 Ω, and the model was used to calculate voltage values as a function of SOC. Error function value was 34.45 V^2. All methods gave the same minimum error function value and R0 value. The model-predicted and experimental voltage values during charging are depicted in Figure 18. The model-predicted and experimental voltage values during discharge are depicted in Figure 19. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the NMC battery voltage. This is presented in Table 12.

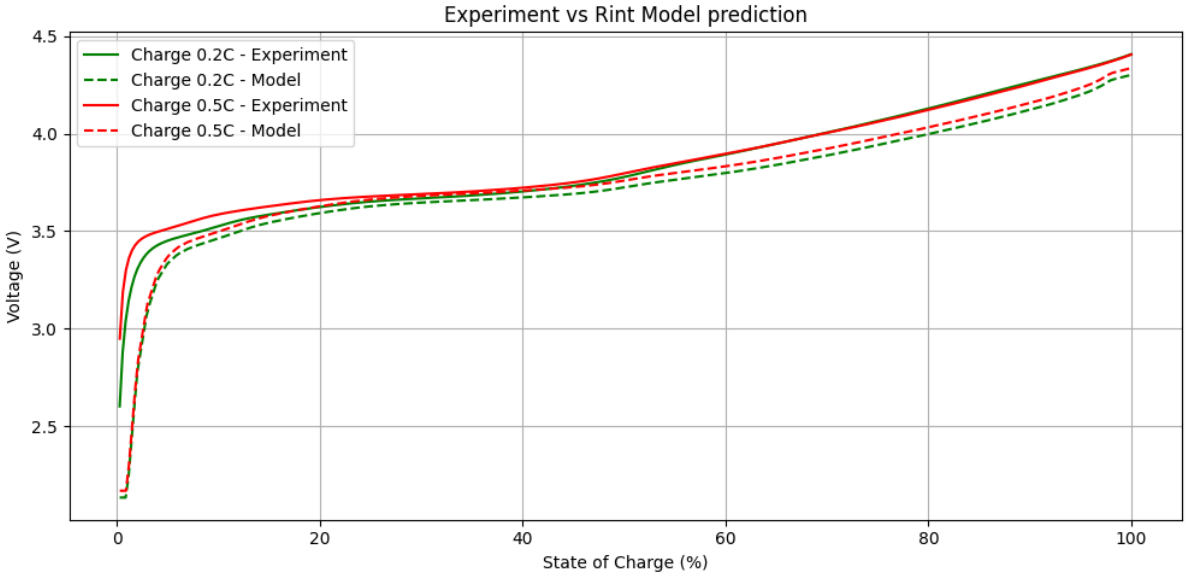


Fig 18. NMC battery voltage and model predicted voltage values as a function of SOC during battery charging.

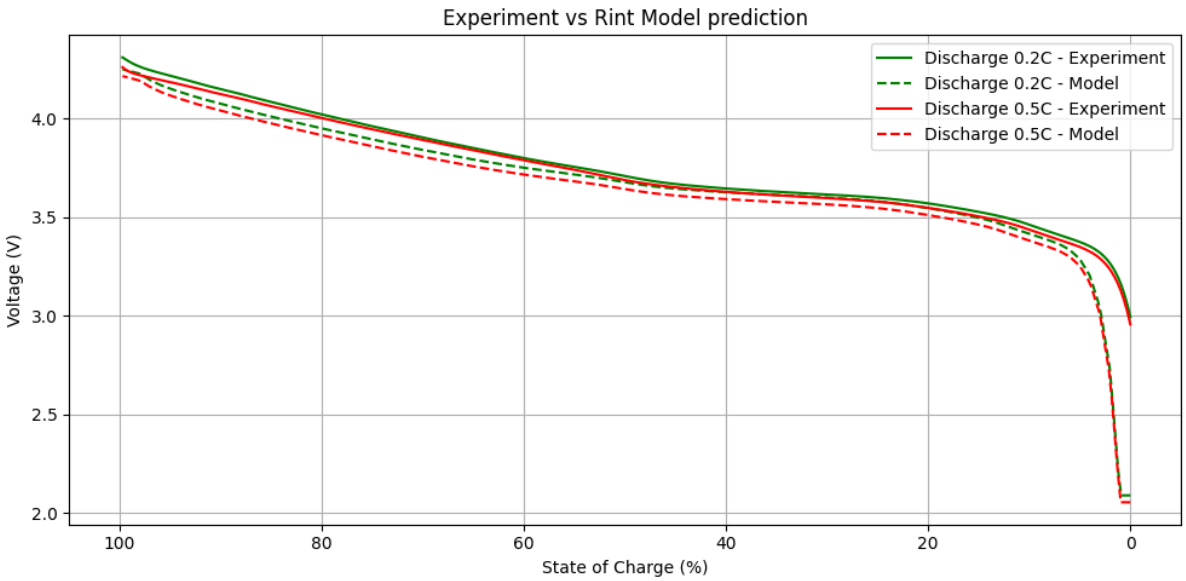


Fig 19. NMC battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 12. Absolute and relative errors between model-predicted and NMC battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.0958	2.55
Charging	0.5	0.0812	2.18
Discharging	0.2	0.0730	2.08
Discharging	0.5	0.0874	2.48

3.2.2 Thevenin model

The Thevenin model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. All methods gave the same minimum error function value but different optimized parameters. The optimal parameters found using the selected methods are presented in table 13. The model-predicted and experimental voltage values during charging are depicted in Figure 20. The model-predicted and experimental voltage values during discharge are depicted in Figure 21.

The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the NMC battery voltage. Optimization methods resulted in the same absolute and relative errors, as well as the same plotted curves. This is presented in table 14.

Table 13. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Error Function Value [V^2]
L-BFGS-B	0.01944	0.02513	200.00	34.45
SLSQP	0.01631	0.02827	500.00	34.45
Powell	0.04258	0.00200	200.00	34.45

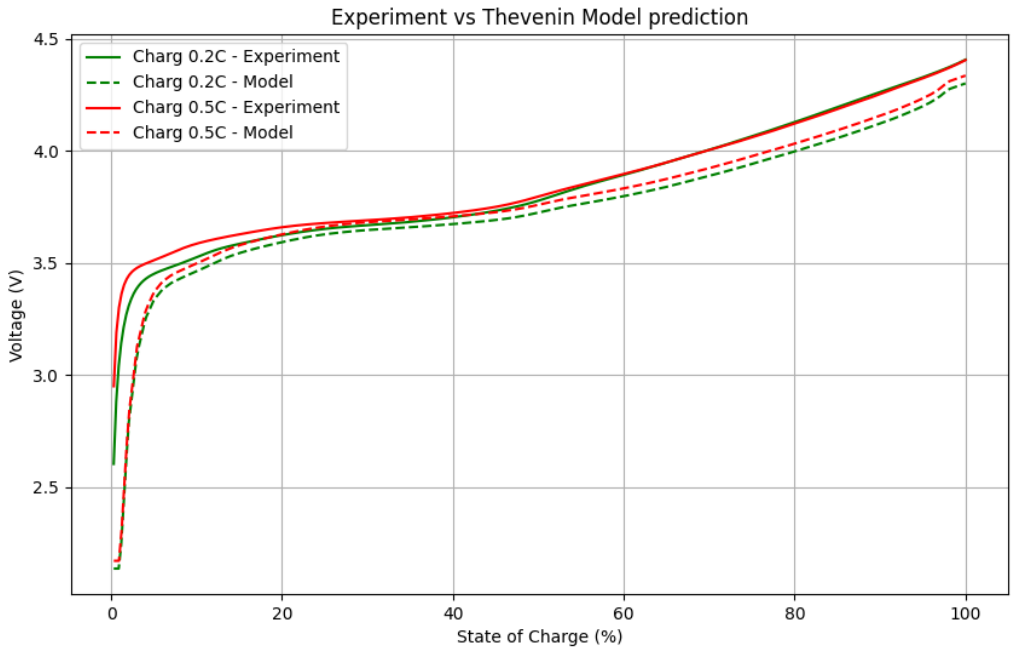


Fig 20. NMC battery voltage and model predicted voltage values as a function of SOC during battery charging.

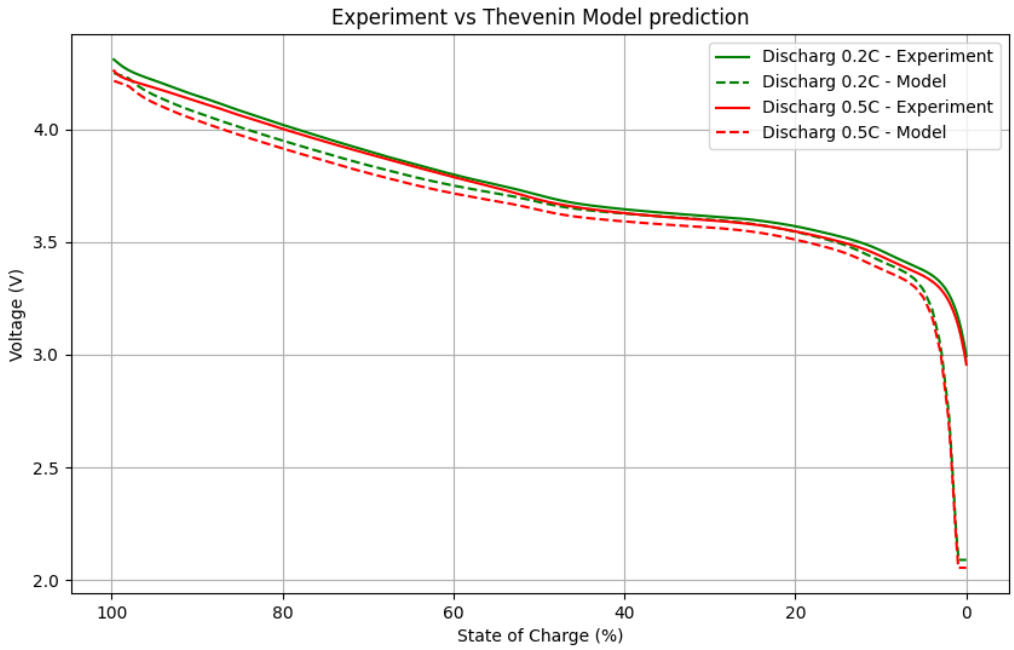


Fig 21. NMC battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 14. Absolute and relative errors between model-predicted and NMC battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.0958	2.55
Charging	0.5	0.0812	2.18
Discharging	0.2	0.0730	2.08
Discharging	0.5	0.0874	2.48

3.2.3 Second order Thevenin model

The Thevenin model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. All methods gave the same minimum error function value but different optimized parameters. The optimal parameters found using the selected methods are presented in table 15. The model-

predicted and experimental voltage values during charging are depicted in Figure 22. The model-predicted and experimental voltage values during discharge are depicted in Figure 23. The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the NMC battery voltage. Optimization methods resulted in the same absolute and relative errors, as well as the same plotted curves. This is presented in Table 16.

Table 15. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Rs2 [Ω]	Cs2 [F]	Error Function Value [V^2]
L-BFGS-B	0.01458	0.01588	3000	0.01411	3000	34.45
SLSQP	0.01885	0.01191	800	0.01382	1500	34.45
Powell	0.03658	0.00500	10000	0.00300	10000	34.45

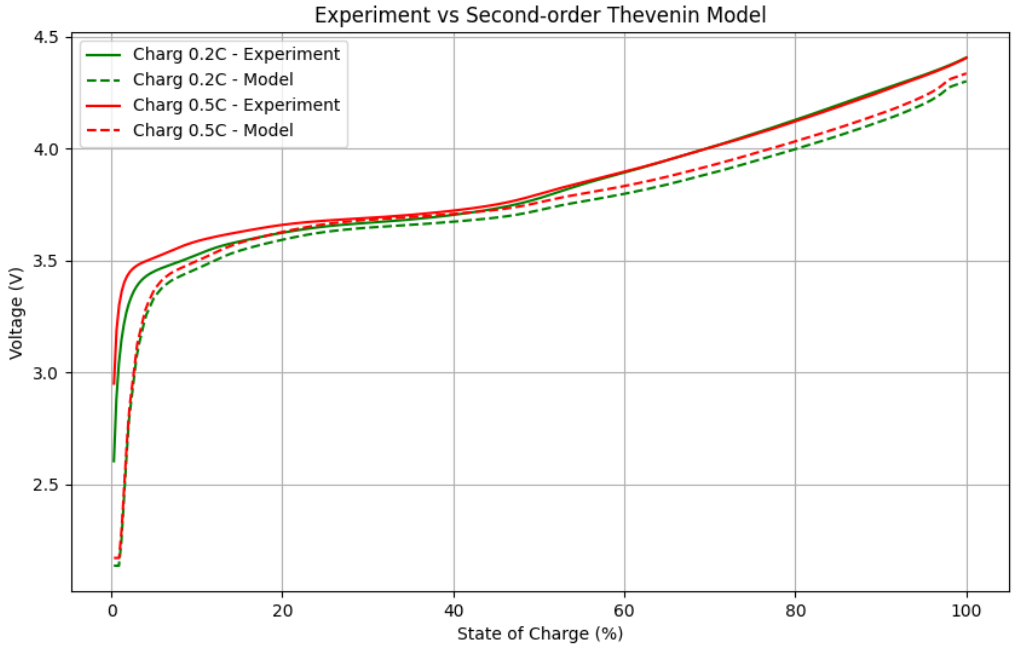


Fig 22. NMC battery voltage and model predicted voltage values as a function of SOC during battery charging.

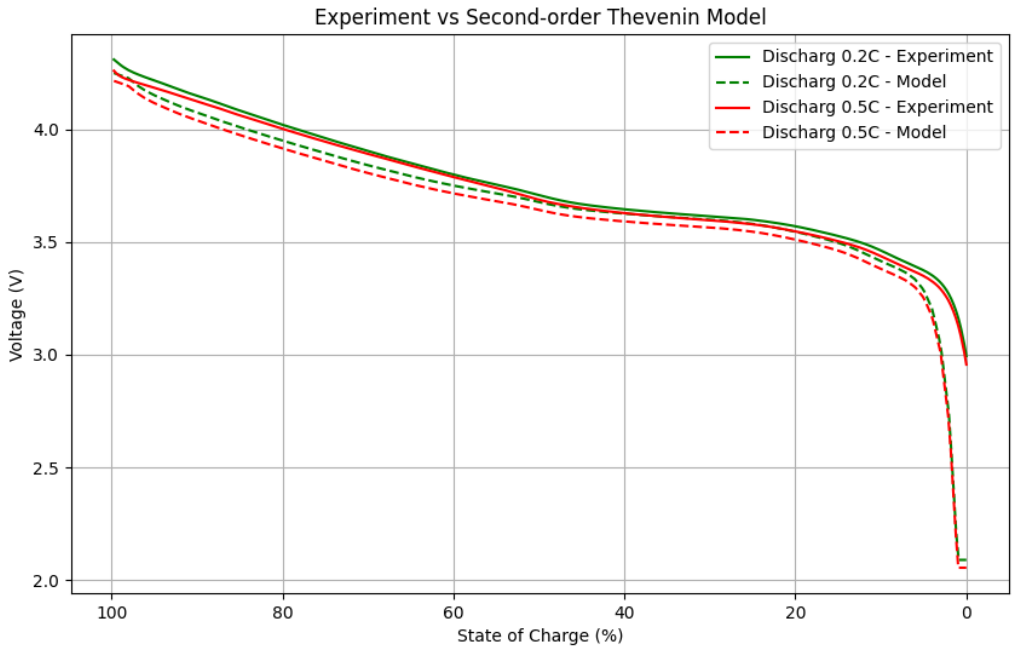


Fig 23. NMC battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 16. Absolute and relative errors between model-predicted and NMC battery voltages.

Charging / Discharging	C-rate	Average absolute error (V)	Average relative error (%)
Charging	0.2	0.0958	2.55
Charging	0.5	0.0812	2.18
Discharging	0.2	0.0730	2.08
Discharging	0.5	0.0874	2.48

3.2.4 PNGV model

The PNGV model was optimized at C-rates of 0.2 and 0.5 during charging and discharging. Powell method gave the best minimum error function value. The optimal parameters found

using the selected methods are presented in table 17. The model-predicted and experimental voltage values during charging are depicted in Figure 24. The model-predicted and experimental voltage values during discharge are depicted in Figure 25. The graphs have been plotted using the parameters given by the Powell optimization method. The average absolute and relative error were calculated from the difference between the model-predicted voltage and the NMC battery voltage. Optimization methods resulted in different absolute and relative errors. This is presented in Table 18.

Table 17. Optimized parameters with different optimization methods.

	R0 [Ω]	Rs [Ω]	Cs [F]	Cb [F]	Error Function Value [V^2]
L-BFGS-B	0.01937	0.02496	200	5000	34.46
SLSQP	0.01937	0.02496	200	5000	34.46
Powell	0.03446	0.01000	500	10000	34.45

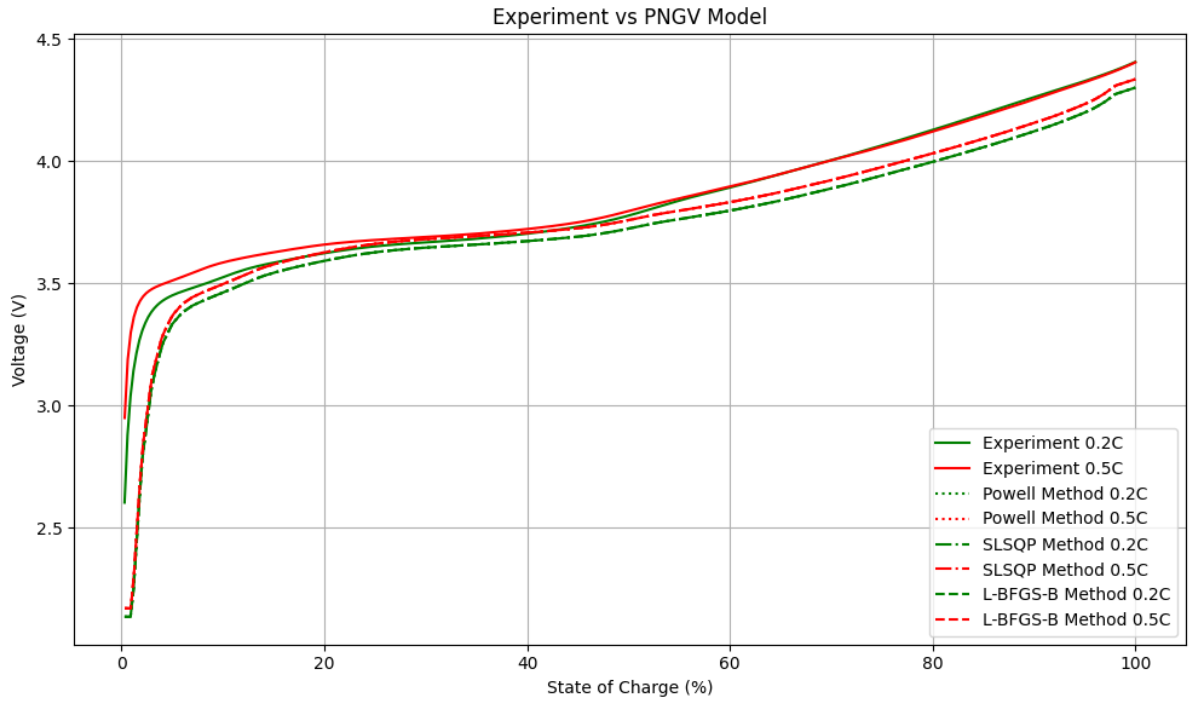


Fig 24. NMC battery voltage and model predicted voltage values as a function of SOC during battery charging.

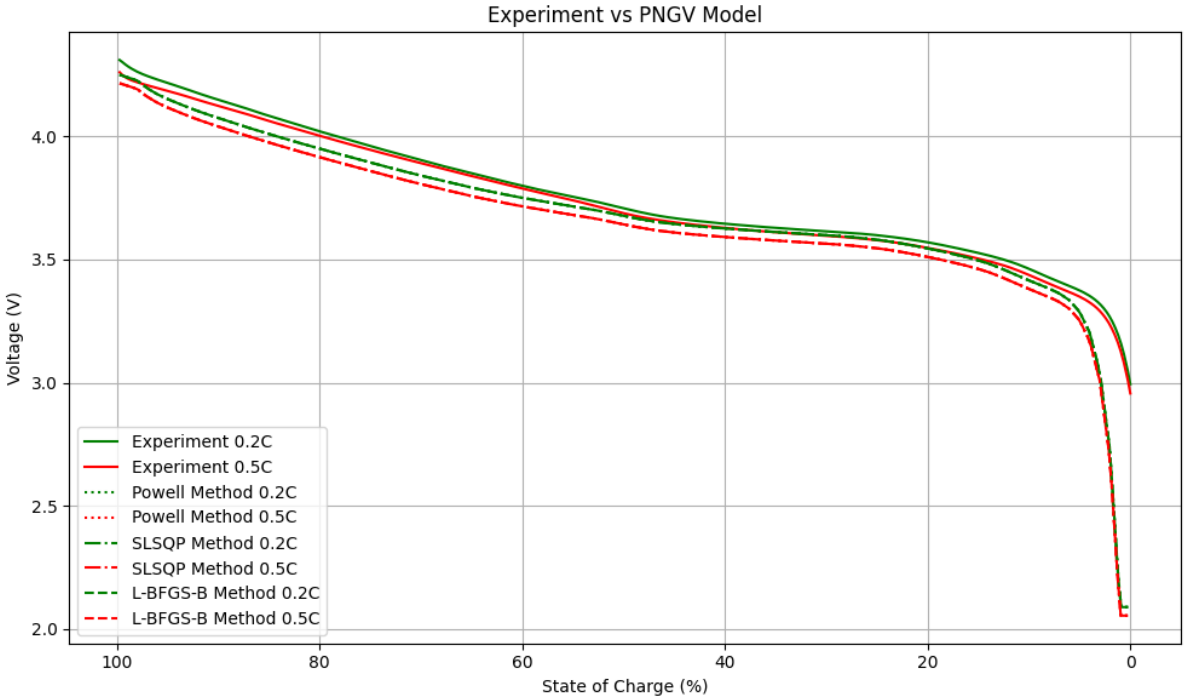


Fig 25. NMC battery voltage and model-predicted voltage values as a function of SOC during battery discharge.

Table 18. Absolute and relative errors between model-predicted and NMC battery voltages.

	Average absolute error (V)			Average relative error (%)		
Optimization method	L-BFGS-B	SLSQP	Powell	L-BFGS-B	SLSQP	Powell
Charging 0.2 C	0.0956	0.0956	0.0957	2.54	2.54	2.55
Charging 0.5 C	0.0812	0.0812	0.0812	2.18	2.18	2.18
Discharging 0.2 C	0.0732	0.0732	0.0731	2.08	2.08	2.08
Discharging 0.5 C	0.0874	0.0874	0.0874	2.48	2.48	2.48

4 Discussion

The results show that all four equivalent circuit models are able to accurately estimate the change in voltage of a LFP and NMC battery as a function of SOC. As can be seen from the results, the models were able to more accurately predict the voltage change of the LFP battery as a function of SOC than for the NMC battery. For LFP battery the average relative error of all models was approximately 1.05% lower than for NMC battery. The models are probably better at modeling LFP battery chemistry because LFP batteries have a flatter OCV-SOC curve, especially in the mid-range of SOC. ECMs rely on OCV vs. SOC mapping, a flatter curve inherently leads to less voltage deviation from the real behavior. The biggest error between the model and the experiment battery voltage is at low and high SOC values during both charging and discharging for both battery chemistries. This indicates that the models work better when the common 20/80 rule is applied, namely battery is exploited only at the SOC between 20 % and 80 %.

The differential equations present in Thevenin model, second order Thevenin model and PNGV model can be solved either using basic Euler step or the analytical solution for a RC circuits. Comparison of these two approaches showed no difference in the optimization results. This indicates that in our data the distance between the data points is small enough for the Euler method to be stable.

Several initial guesses and bounds were tried for each model and method to find the minima of the error function. Even after numerous iterations, methods did not find the minima with the lower value of error. However, with extending the bounds, the computational time became excessively large.

All three optimization located minima with nearly identical error function values across both battery chemistries. However, variations emerged in the model parameters, primarily due to differences in the optimization approaches. The most notable discrepancy appeared in the LFP battery, where, within the PNGV model, the Powell method found a minimum with lower error function value than the other two methods. This is likely because the Powell method is less prone to getting stuck on initial guesses or bounds.

For both battery chemistries, the error function for Rint has a single minimum, leading all optimization methods to produce the same error function value and identical R0 parameter value. In contrast, Thevenin and second order Thevenin exhibit at least three global minima, resulting in optimization methods yielding various optimal parameters in both battery chemistries. When applying the PNGV model to the LFP battery, the error function presents at least two local minima and one global minimum. As a result, L-BFGS-B and SLSQP found two different local minima with different parameter sets, while the Powell located the global minimum. For the NMC battery, one local minimum was identified, meaning L-BFGS-B and SLSQP found identical optimal parameters. Global minimum was found using the Powell method.

On one hand some of these results were expected. The SLSQP optimization method, was observed to get stuck in local minima. This is in accordance with the results of Gong et al. [48], who has shown that this method can yield many local optimal solutions. On the other hand, the Powell method found a global minimum for both battery chemistries. This surprising result contradicts the observation of Ge et al. [52] that this function has potential for getting stuck in local optima.

It is also noteworthy that, on a practical level, the Rint model is able to estimate voltage with the same accuracy as the more complex models. All four battery models closely matched the actual battery voltage output during the initial tests, achieving an average relative error of just 1.24 % for LFP battery and 2.32 % for NMC battery. This is comparable with the results of Sessa et al. [27], who used Thevenin second order model for battery-flywheel hybrid storage system to identify proper strategy to limit battery's aging effect, achieving an average error of just 0.55%.

Since all four equivalent circuit models were able to accurately describe the behavior of LFP and NMC batteries, these models could be used in real-life systems, such as BMS for both EVs and stationary storage. However, more information about the conditions is needed and the models should also be updated to be dependent on external factors such as temperature and other operating conditions.

5 Summary and conclusions

The motivation for this thesis is based on the growing importance of batteries in applications ranging from electric vehicles to stationary energy storage, and the need for reliable, simple, yet accurate models for battery management systems (BMS).

This thesis presents a comprehensive study on modeling of LIBs. Four classes of battery models were introduced and reviewed: electrochemical models that are highly accurate but computationally demanding, data-driven models that offer flexibility and learning capabilities but require large datasets, Empirical/semiempirical aging models that are suited for degradation analysis but tend to oversimplify the aging process of LIBs and the correlation between stress factors, and finally equivalent circuit models (ECMs) that offer a good trade-off between simplicity and accuracy for real-time applications.

The work focuses on ECMs, and their optimization for two widely used battery chemistries: LFP and NMC. Four ECMs were analyzed, namely: Rint model, Thevenin model, second order Thevenin model and PNGV model. Each model was tested on real charge-discharge data for LFP and NMC batteries at varying C-rates. Parameter optimization was conducted using three algorithms: L-BFGS-B, SLSQP, and Powell's method, with a least squares error function defined to minimize voltage prediction error as a function of SOC.

The results show that all four equivalent circuit models are able to accurately describe the change in voltage of a LFP and NMC battery as a function of SOC. However, the models were better suited for modeling the voltage of the LFP battery as a function of SOC than the NMC battery. All three optimization methods found minima with almost the same error function values for each battery chemistry.

On a practical level, the Rint model is able to estimate voltage with the same accuracy as the more complex models. Hence, with these types of LIBs, Rint model should be used, especially in real time applications, because the computational time is lower.

References

[1] F. Larsson, P. Andersson, and B.-E. Mellander, “Are electric vehicles safer than combustion engine vehicles?,” in *Systems Perspectives on Electromobility* , Gothenburg: Chalmers University of Technology, 2017, pp. 34–38.

[2] A. R. Dehghani-Sanij, E. Tharumalingam, M. B. Dusseault, and R. Fraser, “Study of energy storage systems and environmental challenges of batteries,” Apr. 01, 2019, *Elsevier Ltd*. doi: 10.1016/j.rser.2019.01.023.

[3] X. Chen, W. Shen, T. T. Vo, Z. Cao, and A. Kapoor, “An overview of lithium-ion batteries for electric vehicles,” in *2012 10th International Power & Energy Conference (IPEC)*, 2012, pp. 230–235. doi: 10.1109/ASSCC.2012.6523269.

[4] Y. Wang *et al.*, “Lithium and lithium ion batteries for applications in microelectronic devices: A review,” Jul. 15, 2015, *Elsevier*. doi: 10.1016/j.jpowsour.2015.03.164.

[5] A. Ferrese, “Battery Fundamentals,” *GetMobile: Mobile Comp. and Comm.*, vol. 19, no. 3, pp. 29–32, Dec. 2015, doi: 10.1145/2867070.2867082.

[6] M. Tomasov, M. Kajanova, P. Bracinik, and D. Motyka, “Overview of battery models for sustainable power and transport applications,” in *Transportation Research Procedia*, Elsevier B.V., 2019, pp. 548–555. doi: 10.1016/j.trpro.2019.07.079.

[7] M. Shen and Q. Gao, “A review on battery management system from the modeling efforts to its multiapplication and integration,” Aug. 01, 2019, *John Wiley and Sons Ltd*. doi: 10.1002/er.4433.

[8] M. R. Jongerden and B. R. Haverkort, “Battery Modeling.”

[9] W. Zhou, Y. Zheng, Z. Pan, and Q. Lu, “Review on the battery model and SOC estimation method,” Sep. 01, 2021, *MDPI*. doi: 10.3390/pr9091685.

[10] A. Jokar, B. Rajabloo, M. Désilets, and M. Lacroix, “Review of simplified Pseudo-two-Dimensional models of lithium-ion batteries,” Sep. 30, 2016, *Elsevier B.V.* doi: 10.1016/j.jpowsour.2016.07.036.

[11] M. Doyle, T. F. Fuller, and J. Newman, “Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell,” *J Electrochem Soc*, vol. 140, no. 6, p. 1526, Jun. 1993, doi: 10.1149/1.2221597.

[12] B. Balagopal and M. Y. Chow, “Effect of anode conductivity degradation on the Thevenin Circuit Model of lithium ion batteries,” in *IECON Proceedings (Industrial Electronics Conference)*, IEEE Computer Society, Dec. 2016, pp. 2028–2033. doi: 10.1109/IECON.2016.7793429.

[13] “Monolithic Power Systems, ‘Electrochemical Models,’ Monolithic Power Systems, [Online]. Available: https://www.monolithicpower.com/en/learning/mpscholar/battery-management-systems/battery-modeling/electrochemical-models. [Accessed: 03-Nov-2024].”

[14] T. S. Dao, C. P. Vyasarayani, and J. McPhee, “Simplification and order reduction of lithium-ion battery model based on porous-electrode theory,” *J Power Sources*, vol. 198, pp. 329–337, Jan. 2012, doi: 10.1016/j.jpowsour.2011.09.034.

[15] J. Li, K. Adewuyi, N. Lotfi, R. G. Landers, and J. Park, “A single particle model with chemical/mechanical degradation physics for lithium ion battery State of Health (SOH) estimation,” *Appl Energy*, vol. 212, pp. 1178–1190, Feb. 2018, doi: 10.1016/j.apenergy.2018.01.011.

[16] V. Lucaferri, M. Quercio, A. Laudani, and F. Riganti Fulginei, “A Review on Battery Model-Based and Data-Driven Methods for Battery Management Systems,” Dec. 01, 2023, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/en16237807.

[17] P. Domingos, “A few useful things to know about machine learning,” Oct. 2012. doi: 10.1145/2347736.2347755.

[18] M. Jordan, J. Kleinberg, and B. Schölkopf, “Pattern Recognition and Machine Learning.”

[19] H. D. Khalaf Jabbar Rafiqul Zaman Khan, *METHODS TO AVOID OVER-FITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY)*. 2015.

[20] A. and C. J. Montesinos López Osval Antonio and Montesinos López, “Overfitting, Model Tuning, and Evaluation of Prediction Performance,” in *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, Cham: Springer International Publishing, 2022, pp. 109–139. doi: 10.1007/978-3-030-89010-0_4.

[21] D. N. T. How, M. A. Hannan, M. S. Hossain Lipu, and P. J. Ker, “State of Charge Estimation for Lithium-Ion Batteries Using Model-Based and Data-Driven Methods: A Review,” 2019, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2019.2942213.

[22] V. Kecman, “Support Vector Machines – An Introduction,” in *Support Vector Machines: Theory and Applications*, vol. 177, 2005, pp. 1–47. doi: 10.1007/10984697_1.

[23] W. Vermeer, G. R. Chandra Mouli, and P. Bauer, “A Comprehensive Review on the Characteristics and Modeling of Lithium-Ion Battery Aging,” *IEEE Transactions on Transportation Electrification*, vol. 8, no. 2, pp. 2205–2232, Jun. 2022, doi: 10.1109/TTE.2021.3138357.

[24] P. M. Beulah. Devamalar, V. Thulasi. Bai, and M. . Moorthi, *2016 2nd International Conference on Advances in Electrical & Electronics, Information, Communication and Bio Informatics : AEEICB - 2016 : 27th & 28th February*. IEEE, 2016.

[25] S. K. K, P. Pathiyil, and R. Sunitha, “Generic Battery model covering self-discharge and internal resistance variation,” in *2016 IEEE 6th International Conference on Power Systems (ICPS)*, 2016, pp. 1–5. doi: 10.1109/ICPES.2016.7584003.

[26] C. Antaloae, J. Marco, and F. Assadian, “A Novel Method for the Parameterization of a Li-Ion Cell Model for EV/HEV Control Applications,” *IEEE Trans Veh Technol*, vol. 61, no. 9, pp. 3881–3892, 2012, doi: 10.1109/TVT.2012.2212474.

[27] S. D. Sessa, A. Tortella, M. Andriollo, and R. Benato, “Li-ion battery-flywheel hybrid storage system: Countering battery aging during a grid frequency regulation service,” *Applied Sciences (Switzerland)*, vol. 8, no. 11, Nov. 2018, doi: 10.3390/app8112330.

[28] R. Benato, S. D. Sessa, and F. Bevilacqua, “Measurement-Based Lithium-Manganese Oxide Battery Model.”

[29] X. Zhang, W. Zhang, and G. Lei, “A review of li-ion battery equivalent circuit models,” Dec. 01, 2016, *Korean Institute of Electrical and Electronic Material Engineers*. doi: 10.4313/TEEM.2016.17.6.311.

[30] S. Saxena, S. R. Raman, B. Saritha, and V. John, “A novel approach for electrical circuit modeling of Li-ion battery for predicting the steady-state and dynamic I–V characteristics,” *Sadhana - Academy Proceedings in Engineering Sciences*, vol. 41, no. 5, pp. 479–487, May 2016, doi: 10.1007/s12046-016-0486-7.

[31] L. Serrao, “An Aging Model of Ni-MH Batteries for Hybrid Electric Vehicles.”

[32] H. Dai, B. Jiang, and X. Wei, “Impedance characterization and modeling of lithium-ion batteries considering the internal temperature gradient,” *Energies (Basel)*, vol. 11, no. 1, 2018, doi: 10.3390/en11010220.

[33] M. Greenleaf, H. Li, and J. P. Zheng, “Application of physical electric circuit modeling to characterize Li-ion battery electrochemical processes,” *J Power Sources*, vol. 270, pp. 113–120, Dec. 2014, doi: 10.1016/j.jpowsour.2014.07.083.

[34] S. Soylu, *Electric Vehicles*. Rijeka: IntechOpen, 2011. doi: 10.5772/958.

[35] Sheldon S. et al, “Electrical Modeling of Renewable Energy Sources and Energy Storage Devices,” *Journal of Power Electronics*, vol. 4, no. 2, pp. 117–126, Apr. 2004.

[36] R. R. Thakkar, “Electrical Equivalent Circuit Models of Lithium-ion Battery,” in *Management and Applications of Energy Storage Devices*, K. E. Okedu, Ed., Rijeka: IntechOpen, 2021, ch. 1. doi: 10.5772/intechopen.99851.

[37] K. Huang, Y. Wang, and J. Feng, “Research on equivalent circuit Model of Lithium-ion battery for electric vehicles,” in *Proceedings - 2020 3rd World Conference on Mechanical Engineering and Intelligent Manufacturing, WCMEIM 2020*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 492–496. doi: 10.1109/WCMEIM52463.2020.00109.

[38] X. Ding, D. Zhang, J. Cheng, B. Wang, and P. C. K. Luk, “An improved Thevenin model of lithium-ion battery with high accuracy for electric vehicles,” *Appl Energy*, vol. 254, Nov. 2019, doi: 10.1016/j.apenergy.2019.113615.

[39] H. W. C. YE. Yuanjin ZHANG, “Estimation of the SOC of a battery based on the AUKF-BP algorithm[J].,” *Energy Storage Science and Technology, 2021*, vol. 10, no. 1, pp. 237–241, 2021, doi: 10.19799/j.cnki.2095-4239.2020.0285.

[40] C. Sheng *et al.*, “Energy management strategy based on health state for a PEMFC/Lithium-ion batteries hybrid power system,” Nov. 01, 2022, *Elsevier Ltd*. doi: 10.1016/j.enconman.2022.116330.

[41] M. A. Hannan, M. S. H. Lipu, A. Hussain, and A. Mohamed, “A review of lithium-ion battery state of charge estimation and management system in electric vehicle applications: Challenges and recommendations,” 2017, *Elsevier Ltd*. doi: 10.1016/j.rser.2017.05.001.

[42] X. Guo, X. Xu, J. Geng, X. Hua, Y. Gao, and Z. Liu, “SOC estimation with an adaptive unscented Kalman filter based on model parameter optimization,” *Applied Sciences (Switzerland)*, vol. 9, no. 19, Oct. 2019, doi: 10.3390/app9194177.

[43] X. Guo, X. Xu, J. Geng, X. Hua, Y. Gao, and Z. Liu, “SOC estimation with an adaptive unscented Kalman filter based on model parameter optimization,” *Applied Sciences (Switzerland)*, vol. 9, no. 19, Oct. 2019, doi: 10.3390/app9194177.

[44] P. Li, “An improved PNGV modeling and SOC estimation for lithium iron phosphate batteries,” in *IOP Conference Series: Earth and Environmental Science*, Institute of Physics Publishing, Nov. 2017. doi: 10.1088/1755-1315/94/1/012012.

[45] L. Wan, “Improvement and simulation test of PNGV equivalent circuit model,” in *AIP Conference Proceedings*, American Institute of Physics Inc., Jan. 2019. doi: 10.1063/1.5089043.

[46] P. Lin, P. Jin, A. Zou, and Z. Wang, “Real-time identification of partnership for a new generation of vehicles battery model parameters based on the model reference adaptive system,” *Int J Energy Res*, vol. 45, no. 6, pp. 9351–9368, May 2021, doi: 10.1002/er.6465.

[47] C. Zhu and R. H. Byrd, “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization,” 1997.

[48] M. Gong, F. Zhao, S. Zeng, and C. Li, “An experimental study on local and global optima of linear antenna array synthesis by using the sequential least squares programming,” *Appl Soft Comput*, vol. 148, Nov. 2023, doi: 10.1016/j.asoc.2023.110859.

[49] A. Lasko, “SLSQP or Random? Python Optimization of Business Tasks – What is the Choice? Jul. 1, 2019. [Online]. Available: https://alasko.medium.com/slsqp-or-random-python-optimization-of-business-tasks-what-is-the-choice-6e7bd822aa7b. [Accessed: Jun. 2, 2025].”

[50] M. J. D. Powell, “An efficient method for finding the minimum of a function of several variables without calculating derivatives,” *Comput J*, vol. 7, no. 2, pp. 155–162, Jan. 1964, doi: 10.1093/comjnl/7.2.155.

[51] L. Ma, W. Chen, B. Li, Z. You, and Z. Chen, “Fast field calibration of MIMU based on the powell algorithm,” *Sensors (Switzerland)*, vol. 14, no. 9, pp. 16062–16081, Aug. 2014, doi: 10.3390/s140916062.

[52] G. Ge, Y. Pu, J. Zhang, and A. Ouyang, “Powell-Based Bat Algorithm for Solving Nonlinear Equations,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in*

Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, 2018, pp. 840–845. doi: 10.1007/978-3-319-95957-3_90.

[53] "DAXTROMN POWER, 'DAXTROMN 48V 100AH 200AH 10KWH $LiFePO_4$ Battery with RS485 and CAN Bus 48V 200AH $LiFePO_4$ Battery Pack 10KW Lithium Battery 6000+ Cycles Max 32 Parallel,' [Online]. Available: https://daxtromnpower.com/product/daxtromn-48v-100ah-200ah-10kwh-lifep04-battery-with-rs485-and-can-bus-48v-200ah-lifepo4-battery-pack-10kw-lithium-battery-6000-cycles-max-32-parallel/. [Accessed: Apr. 14, 2025].."

[54] C. Weisenberger, B. Meir, S. Röhler, D. K. Harrison, and V. Knoblauch, "A post-mortem study of commercial 18650 lithium-ion cells with LiNi0.5Co0.2Mn0.3O2//Graphite chemistry after prolonged cycling (> 7000 cycles) with low C-rates," *Electrochim Acta*, vol. 379, May 2021, doi: 10.1016/j.electacta.2021.138145.

[55] L. S. Kremer *et al.*, "Influence of the Electrolyte Salt Concentration on the Rate Capability of Ultra-Thick NCM 622 Electrodes," *Batter Supercaps*, vol. 3, no. 11, pp. 1172–1182, Nov. 2020, doi: 10.1002/batt.202000098.

[56] J. J. Jasielec and P. Peljo, "Limitations of Fast Charging of High Energy NMC-based Lithium-Ion Batteries: A Numerical Study," *Batter Supercaps*, vol. 6, no. 10, Oct. 2023, doi: 10.1002/batt.202300189.

Appendices

Appendix 1. Parameters for obtaining the NMC charge and discharge curves

Table A1. Difference between the parameters the parameters of commercial 18650 and the ones used in the model.

Brand		Commercial 18650 [54]	Model
General Parameters	i_1C (mAh/cm2)	3.61	3.61
	Emin (V)	3	3
	Emax (V)	4.05	4.3
Positive Electrode	**Material**	**NMC532**	**NMC532**
	L (µm)	70	70
	Eps_s	0.59	0.6
	Porosity (eps_l)	0.41	0.4
	Rp	~0.5	0.5
	Kappa [S/m]	?	100 *
Negative Electrode	**Material**	**Graphite**	**Graphite**
	L (µm)	72	72
	Eps_s	0.58	0.61
	Porosity (eps_l)	0.42	0.39
	Rp	~20	10
	Kappa [S/m]	?	100 *
Separator	**Material**	**?**	**3EC:7EMC ****
	L (µm)	~20	20
	Kappa [S/m]	?	**
	Concentration [M]	?	1

* Conductivity and Equilibrium Potential for electrodes as taken from COMSOL

** taken from Kremer et al. [55]

Eps_s, i.e. active material fraction is calculated from the articles as:

* from articles: (Specific_Capacity[Ah/m2] / L[m]) * Density[kg/m3] / Experimental_Charge_Density[Ah/kg]

* in the model: 1/((c0max-c0min)*F_const*L_pos/i_1C/1[h])

Porosity = 1 - eps_s

Appendix 2. RINT model optimization and plotting code.

```
import numpy as np
import scipy.optimize as opt
import os
import matplotlib.pyplot as plt

# === User Input for Chemistry ===
chemistry = input("Choose battery chemistry (NMC or LFP): ").strip().upper()
if chemistry not in ["NMC", "LFP"]:
    raise ValueError("Invalid chemistry. Please choose 'NMC' or 'LFP'.")

# === Configuration based on Chemistry ===
base_dir = chemistry #change path if data is in different directory than code
battery_capacity = 2.6 if chemistry == "NMC" else 200

OCV_file = os.path.join(base_dir, "OCV_data.txt")
charge_files = ["Charge_0.2C.txt", "Charge_0.5C.txt"]
discharge_files = ["Discharge_0.2C.txt", "Discharge_0.5C.txt"]

# === Load (X, Y) data ===
def load_XY_data(filepath, skip=1):
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File not found: {filepath}")
    X, Y = [], []
    with open(filepath, 'r') as file:
        for _ in range(skip):
            next(file)
        for line in file:
            try:
                x, y = map(float, line.split())
                X.append(x)
                Y.append(y)
            except ValueError:
                print(f"Skipping invalid line: {line.strip()}")
    return np.array(X), np.array(Y)

def OCV(SOC, X, Y):
    return np.interp(SOC, X, Y)

def rint_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0):
    U_ocv = OCV(SOC_vals, OCV_X, OCV_Y)
    return U_ocv - R0 * I

def error_function(params, charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity):
    R0 = params[0]
    if R0 < 0:
        return np.inf
```

```
    total_error = 0.0
    for C_rate, SOC_vals, U_actual in charge_data:
        I = -C_rate * battery_capacity
        U_model = rint_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0)
        total_error += np.sum((U_model - U_actual) ** 2)
    for C_rate, SOC_vals, U_actual in discharge_data:
        I = C_rate * battery_capacity
        U_model = rint_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0)
        total_error += np.sum((U_model - U_actual) ** 2)
    return total_error

def calculate_errors(U_actual, U_model):
    absolute_error = np.abs(U_actual - U_model)
    with np.errstate(divide='ignore', invalid='ignore'):
        relative_error = np.where(U_actual != 0, absolute_error / U_actual * 100, 0)
    return absolute_error, relative_error

def plot_charge_discharge(charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity, R0):
    plt.figure(figsize=(10, 5))
    plt.title("Experiment vs Rint Model prediction")
    for C_rate, SOC_vals, U_actual in charge_data:
        I = -C_rate * battery_capacity
        U_model = rint_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0)
        color_map = {0.2: 'green', 0.5: 'red'}
        color = color_map.get(round(C_rate, 1), 'black')
        plt.plot(SOC_vals * 100, U_actual, '-', color=color, label=f'Charge {C_rate}C -
Experiment')
        plt.plot(SOC_vals * 100, U_model, '--', color=color, label=f'Charge {C_rate}C -
Model')
        abs_error, rel_error = calculate_errors(U_actual, U_model)
        print(f"Charging {C_rate}C - Avg Absolute Error: {np.mean(abs_error):.4f} V, Avg
Relative Error: {np.mean(rel_error):.2f}%")
    plt.xlabel("State of Charge (%)")
    plt.ylabel("Voltage (V)")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.title("Experiment vs Rint Model prediction")
    for C_rate, SOC_vals, U_actual in discharge_data:
        I = C_rate * battery_capacity
        U_model = rint_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0)
        SOC_plot = SOC_vals[::-1] * 100
        U_actual_plot = U_actual[::-1]
        U_model_plot = U_model[::-1]
        color_map = {0.2: 'green', 0.5: 'red'}
        color = color_map.get(round(C_rate, 1), 'black')
```

```
        plt.plot(SOC_plot, U_actual_plot, '-', color=color, label=f'Discharge {C_rate}C -
Experiment')
        plt.plot(SOC_plot, U_model_plot, '--', color=color, label=f'Discharge {C_rate}C -
Model')
        abs_error, rel_error = calculate_errors(U_actual, U_model)
        print(f"Discharging {C_rate}C - Avg Absolute Error: {np.mean(abs_error):.4f} V, Avg
Relative Error: {np.mean(rel_error):.2f}%")
    plt.xlabel("State of Charge (%)")
    plt.ylabel("Voltage (V)")
    plt.xlim(105, -5)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

# === Load Data ===
OCV_X, OCV_Y = load_XY_data(OCV_file)

charge_data = []
discharge_data = []

for filename in charge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        charge_data.append((C_rate, SOC, U))

for filename in discharge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if chemistry == "LFP":
        SOC = 1 - SOC  # Flip for LFP
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        discharge_data.append((C_rate, SOC, U))

# === Optimization with Multiple Methods ===
initial_guesses = [[0.010], [0.015], [0.020]]
bounds_list = [[(0.001, 0.05)], [(0.005, 0.05)], [(0.001, 0.05)]]
optimization_methods = ["L-BFGS-B", "SLSQP", "Powell"]
all_results = []

for method in optimization_methods:
    print(f"\n=== Trying Optimization Method: {method} ===")
    best_result = None
    best_error = float("inf")
    best_params = None
```

```
    for x0 in initial_guesses:
        for bounds in bounds_list:
            result = opt.minimize(
                error_function,
                x0=x0,
                args=(charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity),
                bounds=bounds if method != "Powell" else None,
                method=method
            )
            if result.success and result.fun < best_error:
                best_result = result
                best_error = result.fun
                best_params = result.x

    if best_result:
        all_results.append((method, best_error, best_params))
        print(f"{method} | R0: {best_params[0]:.5f} Ω | Error: {best_error:.5f}")
    else:
        print(f"{method} optimization failed!")

# === Final Evaluation ===
def all_methods_agree(results, tol=1e-6):
    ref_error, ref_params = results[0][1], results[0][2]
    for _, err, params in results[1:]:
        if not np.isclose(err, ref_error, atol=tol):
            return False
        if not np.allclose(params, ref_params, atol=tol):
            return False
    return True

if all_results:
    all_results.sort(key=lambda x: x[1])  # Sort by error
    best_method, best_error, best_params = all_results[0]

    if all_methods_agree(all_results):
        print("\n✅ All optimization methods give the same results.")
    else:
        print(f"\n✅ Best optimization method: {best_method} with error: {best_error:.5f}")

    R0_opt = best_params[0]
    print(f"Optimized R0: {R0_opt:.5f} Ω")
    plot_charge_discharge(charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity,
R0_opt)
else:
    raise RuntimeError("❌ Optimization failed for all methods!")
```

Appendix 3. Thevenin model optimization and plotting code.

```
import numpy as np
import scipy.optimize as opt
import os
import re
import matplotlib.pyplot as plt

# === Chemistry Input ===
chemistry = input("Choose battery chemistry (NMC or LFP): ").strip().upper()
if chemistry not in ["NMC", "LFP"]:
    raise ValueError("Invalid chemistry. Choose 'NMC' or 'LFP'.")

# === Configuration Based on Chemistry ===
base_dir = chemistry #change path if data is in different directory than code
OCV_file = os.path.join(base_dir, "OCV_data.txt")
battery_capacity = 2.6 if chemistry == "NMC" else 200  # Ah
flip_discharge_SOC = chemistry == "LFP"

# === Load (X, Y) data ===
def load_XY_data(filepath, skip=1):
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File not found: {filepath}")
    X, Y = [], []
    with open(filepath, 'r') as file:
        for _ in range(skip):
            next(file)
        for line in file:
            try:
                x, y = map(float, re.split(r'\s+', line.strip()))
                X.append(x)
                Y.append(y)
            except ValueError:
                print(f"Skipping invalid line: {line.strip()}")
    return np.array(X), np.array(Y)

# === OCV interpolation ===
def OCV(SOC, X, Y):
    return np.interp(SOC, X, Y)

# === Thevenin voltage model ===
def thevenin_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, C_rate):
    U_ocv = OCV(SOC_vals, OCV_X, OCV_Y)
    Us = Rs * I
    U_bat = []
    for i in range(len(SOC_vals)):
        if i == 0:
            u_model = U_ocv[i] - R0 * I - Us
```

```
            U_bat.append(u_model)
            continue
        delta_SOC = abs(SOC_vals[i] - SOC_vals[i - 1])
        dt = max(delta_SOC / abs(C_rate), 1e-6)
        tau = Rs * Cs
        alpha = np.exp(-dt / tau)
        Us = alpha * Us + Rs * I * (1 - alpha)
        u_model = U_ocv[i] - R0 * I - Us
        U_bat.append(u_model)
    return np.array(U_bat)

# === Error function ===
def error_function(params, charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity):
    R0, Rs, Cs = params
    if R0 < 0 or Rs < 0 or Cs <= 0:
        return np.inf
    total_error = 0.0
    for C_rate, SOC_vals, U_actual in charge_data:
        I = -C_rate * battery_capacity
        U_model = thevenin_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)
    for C_rate, SOC_vals, U_actual in discharge_data:
        I = C_rate * battery_capacity
        U_model = thevenin_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)
    return total_error

# === Load Data ===
OCV_X, OCV_Y = load_XY_data(OCV_file)
charge_files = ["Charge_0.2C.txt", "Charge_0.5C.txt"]
discharge_files = ["Discharge_0.2C.txt", "Discharge_0.5C.txt"]

charge_data = []
discharge_data = []

for filename in charge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        charge_data.append((C_rate, SOC, U))

for filename in discharge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if flip_discharge_SOC:
        SOC = 1 - SOC
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
```

```
        discharge_data.append((C_rate, SOC, U))

# === Optimization Loop ===
initial_guesses = [
    [0.010, 0.002, 200],
    [0.015, 0.010, 500],
    [0.020, 0.020, 1000],
]
bounds_list = [
    [(0.001, 0.05), (0.0001, 0.1), (1, 2000)],
    [(0.005, 0.05), (0.001, 0.2), (10, 3000)],
    [(0.001, 0.05), (0.001, 0.5), (10, 5000)],
]
methods = ["L-BFGS-B", "SLSQP", "Powell"]
all_results = []

for method in methods:
    print(f"\n=== Trying Method: {method} ===")
    best = None
    best_err = float("inf")
    for x0 in initial_guesses:
        for b in bounds_list:
            result = opt.minimize(
                error_function,
                x0=x0,
                args=(charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity),
                bounds=None if method == "Powell" else b,
                method=method
            )
            if result.success and result.fun < best_err:
                best = result
                best_err = result.fun
    if best:
        all_results.append((method, best_err, best.x))
        print(f"{method} | Error: {best_err:.5f} | Params: R0={best.x[0]:.5f},
Rs={best.x[1]:.5f}, Cs={best.x[2]:.2f}")
    else:
        print(f"{method} failed.")

if not all_results:
    raise RuntimeError("❌ Optimization failed for all methods!")

# === Plotting functions for multiple methods ===
def calculate_percentage_error(U_actual, U_model):
    return np.abs(U_actual - U_model) / np.maximum(np.abs(U_actual), 1e-3) * 100

def calculate_absolute_error(U_actual, U_model):
    return np.abs(U_actual - U_model)
```

```
def thevenin_voltage_plot(SOC, I, params, C_rate):
    return thevenin_voltage_model(SOC, OCV_X, OCV_Y, I, *params, C_rate)

def plot_comparison_multiple_methods(data, label, all_results, flip_xaxis=False):
    plt.figure(figsize=(10, 6))
    color_map = {0.2: 'green', 0.5: 'red'}
    method_colors = {'L-BFGS-B': 'blue', 'SLSQP': 'orange', 'Powell': 'purple'}

    # Sort data by C-rate: 0.2C first, then 0.5C
    data_sorted = sorted(data, key=lambda x: x[0])

    for C_rate, SOC_vals, U_actual in data_sorted:
        SOC_percent = SOC_vals * 100
        color = color_map.get(round(C_rate, 1), 'black')
        plt.plot(SOC_percent, U_actual, '-', color=color, linewidth=2,
                 label=f"Experiment {C_rate:.1f}C")

    for method, err, params in all_results:
        print(f"\n--- {label} | Method: {method} | Error: {err:.5f} ---")
        for C_rate, SOC_vals, U_actual in data_sorted:
            I = -C_rate * battery_capacity if label == "Charging" else C_rate *
battery_capacity
            U_model = thevenin_voltage_plot(SOC_vals, I, params, C_rate)
            SOC_percent = SOC_vals * 100
            method_color = method_colors.get(method, 'black')
            label_str = f"{method} Method {C_rate:.1f}C"  # 🔄 Changed 'Model' to 'Method'
            plt.plot(SOC_percent, U_model, '--', color=method_color, label=label_str)

            abs_error = calculate_absolute_error(U_actual, U_model)
            perc_error = calculate_percentage_error(U_actual, U_model)
            print(f"  C-rate {C_rate:.1f}C -> Avg Abs Error: {np.mean(abs_error):.4f} V,
Avg % Error: {np.mean(perc_error):.2f}%")

    plt.xlabel("State of Charge (%)")
    plt.ylabel("Voltage (V)")
    plt.title("Experiment vs Thevenin Model")
    plt.grid(True)
    plt.legend()
    if flip_xaxis:
        plt.gca().invert_xaxis()
    plt.xticks(np.arange(0, 110, 20))
    plt.tight_layout()
    plt.show()


# === Plot all methods' results for charge and discharge ===
plot_comparison_multiple_methods(charge_data, "Charging", all_results)
plot_comparison_multiple_methods(discharge_data, "Discharging", all_results,
flip_xaxis=True)
```

```
# === Model Output Consistency Check ===
print("\n🔍 Checking model output differences between methods:")
C_rate_test = 0.5
SOC_test, U_test = discharge_data[1][1], discharge_data[1][2]
model_outputs = []
for method, err, params in all_results:
    I = C_rate_test * battery_capacity
    U_model = thevenin_voltage_model(SOC_test, OCV_X, OCV_Y, I, *params, C_rate_test)
    model_outputs.append((method, U_model))

for i in range(len(model_outputs)):
    for j in range(i + 1, len(model_outputs)):
        method_i, Ui = model_outputs[i]
        method_j, Uj = model_outputs[j]
        max_diff = np.max(np.abs(Ui - Uj))
        print(f"Max voltage difference between {method_i} and {method_j}: {max_diff:.10f}
V")
```

Appendix 4. Second Order Thevenin model optimization and plotting code.

```
import numpy as np
import scipy.optimize as opt
import os
import re
import matplotlib.pyplot as plt

# === Chemistry Input ===
chemistry = input("Choose battery chemistry (NMC or LFP): ").strip().upper()
if chemistry not in ["NMC", "LFP"]:
    raise ValueError("Invalid chemistry. Choose 'NMC' or 'LFP'.")

# === Configuration Based on Chemistry ===
base_dir = chemistry #change path if data is in different directory than code
OCV_file = os.path.join(base_dir, "OCV_data.txt")
battery_capacity = 2.6 if chemistry == "NMC" else 200  # Ah
flip_discharge_SOC = chemistry == "LFP"

# === Load (X, Y) data ===
def load_XY_data(filepath, skip=1):
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File not found: {filepath}")
    X, Y = [], []
    with open(filepath, 'r') as file:
        for _ in range(skip):
            next(file)
        for line in file:
            try:
                x, y = map(float, re.split(r'\s+', line.strip()))
                X.append(x)
                Y.append(y)
            except ValueError:
                print(f"Skipping invalid line: {line.strip()}")
    return np.array(X), np.array(Y)

# === OCV interpolation ===
def OCV(SOC, X, Y):
    return np.interp(SOC, X, Y)

# === 2nd Order Thevenin Voltage Model ===
def thevenin_2nd_order_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs1, Cs1, Rs2, Cs2,
C_rate):
    U_ocv = OCV(SOC_vals, OCV_X, OCV_Y)
    Us1 = Rs1 * I
    Us2 = Rs2 * I
    U_bat = []
```

```
    for i in range(len(SOC_vals)):
        if i == 0:
            u_model = U_ocv[i] - R0 * I - Us1 - Us2
            U_bat.append(u_model)
            continue

        delta_SOC = abs(SOC_vals[i] - SOC_vals[i - 1])
        dt = max(delta_SOC / abs(C_rate), 1e-6)

        tau1 = Rs1 * Cs1
        alpha1 = np.exp(-dt / tau1)
        Us1 = alpha1 * Us1 + Rs1 * I * (1 - alpha1)

        tau2 = Rs2 * Cs2
        alpha2 = np.exp(-dt / tau2)
        Us2 = alpha2 * Us2 + Rs2 * I * (1 - alpha2)

        u_model = U_ocv[i] - R0 * I - Us1 - Us2
        U_bat.append(u_model)

    return np.array(U_bat)

# === Error Function for Optimizer (2nd Order) ===
def error_function_2nd(params, charge_data, discharge_data, OCV_X, OCV_Y,
battery_capacity):
    R0, Rs1, Cs1, Rs2, Cs2 = params
    if any(p < 0 for p in [R0, Rs1, Rs2]) or Cs1 <= 0 or Cs2 <= 0:
        return np.inf

    total_error = 0.0
    for C_rate, SOC_vals, U_actual in charge_data:
        I = -C_rate * battery_capacity
        U_model = thevenin_2nd_order_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs1, Cs1,
Rs2, Cs2, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)

    for C_rate, SOC_vals, U_actual in discharge_data:
        I = C_rate * battery_capacity
        U_model = thevenin_2nd_order_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs1, Cs1,
Rs2, Cs2, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)

    return total_error

# === Load Data ===
OCV_X, OCV_Y = load_XY_data(OCV_file)
charge_files = ["Charge_0.2C.txt", "Charge_0.5C.txt"]
discharge_files = ["Discharge_0.2C.txt", "Discharge_0.5C.txt"]
```

```
charge_data = []
discharge_data = []

for filename in charge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        charge_data.append((C_rate, SOC, U))

for filename in discharge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if flip_discharge_SOC:
        SOC = 1 - SOC
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        discharge_data.append((C_rate, SOC, U))

# === Initial guesses and bounds for optimization ===
initial_guesses_2nd = [
    [0.010, 0.005, 10000, 0.003, 10000],
    [0.020, 0.010, 500, 0.010, 1000],
    [0.015, 0.003, 800, 0.005, 1500],
    [0.005, 0.001, 100, 0.001, 200],
    [0.012, 0.007, 600, 0.004, 900]
]

bounds_list_2nd = [
    [(0.001, 0.05), (0.0001, 0.1), (10, 3000), (0.0001, 0.1), (10, 3000)],
    [(0.001, 0.1), (0.001, 0.2), (1, 5000), (0.001, 0.2), (1, 5000)],
    [(0.001, 0.05), (0.001, 0.2), (50, 2000), (0.001, 0.2), (50, 2000)]
]

# === Multi-method optimization ===
methods = ["L-BFGS-B", "SLSQP", "Powell"]
all_results = []

for method in methods:
    print(f"\n=== Trying Method: {method} ===")
    best = None
    best_err = float("inf")
    for x0 in initial_guesses_2nd:
        for b in bounds_list_2nd:
            result = opt.minimize(
                error_function_2nd,
                x0=x0,
                args=(charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity),
                bounds=None if method == "Powell" else b,
```

```
                method=method
            )
            if result.success and result.fun < best_err:
                best = result
                best_err = result.fun
    if best:
        all_results.append((method, best_err, best.x))
        R0, Rs1, Cs1, Rs2, Cs2 = best.x
        print(f"{method} | Error: {best_err:.5f} | Params: "
              f"R0={R0:.5f}, Rs1={Rs1:.5f}, Cs1={Cs1:.2f}, Rs2={Rs2:.5f}, Cs2={Cs2:.2f}")
    else:
        print(f"{method} failed.")

# === Check for identical results ===
def all_methods_agree(results, tol=1e-6):
    ref_err, ref_params = results[0][1], results[0][2]
    for _, err, params in results[1:]:
        if not np.isclose(err, ref_err, atol=tol):
            return False
        if not np.allclose(params, ref_params, atol=tol):
            return False
    return True

# === Final Result Selection ===
if all_results:
    all_results.sort(key=lambda x: x[1])
    best_method, best_error, best_params = all_results[0]

    if all_methods_agree(all_results):
        print("\n✅ All methods give the same result.")
    else:
        print(f"\n✅ Best method: {best_method} with error: {best_error:.5f}")

    R0_opt, Rs1_opt, Cs1_opt, Rs2_opt, Cs2_opt = best_params
    print(f"Optimized Parameters:\n"
          f"R0 = {R0_opt:.5f} Ω\n"
          f"Rs1 = {Rs1_opt:.5f} Ω\n"
          f"Cs1 = {Cs1_opt:.2f} F\n"
          f"Rs2 = {Rs2_opt:.5f} Ω\n"
          f"Cs2 = {Cs2_opt:.2f} F")
else:
    raise RuntimeError("❌ Optimization failed for all methods!")

# === Multi-method Plotting Functions for Second-Order Model ===
def thevenin_voltage_plot_2nd_multi(SOC, I, params, C_rate):
    return thevenin_2nd_order_voltage_model(SOC, OCV_X, OCV_Y, I, *params, C_rate)

def calculate_percentage_error(U_actual, U_model):
    return np.abs(U_actual - U_model) / np.maximum(np.abs(U_actual), 1e-3) * 100
```

```
def calculate_absolute_error(U_actual, U_model):
    return np.abs(U_actual - U_model)

def plot_comparison_multiple_methods(data, label, all_results, flip_xaxis=False):
    plt.figure(figsize=(10, 6))
    color_map = {0.2: 'green', 0.5: 'red'}
    method_colors = {'L-BFGS-B': 'blue', 'SLSQP': 'orange', 'Powell': 'purple'}

    added_labels = set()
    exp_labels_ordered = []
    model_labels_ordered = []

    for C_rate in sorted(set(d[0] for d in data)):
        # Plot experimental data
        for method, err, params in all_results:
            for cr, SOC_vals, U_actual in data:
                if cr != C_rate:
                    continue
                I = -C_rate * battery_capacity if label == "Charging" else C_rate *
battery_capacity
                U_model = thevenin_voltage_plot_2nd_multi(SOC_vals, I, params, C_rate)
                SOC_percent = SOC_vals * 100
                color = color_map.get(round(C_rate, 1), 'black')
                method_color = method_colors.get(method, 'black')

                # Plot experimental curve once per C-rate
                exp_label = f"Experiment {C_rate:.1f}C"
                if exp_label not in added_labels:
                    plt.plot(SOC_percent, U_actual, '-', color=color, alpha=0.7,
label=exp_label)
                    added_labels.add(exp_label)
                    exp_labels_ordered.append(exp_label)
                else:
                    plt.plot(SOC_percent, U_actual, '-', color=color, alpha=0.3)

                # Plot model prediction
                method_label = f"{method} Method {C_rate:.1f}C"
                if method_label not in added_labels:
                    plt.plot(SOC_percent, U_model, '--', color=method_color,
label=method_label)
                    added_labels.add(method_label)
                    model_labels_ordered.append(method_label)
                else:
                    plt.plot(SOC_percent, U_model, '--', color=method_color)

                abs_error = calculate_absolute_error(U_actual, U_model)
                perc_error = calculate_percentage_error(U_actual, U_model)
```

```
                print(f"  {method} {label} {C_rate:.1f}C -> Avg Abs Error:
{np.mean(abs_error):.4f} V, Avg % Error: {np.mean(perc_error):.2f}%")

    # Finalize plot
    plt.xlabel("State of Charge (%)")
    plt.ylabel("Voltage (V)")
    plt.title("Experiment vs Second order Thevenin Model")
    plt.grid(True)

    # Sort and apply legend
    handles, labels = plt.gca().get_legend_handles_labels()
    order = exp_labels_ordered + model_labels_ordered
    ordered_handles = [handles[labels.index(lbl)] for lbl in order]
    plt.legend(ordered_handles, order)

    if flip_xaxis:
        plt.gca().invert_xaxis()
    plt.xticks(np.arange(0, 110, 20))
    plt.tight_layout()
    plt.show()

# === Plot all methods' results for charge and discharge ===
plot_comparison_multiple_methods(charge_data, "Charging", all_results)
plot_comparison_multiple_methods(discharge_data, "Discharging", all_results,
flip_xaxis=True)

# === Model Output Consistency Check ===
print("\n🔍 Checking model output differences between methods:")
C_rate_test = 0.5
SOC_test, U_test = discharge_data[1][1], discharge_data[1][2]
model_outputs = []

for method, err, params in all_results:
    I = C_rate_test * battery_capacity
    U_model = thevenin_voltage_plot_2nd_multi(SOC_test, I, params, C_rate_test)
    model_outputs.append((method, U_model))

for i in range(len(model_outputs)):
    for j in range(i + 1, len(model_outputs)):
        method_i, Ui = model_outputs[i]
        method_j, Uj = model_outputs[j]
        max_diff = np.max(np.abs(Ui - Uj))
        print(f"Max voltage difference between {method_i} and {method_j}: {max_diff:.10f}
V")
```

Appendix 5. PNGV model optimization and plotting code.

```
import numpy as np
import scipy.optimize as opt
import os
import re
import matplotlib.pyplot as plt

# === Chemistry Input ===
chemistry = input("Choose battery chemistry (NMC or LFP): ").strip().upper()
if chemistry not in ["NMC", "LFP"]:
    raise ValueError("Invalid chemistry. Choose 'NMC' or 'LFP'.")

# === Configuration Based on Chemistry ===
base_dir = chemistry #change path if data is in different directory than code
OCV_file = os.path.join(base_dir, "OCV_data.txt")
battery_capacity = 2.6 if chemistry == "NMC" else 200  # Ah
flip_discharge_SOC = chemistry == "LFP"

# === Load (X, Y) data ===
def load_XY_data(filepath, skip=1):
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File not found: {filepath}")

    X, Y = [], []
    with open(filepath, 'r') as file:
        for _ in range(skip):
            next(file)
        for line in file:
            try:
                x, y = map(float, re.split(r'\s+', line.strip()))
                X.append(x)
                Y.append(y)
            except ValueError:
                print(f"Skipping invalid line: {line.strip()}")
    return np.array(X), np.array(Y)

# === OCV interpolation ===
def OCV(SOC, X, Y):
    return np.interp(SOC, X, Y)

# === PNGV voltage model ===
def PNGV_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, Cb, C_rate):
    U_ocv = OCV(SOC_vals, OCV_X, OCV_Y)
    Us = Rs * I
    Ub = 0
    U_bat = []
```

```
    for i in range(len(SOC_vals)):
        if i == 0:
            u_model = U_ocv[i] - R0 * I - Us - Ub
            U_bat.append(u_model)
            continue

        delta_SOC = abs(SOC_vals[i] - SOC_vals[i - 1])
        dt = max(delta_SOC / abs(C_rate), 1e-6)

        dUs_dt = (-Us + Rs * I) / (Rs * Cs)
        Us += dt * dUs_dt

        Ub += dt * I / Cb

        u_model = U_ocv[i] - R0 * I - Us - Ub
        U_bat.append(u_model)

    return np.array(U_bat)

# === Error function for optimizer ===
def error_function(params, charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity,
method=None):
    R0, Rs, Cs, Cb = params

    if R0 < 0 or Rs < 0 or Cs <= 0:
        return np.inf

    if method == "Powell":
        if not (1 <= Cb <= 10000):
            return np.inf
    else:
        if Cb <= 0:
            return np.inf

    total_error = 0.0
    for C_rate, SOC_vals, U_actual in charge_data:
        I = -C_rate * battery_capacity
        U_model = PNGV_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, Cb, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)

    for C_rate, SOC_vals, U_actual in discharge_data:
        I = C_rate * battery_capacity
        U_model = PNGV_voltage_model(SOC_vals, OCV_X, OCV_Y, I, R0, Rs, Cs, Cb, C_rate)
        total_error += np.sum((U_model - U_actual) ** 2)

    return total_error

# === Load Data ===
OCV_X, OCV_Y = load_XY_data(OCV_file)
```

```
charge_files = ["Charge_0.2C.txt", "Charge_0.5C.txt"]
discharge_files = ["Discharge_0.2C.txt", "Discharge_0.5C.txt"]

charge_data = []
discharge_data = []

for filename in charge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        charge_data.append((C_rate, SOC, U))

for filename in discharge_files:
    filepath = os.path.join(base_dir, filename)
    SOC, U = load_XY_data(filepath)
    if flip_discharge_SOC:
        SOC = 1 - SOC
    if len(SOC) > 0:
        C_rate = float(filename.split('_')[1].replace('C.txt', '').replace('C', ''))
        discharge_data.append((C_rate, SOC, U))

# === Optimization ===

initial_guesses = [
    [0.010, 0.002, 200, 5000],
    [0.015, 0.010, 500, 3000],
    [0.020, 0.020, 1000, 2000],
]

bounds_list = [
    [(0.001, 0.05), (0.0001, 0.1), (1, 2000), (10, 10000)],
]

methods = ["L-BFGS-B", "SLSQP", "Powell"]
all_results = []

for method in methods:
    print(f"\n=== Trying Method: {method} ===")
    best = None
    best_err = float('inf')
    for x0 in initial_guesses:
        for b in bounds_list:
            args = (charge_data, discharge_data, OCV_X, OCV_Y, battery_capacity, method)
            result = opt.minimize(
                error_function,
                x0=x0,
                args=args,
                bounds=None if method == "Powell" else b,
```

```
                method=method
            )
            if result.success and result.fun < best_err:
                best = result
                best_err = result.fun
    if best:
        all_results.append((method, best_err, best.x))
        R0, Rs, Cs, Cb = best.x
        print(f"Best for {method}: Error={best_err:.6f}, Params: R0={R0:.5f}, Rs={Rs:.5f},
Cs={Cs:.2f}, Cb={Cb:.2f}")
    else:
        print(f"{method} optimization failed.")

def all_methods_agree(results, tol=1e-6):
    ref_err, ref_params = results[0][1], results[0][2]
    for _, err, params in results[1:]:
        if not np.isclose(err, ref_err, atol=tol):
            return False
        if not np.allclose(params, ref_params, atol=tol):
            return False
    return True

if all_results:
    all_results.sort(key=lambda x: x[1])
    best_method, best_error, best_params = all_results[0]

    if all_methods_agree(all_results):
        print("\n✅ All methods agree on the result.")
    else:
        print(f"\n✅ Best method: {best_method} with error: {best_error:.6f}")

    R0_opt, Rs_opt, Cs_opt, Cb_opt = best_params
    print(f"Optimized parameters:\n"
          f"R0 = {R0_opt:.5f} Ω\n"
          f"Rs = {Rs_opt:.5f} Ω\n"
          f"Cs = {Cs_opt:.2f} F\n"
          f"Cb = {Cb_opt:.2f} F")
else:
    raise RuntimeError("❌ Optimization failed for all methods!")

# === PNGV Model wrapper for multi-method plotting ===
def PNGV_voltage_plot_multi(SOC, I, params, C_rate):
    R0, Rs, Cs, Cb = params
    return PNGV_voltage_model(SOC, OCV_X, OCV_Y, I, R0, Rs, Cs, Cb, C_rate)

# === Error calculation helpers ===
def calculate_percentage_error(U_actual, U_model):
    return np.abs(U_actual - U_model) / np.maximum(np.abs(U_actual), 1e-3) * 100
```

```
def calculate_absolute_error(U_actual, U_model):
    return np.abs(U_actual - U_model)

# === Multi-method plotting for PNGV model ===
def plot_comparison_multiple_methods_PNGV(data, mode_label, all_results, flip_xaxis=False):
    plt.figure(figsize=(10, 6))

    color_map = {0.2: 'green', 0.5: 'red'}
    method_styles = {
        'L-BFGS-B': '--',   # dashed
        'SLSQP': '-.',      # dash-dot
        'Powell': ':'       # dotted
    }

    # Collect unique C-rates sorted
    C_rates = sorted(set(round(c_rate, 3) for c_rate, _, _ in data))

    # Plot all experimental data first
    exp_handles = []
    for C_rate in C_rates:
        for c_rate, SOC_vals, U_actual in data:
            if round(c_rate, 3) == C_rate:
                SOC_percent = SOC_vals * 100
                color = color_map.get(round(C_rate, 1), 'black')
                handle, = plt.plot(SOC_percent, U_actual, '-', color=color,
label=f"Experiment {C_rate:.1f}C")
                exp_handles.append(handle)
                break  # Plot once per C-rate

    # Now plot all methods per C-rate with custom color and style
    method_handles = []
    for method, err, params in all_results:
        print(f"\n--- {mode_label} | Method: {method} | Total Error: {err:.5f} ---")
        for C_rate in C_rates:
            for c_rate, SOC_vals, U_actual in data:
                if round(c_rate, 3) == C_rate:
                    I = -C_rate * battery_capacity if
mode_label.lower().startswith("charg") else C_rate * battery_capacity
                    U_model = PNGV_voltage_plot_multi(SOC_vals, I, params, C_rate)
                    SOC_percent = SOC_vals * 100

                    color = color_map.get(round(C_rate, 1), 'black')
                    linestyle = method_styles.get(method, '--')

                    handle, = plt.plot(SOC_percent, U_model, linestyle=linestyle,
color=color,
                                       label=f"{method} Method {C_rate:.1f}C")
                    method_handles.append(handle)
```

```
                    # Calculate and print errors
                    abs_error = calculate_absolute_error(U_actual, U_model)
                    perc_error = calculate_percentage_error(U_actual, U_model)
                    print(f"  C-rate {C_rate:.1f}C -> Avg Abs Error:
{np.mean(abs_error):.4f} V, Avg % Error: {np.mean(perc_error):.2f}%")
                    break

    plt.xlabel("State of Charge (%)")
    plt.ylabel("Voltage (V)")
    plt.title("Experiment vs PNGV Model")
    plt.grid(True)
    if flip_xaxis:
        plt.gca().invert_xaxis()
    plt.xticks(np.arange(0, 110, 20))
    plt.tight_layout()

    # Combine handles and labels: experiments first, then methods
    all_handles = exp_handles + method_handles
    labels = [h.get_label() for h in all_handles]
    plt.legend(all_handles, labels)
    plt.show()


plot_comparison_multiple_methods_PNGV(charge_data, "Charge", all_results)
plot_comparison_multiple_methods_PNGV(discharge_data, "Discharge", all_results,
flip_xaxis=True)

# === Model Output Consistency Check ===
print("\n🔍 Checking PNGV model output differences between methods:")
C_rate_test = 0.5
SOC_test, U_test = discharge_data[1][1], discharge_data[1][2]
model_outputs = []

for method, err, params in all_results:
    I = C_rate_test * battery_capacity
    U_model = PNGV_voltage_plot_multi(SOC_test, I, params, C_rate_test)
    model_outputs.append((method, U_model))

for i in range(len(model_outputs)):
    for j in range(i + 1, len(model_outputs)):
        method_i, Ui = model_outputs[i]
        method_j, Uj = model_outputs[j]
        max_diff = np.max(np.abs(Ui - Uj))
        print(f"Max voltage difference between {method_i} and {method_j}: {max_diff:.10f}
V")
```