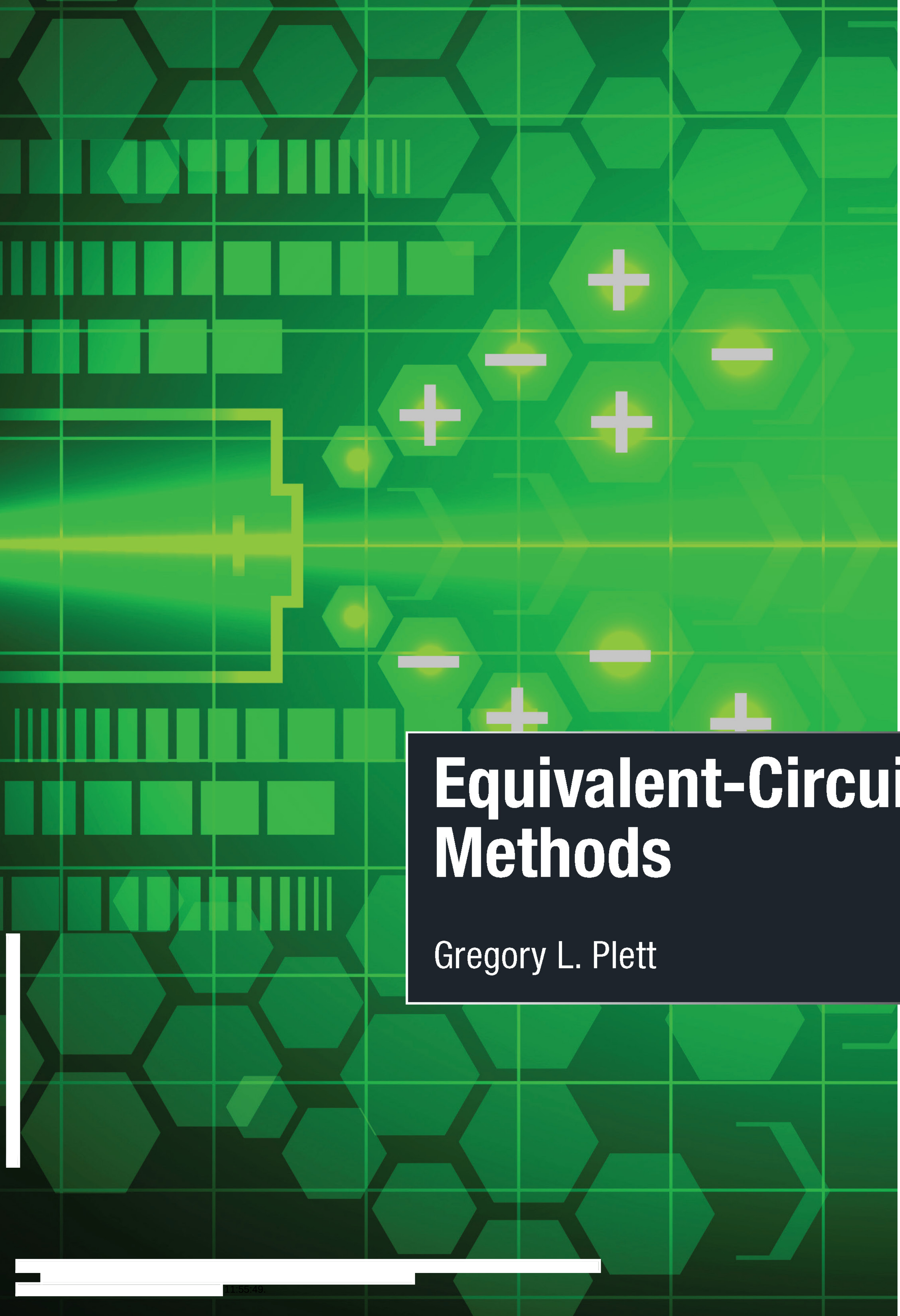


BATTERY MANAGEMENT SYSTEMS

Equivalent-Circuit Methods

Gregory L. Plett

VOLUME II

Battery Management Systems

Volume II

Equivalent-Circuit Methods

Battery Management Systems

Volume II

Equivalent-Circuit Methods

Gregory L. Plett

ARTECH
HOUSE

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalog record for this book is available from the British Library.

ISBN-13: 978-1-63081-027-6

Cover design by John Gomes

Contents

11:56:03.

11:56:03.

Preface

This book comprises the second volume in what is planned to be a three-volume series describing battery-management systems. The first volume focused on deriving mathematical sets of equations or *models* that describe how battery cells work, inside and out. This second volume applies equivalent-circuit style models to solve problems in battery management and control. The third volume will show how physics-based models also can be used to solve problems in battery management and control, leading to better results. The intent of the series is not to be encyclopedic; rather, it is to put forward only the current best practices, with sufficient fundamental background to understand them thoroughly.

This particular volume is organized in the following way:

- Chapter 1 introduces the requirements of a battery-management system. These include sensing, control, protection, state and health estimation, and communications.
- Chapter 2 reviews equivalent-circuit models of lithium-ion cells and shows how to use them to simulate the response of a battery pack to an input stimulus.
- Chapter 3 investigates battery-cell state estimation. Nonlinear Kalman filters are shown to give very good estimates, along with dynamic error bounds that enable confident use of the estimates when computing battery-pack energy and power.
- Chapter 4 looks at state-of-health estimation. A simple method can be used to estimate cell resistance, but we find that it is more difficult to estimate cell total capacity. A regression technique based on total-least squares provides optimal unbiased results.
- Chapter 5 discusses cell balancing. It considers factors that lead to imbalance, some questions that must be addressed when designing a balancer, and some circuit options for balancing cells.
- Chapter 6 explores computation of power limits where terminal-voltage constraints are applied. The simple method from Chap. 1 is extended to work with a full equivalent-circuit cell model.

- Finally, Chap. 7 exposes the fundamental flaw with voltage-based power-limit estimates and introduces physics-based methods that can be used alongside a circuit model to give better limits.

The intended audience for this material is someone with an undergraduate degree in engineering—principally electrical or mechanical. The reader does not need to be intimately familiar with all the concepts from Volume I of this series to be able to benefit from the topics in this volume. However, the deeper understanding that can be developed by studying Volume I will add richness to the study, and will aid understanding some concepts explored in Chaps. 2, 4, and 7 of this book that would otherwise be quite opaque.

The content in this book has been taught multiple times to students of diverse backgrounds in *ECE5720: Battery Management Systems* at the University of Colorado Colorado Springs. Lecture notes and lecture videos are available at `http://mocha-java.uccs.edu/ECE5720/index.html`. As the lecture videos sometimes explain the concepts of this book in a somewhat different way, the additional perspective may be an advantage to the learner.

I am greatly indebted to a number of my colleagues and students who have supported and assisted me over the years in understanding and developing the theory and methods presented in this work. First, I would like to acknowledge Dr. Daniel Rivers, founding CEO of Compact Power, Inc. (now, LGCPI), who introduced me to this field in the first place. Without his support and encouragement, I never would have studied battery management, and this book would not exist. I would also like to thank Dr. Saeed Siavoshani for inviting me to participate as an instructor in an SAE Hybrid-Electric-Vehicle Academy some years ago. Much of the material of this book was first developed to present at that academy; it was then expanded to become the *Battery Management Systems* course just mentioned, and now it has matured to book form. Among my students, I owe a special debt of gratitude to Mr. Lukas Aldrich and Mr. Kirk Stetzel for critiquing the content and helping with many of the examples, and to Mr. Alfred Randall and Mr. Roger Perkins for their work on reduced-order models of cell degradation that are presented in Chap. 7. My colleague and friend Dr. M. Scott Trimboli has also been a great encourager of this work, as he was with Volume I.

Despite my best intentions, there are certain to be errors and confusing statements in this book. Please feel free to send me corrections and suggestions for improvements.

1 Battery-Management-System Requirements

This book investigates the proper management of battery packs, a task that requires both hardware (electronics) and software (computer program) components. The hardware elements incorporate electronic circuits to ensure the safety of the battery pack and its operator and to make measurements that include battery-cell voltages, electrical current, and temperature. The software portions monitor and coordinate the activities of the battery pack.

While we look at both hardware and software aspects in this book, we devote most of our attention to software methods or *algorithms*. These implement mathematical calculations that use measured data to estimate and summarize battery-pack present operational status and to predict its near-future performance limits. And, although most of the approaches that we will discuss can be applied to battery packs comprising cells of any chemistry, we will focus on applications involving lithium-ion battery cells.

A survey of the relevant literature uncovers many different methods for different aspects of battery management. We will explore some simple methods to introduce many of the key concepts but will devote most of our study to some more complex but also more accurate and robust approaches, which we prefer. However, we recognize that an implementation of a more complex algorithm requires more processing power—and hence a greater cost—than a simpler counterpart, so these algorithms are best suited to applications involving mission-critical systems or large battery packs comprising many battery cells where a substantial investment must be protected.

The methods and algorithms we discuss would typically be implemented by a *battery-management system* or BMS.[1] A BMS is an *embedded system*; that is, purpose-built electronics plus processing to enable a specific application. For example, Fig. 1.1 shows the electronics por-

[1] IEEE Standard 1491 defines a *battery monitoring system* as "A permanently installed system for measuring, storing, and reporting battery operating parameters."

tion of a prototype BMS being developed for research purposes at the University of Colorado Colorado Springs.

Figure 1.1: An example BMS.

THE PRIMARY PURPOSES of a battery-management system are:

- First and foremost, to protect the safety of the operator of the battery-powered system. The BMS must detect unsafe operating conditions and respond. This may demand disconnecting and isolating the battery pack from the load, warning the operator by some display or alert, and so forth.
- Second, to protect cells of the battery pack from damage in abuse or failure cases. This may involve active intervention under software control, or specialized electronics that can detect failures and isolate the failing components from the rest of the battery pack and from the load it powers.
- Third, to prolong the life of the battery under normal operating cases. The BMS does so by coordinating with the controller of the load it powers, advising it of dynamic limits on power that can be sourced or sunk over some short future interval that ensures that the battery pack will not be overcharged or overdischarged. It also controls the thermal-management system, ensuring that the battery pack is kept within its design operational-temperature range.
- Fourth, to maintain the battery pack in a state in which it can fulfill its functional design requirements. Thus, for example, it will not allow a battery pack to become so far discharged that it cannot deliver its rated discharge power, nor will it allow the pack to become so highly charged that it cannot receive its rated charge power at any point in time.

THERE IS A COST associated with advanced methods of battery management, so not all applications implement all features. This cost adds to the purchase price of the battery pack, so the battery-management algorithms must provide tangible value. A good rule of thumb says "your battery is 'cheap enough' if you can't remember the last time you replaced it." The idea is that replacing the inexpensive battery in something like a television remote control is not financially painful, so spending more money on an advanced remote control that makes more efficient use of the battery is probably not worth it. However, mission-critical and large battery packs represent a greater investment and motivate better battery management. If you are required to replace an expensive battery prematurely, or if you are unable to complete your mission due to battery failure because of poor management, you will remember it for a *very* long time! Another way of looking at this is to consider the cost of the battery pack in relation to the cost of a battery-pack failure. The cost of battery failure for a television remote control is not large; however, the cost of battery-pack failure for a mission-critical or large battery installation can be very high. Consequently, as this book focuses on advanced methods for management and control of battery packs, it is most relevant for applications where the cost of battery-pack failure is high and where the added cost can be justified, although the methods that we discuss are quite general.

An important application category considers vehicles having electric-drivetrain components. These vehicular applications include the following generic subcategories:

Hybrid-electric vehicles (HEVs). These vehicles have motive power provided by an electric motor plus at least one other source (e.g., a gasoline engine). A battery pack stores a small amount of energy, and the battery–motor combination is used primarily for power boost when the vehicle must accelerate, or as a power sink when the vehicle must decelerate. This enables the gasoline engine to operate at a more constant operational point comprising a combination of revolutions per minute (RPM) and torque, which can be more efficient and can also allow the vehicle to achieve the same overall peak performance requirements with a smaller engine. HEVs have essentially zero all-electric vehicle range and are never plugged in to recharge their battery pack; instead, the gasoline engine recharges the battery when extra power is available. An example HEV is the Toyota Prius, shown in the top frame of Fig. 1.2.

Plug-in hybrid-electric vehicles (PHEVs). These vehicles are similar to HEVs but have a somewhat larger battery pack and motor. They can operate in electric-only mode under some operating conditions, typically at lower speeds such as for residential or city driving. Consequently, they have some all-electric range, often on the order of 10 to 20 miles. The vehicle can be "plugged in" to the utilities grid to recharge the battery pack. Subsequently, the vehicle operates first in a *charge-depletion* mode where the majority of the traction power is taken from the battery and not from the gasoline engine. When the battery charge is depleted to some minimum allowed level, the vehicle then switches to a *charge-sustaining* mode where it operates just like a standard HEV. An example PHEV is the Ford C-MAX Energi, which is shown in the second frame in Fig. 1.2.

Extended-range electric vehicles (E-REV). E-REVs are similar to PHEVs but have somewhat larger battery pack and motor. They can operate in electric-only mode under nearly all operating conditions so long as battery power is available. Their all-electric range is also greater, often on the order of 35 or more miles. As with a PHEV,

Figure 1.2: Vehicles having electrified drivetrain. From top to bottom: Toyota Prius, Ford C-MAX Energi, Chevy Volt, and Tesla Model S.
(All photos carry Creative-Commons "Attribution 2.0 Generic (CC BY 2.0)" license, and were taken by Soknet Soknet, Kyle Harris, Kārlis Dambrāns, and Niels de Wit, respectively. Cropped from originals downloaded from Flickr.com.)

they may be plugged in to charge their battery pack and they operate in charge-depletion and charge-sustaining modes. Since many commuters do not drive more than 35 miles in a day, they will rarely exceed the all-electric range of the vehicle and so the car will operate essentially as an electric vehicle for them. But if they ever do need to drive a distance farther than the battery energy alone allows, "extended range" is provided by the gasoline engine. An example E-REV is the Chevy Volt, which is shown in the third frame in Fig. 1.2.

Electric vehicles (EVs), also known as *battery-electric vehicles (BEVs)*. For these vehicles, the battery–motor combination provides the only source of motive power. There is no gasoline engine. Consequently, the vehicle design is much simpler than for any of the flavors of hybrid, but the vehicle range is limited by the amount of useable energy that can be stored by the battery. The size of the battery pack then becomes a critical design variable that is factored into the economic optimizations performed when designing the vehicle. Some commercial EVs have range of less than 100 miles, but others have range more than 300 miles between recharges. An example EV is the Tesla Model S, which is shown in the bottom frame in Fig. 1.2.

All of these vehicle types employ battery packs that are "large," "high voltage," and "high current." There are some distinctions in design, which we will detail when necessary. However, the commonalities of their battery systems are more significant than their differences; so when distinctions aren't important, we refer to any member of the class of vehicles having electric-drivetrain components as *xEV*. As xEVs represent a very important category of application requiring large battery packs, many of the examples presented in this book will be described in terms of an xEV implementation.

However, we do note that there are other significant and growing application domains requiring large battery packs. These include some that support and supplement the standard utilities electrical grid. For example, a large battery bank may be used for *grid-storage* of energy when opportunistic generation exceeds demand, such as from a solar or wind farm. The stored energy can be supplied back to the grid at a later time when the primary generation power is unavailable. Or, a battery pack may be used for *grid-backup* to provide energy to a load during grid power outages. For example, large, mobile grid-backup systems can be installed in the trailer of a semitruck and driven to neighborhoods to restore power temporarily during an emergency or during scheduled grid maintenance. Also, large battery packs are becoming more common in *frequency regulation* applications, where the battery pack operates as small-scale grid storage to make use of short-term surplus generation to charge the pack and in turn provide short-term power to the grid when insufficient generation is available.

These applications—and others—are very different externally, and do place some different requirements on the battery-management-system design. However, they all tend to share the same kinds of algorithmic requirements, which is the primary focus of this book. Therefore, most of what is presented herein will be of interest to any battery-pack controls designer.

1.1 Battery-pack topology

Electrical *power* is computed as electric current multiplied by voltage: that is, $p = iv$. Therefore, high-power battery packs must deliver either high current, high voltage, or both. To achieve a design requirement on maximum power, the battery-pack engineer must make decisions regarding the topology of the pack. What should be its voltage range and peak current?

The chemistry of an individual cell fixes its voltage range; so for high-voltage packs we must connect multiple cells in series. The pack voltage is then the sum of the individual cell voltages. Roughly speaking (i.e., if all cell voltages are assumed equal), we have that $v_{\text{pack}} = N_s \times v_{\text{cell}}$, where the battery-pack topology wires N_s cells in series.

Cell construction places limits on the maximum current that the cell is designed to sustain; so for high-current packs we must wire cells in parallel. The pack current is then the sum of the currents going through the individual cells that are wired in parallel. Roughly speaking (i.e., if all of these currents are assumed equal), we have that $i_{\text{pack}} = N_p \times i_{\text{cell}}$, where the battery-pack topology wires N_p cells in parallel.

The tradeoff regarding how many cells to wire in parallel versus in series in order to achieve a battery-pack maximum-power design requirement is generally determined by economic and safety factors. Groups of cells termed *modules* are usually designed to have maximum voltage less than 50 V for safety,[2] and battery-pack total voltage is generally kept to less than 600 V because power electronics that operate at higher voltages are presently very expensive. Within these voltage ranges, when designing to meet a power-demand requirement, higher voltages tend to be preferred over higher currents to minimize resistive power losses in both the wiring that is internal to the battery pack and to the load (computed as $i_{\text{pack}}^2 \times R$). This also allows using wire having smaller diameter (and lower cost) as power loss scales with the square of current versus only linearly with resistance.

[2] There is an evolving discussion regarding what dc voltage range is considered to be safe and what is considered to be hazardous. The 2012 edition of the *NFPA 70E: Standard for Electrical Safety in the Workplace* listed a dc shock hazard limit of 50 V dc, but this was raised to 100 V dc in the 2015 edition. Many designers still adhere to the 50 V limit to minimize risk.

11:56:42.

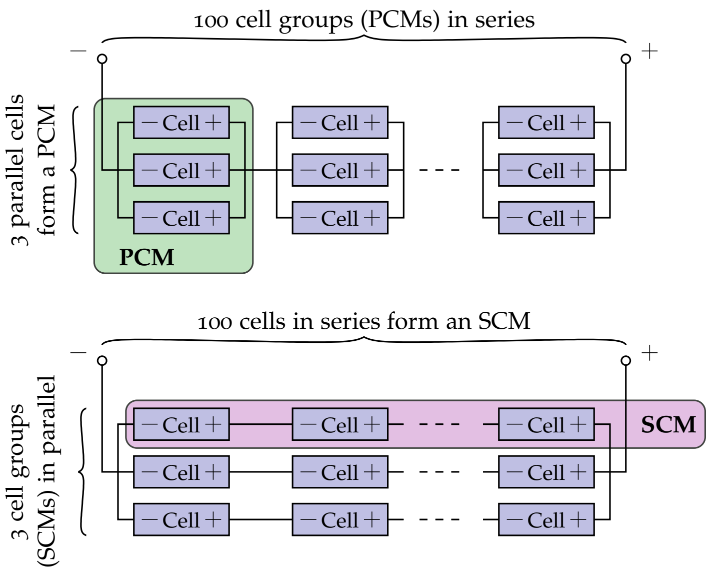

100 cell groups (PCMs) in series
−
+
3 parallel cells form a PCM
− Cell +
PCM
100 cells in series form an SCM
SCM
3 cell groups (SCMs) in parallel

Figure 1.3: Two different approaches to modularizing 300 cells in a high-power battery pack: PCM versus SCM.

Battery-pack design is often *modular*. That is, a design is first created and optimized for a module comprising a small group of cells; then, modules are connected in series and/or parallel to achieve the overall battery-pack design objectives. This allows reusing a fixed-module design for many different possible application scenarios and minimizes *nonrecurring engineering (NRE)* costs.

Modules may be constructed with any number of cells wired in series and parallel, and we will explore some of the tradeoffs in Chap. 2. Here, we note only that two extreme cases are shown in Fig. 1.3. The *parallel-cell-module (PCM)* approach wires cells in parallel to make modules, then wires modules in series to create a battery pack. The battery pack drawn in the top of the figure configures 300 cells by wiring three cells in parallel to make a PCM, and then wiring 100 PCMs in series to make the pack. The *series-cell-module (SCM)* approach instead wires cells in series to make modules, then wires modules in parallel to make a battery pack. The battery pack drawn in the bottom of the figure configures the same 300 cells by wiring 100 cells in series to make an SCM, and then wiring three SCMs in parallel to make the pack.[3]

[3] If all cells are identical, then these two battery packs provide the same energy, power, and so forth. However, if there are differences between the cells, the packs can operate in quite different ways. We explore this more in Chap. 2, where we see how to simulate battery packs having different topologies.

We can design battery packs and battery-management system for either approach. Most often, however, we use something in between these extremes. For example, a 3P6S module has eighteen cells configured with three cells wired in parallel and six cells in series. The module power and energy are then both approximately eighteen times that of a single cell (but not exactly, as we shall discover later).

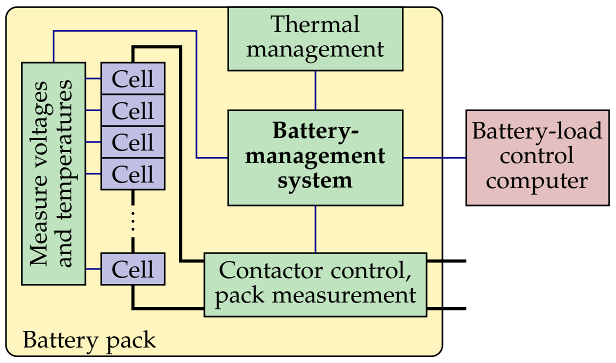

Thermal management
Measure voltages and temperatures
Cell
Cell
Cell
Cell
Cell
Battery-management system
Battery-load control computer
Contactor control, pack measurement
Battery pack

Figure 1.4: The context of the BMS inside the battery pack.

1.2 BMS design requirements

The BMS is interconnected with all major components in the battery pack, as illustrated in Fig. 1.4. These include the *battery stack*, comprising all of the cells, the sensing and control electronics, and at least some part of the thermal-management system.

The electronics may be implemented as a single monolithic design, or there may be separate elements having different functionality distributed throughout the battery pack. As an example, the drawing shows cell-voltage and temperature-sensing elements as distinct from pack sensing and control and from overall supervisory battery management, all within the confines of an overall battery pack, which is drawn as the yellow shaded box. In an xEV, the battery-load control computer is the overall vehicle controller; in other applications, the corresponding controller would be interfaced.

Regardless of electronics topology or battery application, BMS functional requirements can be broken down into five overall categories:

1. *Sensing and high-voltage control*: The BMS must measure cell voltages, module temperatures, and battery-pack current. It must also detect isolation faults and control the contactors and the thermal-management system.
2. *Protection*: It must include electronics and logic to protect the operator of the battery-powered system and the battery pack itself against overcharge, overdischarge, overcurrent, cell short circuits, and extreme temperatures.
3. *Interface*: The BMS must communicate regularly with the application that the battery pack powers, reporting available energy and power and other indicators of battery-pack status. Further, it must record unusual error or abuse events in permanent memory for technician diagnostics via occasional on-demand download.
4. *Performance management*: It must be able to estimate state of charge

(SOC) for all the cells of the battery pack, compute battery-pack available energy and power limits, and balance cells in the battery pack.

5. *Diagnostics*: Finally, it must be able to estimate state-of-health (SOH), including detecting abuse, and may be required to estimate state-of-life (SOL) of the battery cells and pack.

In this chapter, we present a high-level overview of these five requirements. Later chapters will develop performance-management and diagnostic topics in detail. In particular, Chaps. 3 through 7 focus on SOC estimation, SOH estimation, balancing, and power-limits estimation in more depth.

As we proceed through the remainder of this chapter, the section titles will refer to requirement numbers from the above list. For example, all section titles beginning with "Requirement 1" have to do with sensing and high-voltage control, and so forth.

1.3 Requirement 1a. Battery-pack sensing: Voltage

In a battery pack comprising lithium-ion cells, *all* individual cell voltages must be measured.[4] Cell terminal voltage can provide a measure of the relative balance of cells in the battery pack, and is a critical input to most SOC and SOH estimation algorithms.

[4] If cells are wired in parallel, they will share the same terminal voltage, and so voltages of parallel cells cannot be distinguished. We could more correctly say something like "all voltages of individual parallel groups of cells must be measured."

Out-of-bounds cell voltage is also an indicator of a number of serious lifetime and safety issues. For example, overcharging a lithium-ion cell can initiate unwanted internal chemical side reactions that degrade the cell. Overdischarging can precipitate a sequence of events leading to a cell short-circuiting. In extreme cases, either of these can cause *thermal runaway*, where heat generated by side reaction or short circuit accelerates the failure mechanisms via natural positive feedback, ultimately resulting in a battery fire and/or explosion. Due to various reasons discussed in Chap. 5, we cannot assume that all cells automatically have the same voltage, and so we cannot skip measuring any voltage in a lithium-ion battery pack.

A number of silicon vendors have come to the aid of the BMS electronics designer by designing integrated-circuit (IC) "chipsets" that can assist with cell-voltage measurement.[5] These are low-cost devices with little or no general-purpose processing capabilities on-chip. They implement the difficult task of measuring analog voltages with high accuracy, high common-mode rejection, and fast response in high-electromagnetic-interference (EMI), high-temperature, and high-vibration environments, such as encountered in a vehicular application. A single IC can be used to measure the voltages of a

[5] See, for example, Andrea, D., *Battery Management Systems for Large Lithium Ion Battery Packs,* Artech House, 2010, which has quite a bit more detail on this topic, and on BMS electronics design in general.

number of cells connected in series, and two ICs can often be placed in parallel for redundant fault-tolerant designs.

One example is the popular LTC6803 part, designed in Colorado Springs by Linear Technology Corporation. A functional block diagram of this IC is shown in Fig. 1.5. A single LTC6803 can monitor the voltages of up to 12 cells wired in series in a module. An internal analog multiplexer connects cells to a 12-bit delta-sigma analog-to-digital converter, and a single "read" command can trigger the measurements of all 12 voltages in rapid sequence. Up to 10 ICs can be daisy-chained together to monitor up to 120 cells in a battery pack and electrically isolated communications are built into the design. It supports either internal (slow) or external (faster) dissipative cell-balancing circuitry and up to four temperature measurements per module (in addition to the internal IC die temperature). It can be powered by the module of cells itself or from an external source.

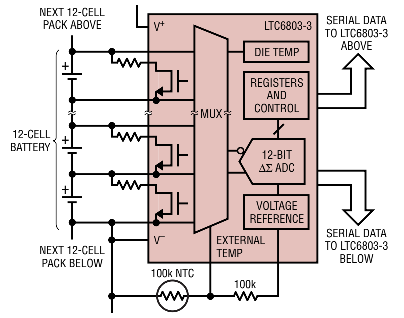


Figure 1.5: An example IC for battery-stack voltage measurements.
(Figure used with permission of Linear Technology Corporation.)

When selecting a voltage-monitoring IC for a BMS electronics design, some of its specifications to consider and compare to design requirements for the battery pack include how many cells can each IC monitor? How many cells in total can be monitored? Does the IC support either passive or active balancing? What is the measurement resolution and accuracy? How many temperature measurements will it support? How many wires are required to communicate between ICs (if multiple ICs are required in a design)? And, what is the chipset availability and per-cell cost?

1.4 *Requirement 1b. Battery-pack sensing: Temperature*

As we have already explored in detail in Volume I of this series, lithium-ion cell dynamics are strong functions of temperature. For example, resistance increases and chemical processes tend to slow down at cold temperatures. So, we must know cell temperature to be able to predict near-future cell performance. Further, degradation mechanisms are also temperature-dependent: generally, high temperatures accelerate degradation rates, although charging a battery pack at low temperatures can also lead to premature failure via lithium-plating. Also, a BMS must be able to sense temperature to control the thermal-management system to maintain battery-pack temperatures in a safe region.

Ideally, we would measure each cell's *internal* temperature. However, cells are not (mass-)produced with temperature sensors built in, so we must rely on measurements of cell *external* temperatures. In addition, as there is an expense per sensor, we would like to minimize the total number of sensors required. With an accurate pack thermal model, we can place a limited number of temperature sensors external to one or more cells per module and then estimate internal temperatures of all cells. This is usually sufficient.

But, how do we measure temperature? To measure any simple physical quantity electronically, we must first represent that quantity as a voltage signal and then measure the voltage via an analog-to-digital converter circuit. There are two primary approaches to doing so to measure temperature.

The first is to use a *thermocouple*, which is a device comprising two dissimilar metals in contact with each other that acts as a miniature battery. A very small voltage is produced when the temperature of the thermocouple is different from a reference temperature at another part of the measurement circuit, and this voltage depends on magnitude of the temperature difference. The thermocouple voltage can be amplified and measured and temperature can be computed from this measurement. A design challenge when using thermocouples is that the reference temperature must be independently known or measured, which probably makes thermocouples best suited for laboratory testing and not for production BMS designs.

The second method uses a *thermistor*, and is probably better-suited for use in commercial products. All resistors have value that varies somewhat with temperature but are usually designed to minimize this variation. A thermistor, on the other hand, is designed to maximize and to capitalize on temperature variation. Negative-temperature-coefficient (NTC) thermistors have resistance that varies inversely with temperature, and positive-temperature-coefficient (PTC) thermistors have resistance that varies directly with temperature. If we can measure thermistor resistance, we can then infer temperature.

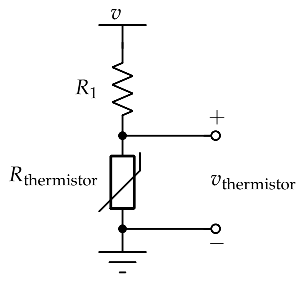


Figure 1.6: Voltage-divider circuit.

To measure resistance, we can use a voltage-divider circuit, such as shown in Fig. 1.6. In the circuit, the top resistor R_1 has resistance that does not vary appreciably with temperature, but the lower resistor $R_{\text{thermistor}}$ has value that is designed to vary significantly with temperature. We compute overall current as

$$i = \frac{v}{R_1 + R_{\text{thermistor}}}.$$

Then, we note that the measured voltage is $v_{\text{thermistor}} = iR_{\text{thermistor}}$ or

$$v_{\text{thermistor}} = \frac{R_{\text{thermistor}}}{R_1 + R_{\text{thermistor}}} v.$$

The value of R_1 is designed to limit power loss through the measurement circuit but at the same time provide a useful measurement range for $v_{\text{thermistor}}$.

If we measure $v_{\text{thermistor}}$ and know the circuit-design parameters, we can compute

$$R_{\text{thermistor}} = \frac{v_{\text{thermistor}}}{v - v_{\text{thermisor}}} R_1. \tag{1.1}$$

The thermistor data sheet will give an expression relating $R_{\text{thermistor}}$ to temperature. For example, a representative relationship computes

$$R_{\text{thermistor}} = R_0 \exp\left(\beta\left(\frac{1}{273.15 + T} - \frac{1}{273.15 + T_0}\right)\right), \tag{1.2}$$

where the temperature being measured is denoted as T and nominal resistance at reference temperature T_0 is denoted as R_0; temperatures are converted from celsius to kelvin by adding 273.15, and β is a device parameter. The top frame of Fig. 1.7 plots thermistor resistance for an NTC device having $R_0 = 100\,\text{k}\Omega$ at $T_0 = 25\,^\circ\text{C}$ and $\beta = 4282$.

If this device were placed in the lower leg of a voltage divider having a 5 V source and $R_1 = 100\,\text{k}\Omega$, we would measure the thermistor voltage $v_{\text{thermistor}}$ as a function of temperature that is plotted in the middle frame of Fig. 1.7. We can then compute thermistor resistance via Eq. (1.1) and temperature via solving Eq. (1.2) for that value of resistance. For efficient computation in an embedded BMS, the overall relationship between measured voltage and temperature can be precomputed and stored in a *lookup table (LUT)*. The result for this example is plotted in the bottom frame of Fig. 1.7.

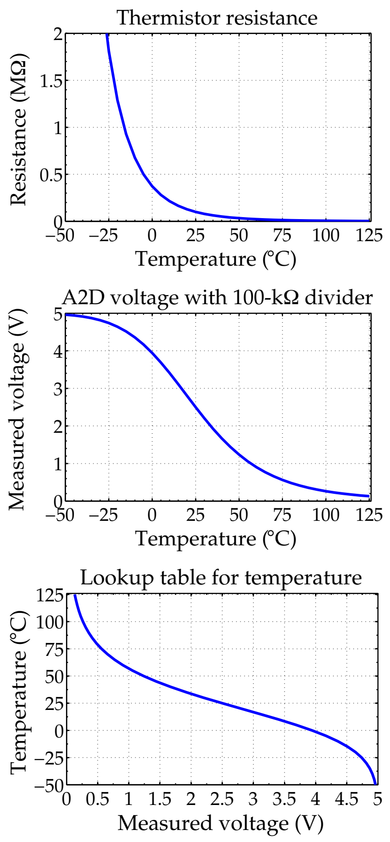


Figure 1.7: Thermistor usage example.

1.5 *Requirement 1c. Battery-pack sensing: Current*

A BMS must measure battery-pack current. In part, this is to detect and log abuse conditions and to ensure safety. However, battery current is also a critical input to most SOC and SOH estimation algorithms.

There are two basic electronics elements that can be used in a circuit to sense current: current-shunts and Hall-effect sensors.

A *current-shunt* is a low-value (e.g., $0.1\,\text{m}\Omega$) high-precision resistor placed in series with the battery pack, usually at the negative terminal. The voltage drop v_{shunt} across the shunt resistance R_{shunt} is measured using a standard analog-to-digital converter, and current is computed as $i = v_{\text{shunt}}/R_{\text{shunt}}$. Since the shunt resistance must be small (to avoid large power losses due to $i^2 R_{\text{shunt}}$ heating), the voltage drop across the shunt will be small as well. So, the voltage is usually amplified before sensing and the calculation for current is adjusted accordingly.

Fig. 1.8 shows a photograph of a current shunt (top frame) and a block diagram for current sensing using a current shunt (bottom frame). Examining the current shunt itself in more detail, note that

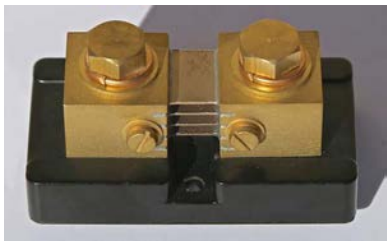
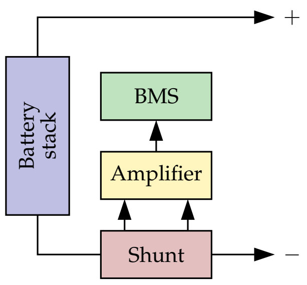



Figure 1.8: Sensing current with a current shunt.

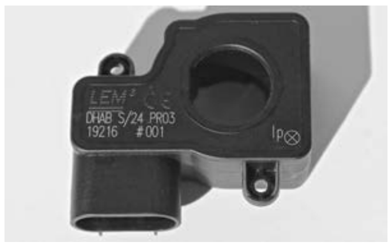
there are four connection terminals. The large terminals on the top of the device are for connecting to the main battery-pack current-carrying wires: one side is connected to the negative terminal of the battery stack, and the other side is connected to the output negative terminal of the battery pack. The entire pack current then passes through the parallel plates that form the calibrated resistance in the center of the shunt, between these large terminals. Specifically, the resistance between the two smaller screw terminals is calibrated, and the sensing leads are connected to these smaller terminals.[6]

[6] This scheme is called a *Kelvin connection* and enables *four-wire voltage measurement*. Since essentially zero current is drawn by the voltage-sensing electronics, there is negligible voltage drop across the resistance of the smaller terminals, and current can be calculated as stated earlier. However, if one were to (mistakenly) connect the voltage-sensing wires to the larger screw terminals, the voltage drop of the large battery-pack current passing through the uncalibrated resistance of the large screw terminals would significantly degrade the accuracy of the current calculation.

The primary advantage of current shunts (over Hall-effect sensors, described next) is that they have no offset at zero current, regardless of temperature. This is of vital importance if one is updating an SOC estimate by integrating current into and out of a battery pack (this is known as *coulomb counting*). However, the amplification and measurement electronics can still introduce an offset, which must then be calibrated out of every measurement.

A disadvantage when using current shunts is that they must usually be electrically isolated from the main BMS circuitry (e.g., in an automotive application, the BMS is powered via an external 12 V supply that must be isolated from the high-voltage battery). This, plus amplification circuitry, adds complexity to the design. The resistance of the current shunt also changes with temperature, so temperature should be measured and resistance calibrated. In addition, the shunt itself introduces some energy losses, and the heat that is generated must be dissipated via the thermal management system.

If a coil is wrapped around a primary current-carrying conductor, the electromagnetic field produced by the conductor induces a secondary current in the coil. *Hall-effect sensors* measure this induced current to infer the primary current. Fig. 1.9 shows a photograph of a Hall-effect sensor (top frame) and a block diagram for current sensing using a Hall-effect sensor (bottom frame). The main battery-pack current-carrying wire passes through the oval opening in the center of the sensor—no direct electrical connection is made between the sensor and the high-voltage battery pack. This yields the distinct advantage that Hall-effect sensors are automatically isolated electrically from the high-voltage battery and so no special isolation circuitry is needed.



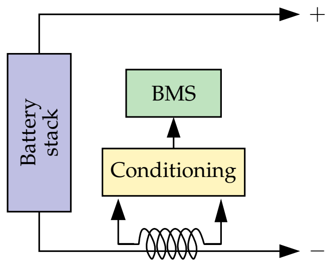


Figure 1.9: Sensing current with a Hall-effect sensor.

However, because Hall-effect sensors operate based on electromagnetic principles, magnetic hysteresis is inherent in the device. Feedback signal-conditioning circuitry can be devised to guard against this hysteresis, and some sensors come prepackaged with such circuitry. Even so, Hall-effect sensors suffer from at least some measurement offset at zero current, which drifts over time and which changes with temperature. Even if this offset is "zeroed out" in software (by subtracting an initial measurement) at some initial temperature during a BMS startup routine (when it is known that zero current is flowing since contactors are not yet closed), this ad hoc calibration does not correct for time- and temperature-varying drift in the bias. As the bias plays havoc with a number of BMS algorithms, some kind of compensation is necessary. For example, if the BMS ever knows for certain that zero current is flowing, the bias can be "zeroed out" at those times. Some HEV applications allow for this.

1.6 Requirement 1d: High-voltage contactor control

For safety reasons, as described in more detail in Sect. 1.7, high-voltage battery packs are designed to be isolated electrically from chassis ground. If someone were to touch chassis ground with one hand and either terminal of the battery pack with the other, the electrical isolation should be sufficient such that he or she is completely safe. This will not be true if there is an *isolation fault* or *ground fault*, which is why we investigate how to detect such faults in Sect. 1.7.

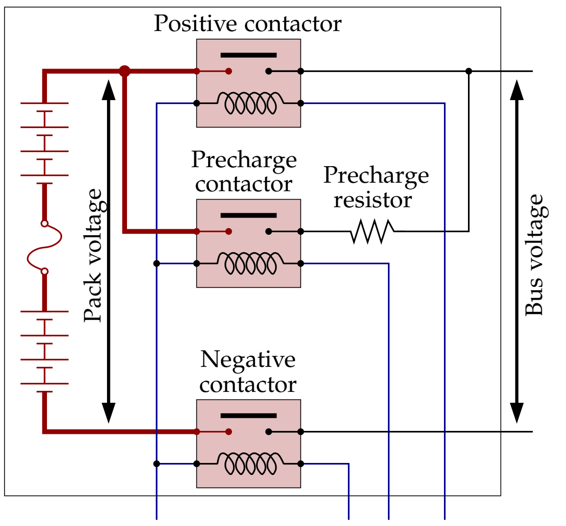
For similar safety concerns, the battery pack internal high-voltage bus is completely disconnected from the load at both battery-pack external terminals when not in use. This requires two high-current capable relays known as *contactors*. Contactors used in battery packs are normally-open devices, so if the BMS loses power for any reason, the terminals of the contactor are de-energized and the battery pack is disconnected from the load.

Battery loads are often capacitive. For example, in a vehicle application, the motor-driving circuitry requires large capacitors to filter the transients caused by switching power to the motor-drive coils. This causes a complication during the startup sequence that connects the battery pack to its load. Supposing that the capacitive load is in a discharged state, if both contactors were to be closed simultaneously, a huge spike of current would attempt to flow instantly, potentially welding the contactors closed or blowing a fuse.

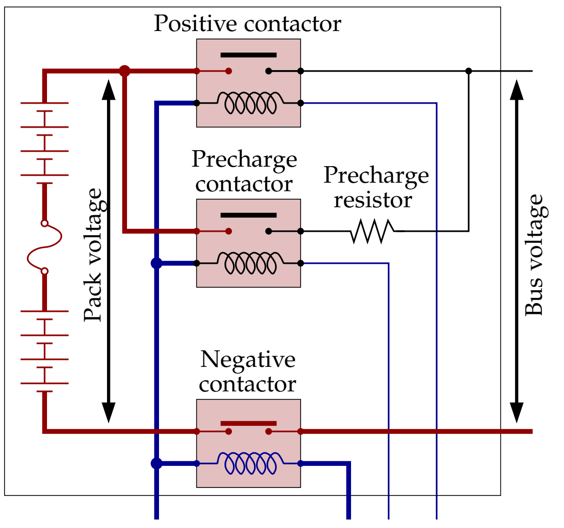
So, a third *precharge contactor* is used. The overall startup process is depicted in Fig. 1.10. In frame (a), the pack is initially disconnected from the load. All contactors are open and the thick red lines—indicating the power path—extend from the battery stack only to the internal side of the contactors.[7] The negative contactor is activated first, as illustrated in frame (b), where the thick blue lines indicate the low-voltage signal paths being activated. This connects the "$-$" terminal of the battery stack to the "$-$" terminal of the load.

[7] Incidentally, the figure also shows that a fuse is usually inserted in middle of the battery stack. If the fuse is blown for some reason, this limits the maximum voltage encountered by the service technician to at most half of the overall battery-stack voltage.

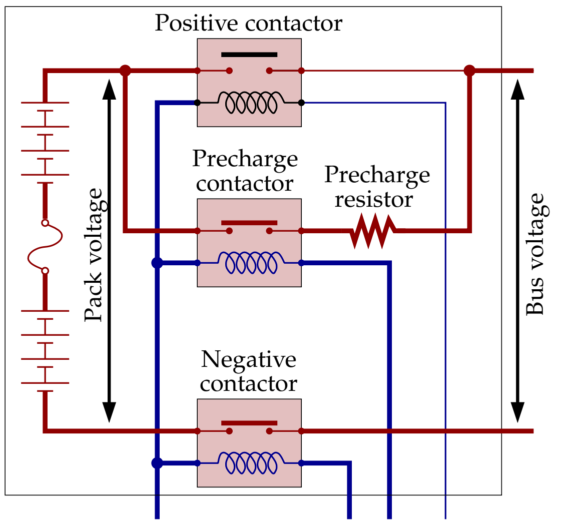
Next, the precharge contactor is activated (frame (c)). The precharge resistor limits current flow and allows the battery pack to charge up



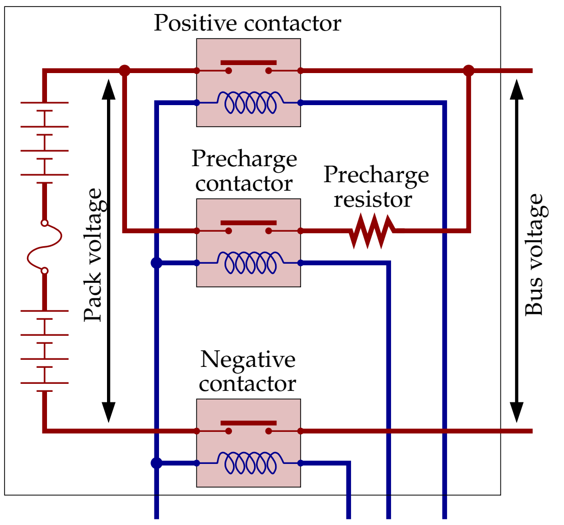
(a)



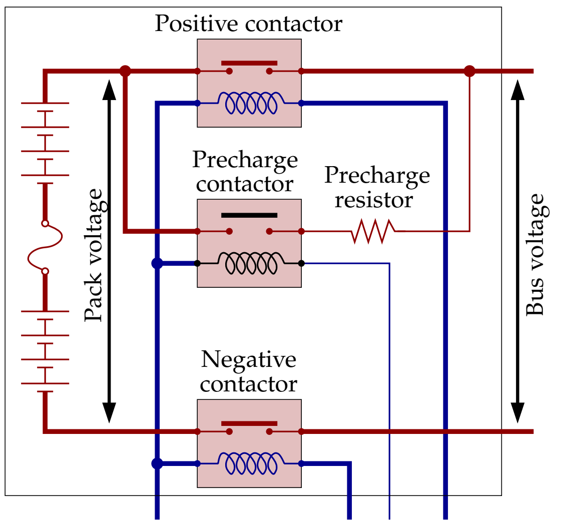
(b)



(c)



(d)



(e)

Figure 1.10: Startup sequence for connecting a battery pack to its load.

11:56:42.

the capacitive load at a controlled and safe rate. Often, the precharge-resistor temperature is monitored—if it becomes too high the load may have a short-circuit fault: The startup process is aborted and the pack is disconnected from the load. Similarly, the bus and battery-stack voltages are monitored. If these voltages don't converge to the same neighborhood after a specified interval, the load may have a short-circuit fault: The pack disconnects.[8]

[8] Measuring battery-stack and bus voltages requires high-impedance voltage dividers and isolation circuitry. Bus voltage can instead be inferred from stack voltage and pack current: when pack current approaches zero, we can conclude that bus voltage has approached stack voltage.

Assuming that bus and battery-stack voltages have become "close enough" "quickly enough," the BMS then closes the positive contactor (frame (d)). This connects the "+" terminal of the battery stack to the "+" terminal of the load. The load is now connected directly to the battery stack through a low-resistance path. The precharge contactor is then opened (frame (e)).

The procedure to follow when the battery pack is being shut down is not as clear. Since the capacitive load and the battery voltage have already come into equilibrium, the danger of contactor fusing/welding is not as severe. As long as the inductance of the load is small, simply opening the main contactors is probably sufficient. (If not, the startup procedure could be used in reverse to drain the inductive energy prior to complete disconnect of the battery pack.)

1.7 *Requirement 1e. Isolation sensing*

In a standard automobile, the negative terminal of the 12 V lead-acid battery is connected directly to the vehicle frame or *chassis*. This saves money on wiring as only the positive voltage requires separate wires to distribute power throughout the vehicle. It also does not pose a significant safety risk as it is unlikely that 12 V across a person's body would cause harm.[9]

[9] Recall the discussion from footnote 2 on what dc voltages are considered hazardous.

However, if someone were to touch both terminals of a high-voltage battery pack, injury or even death could occur. Therefore, battery-pack designers must take great care to minimize the likelihood of this happening. So, all wiring is fully insulated and at no point in the circuit is either high-voltage battery terminal connected to the highly exposed vehicle chassis.

This greatly enhances safety. However, if the vehicle were to be involved in an accident, or if vibration or any other kind of wear breaks through the insulation on the high-voltage wiring, it is possible that one of the terminals of the battery pack could come into contact with the vehicle chassis through a low-resistance path. This then poses a safety hazard, which we must detect in order to warn the vehicle operator of the danger.

The Federal Motor Carrier Safety Administration dictates a procedure for detecting loss of isolation.[10] The premise is that it should be

[10] These are documented in Federal Motor Vehicle Safety Standard FMVSS 305 and Society of Automotive Engineers standard SAE J1766.

safe for someone to touch either terminal of the battery pack and the chassis ground at the same time. The metric states that isolation is considered sufficient if less than 2 mA of current flows when connecting chassis ground to either the positive or negative terminal of the battery pack via a direct short circuit.

We derive this procedure with reference to Fig. 1.11. The battery pack should not be connected in any way to the chassis. In the circuit, both R_1 and R_2 should be infinite. However, it is possible that isolation has been lost. In that case, either R_1 or R_2 will become less than infinite. We call the lesser of R_1 and R_2 the *isolation resistance* R_i, and it is our primary objective here to determine thus resistance. According to the safety criterion, R_i must be greater than $v_b/0.002\,\text{A}$ or $R_i > 500v_b$, where v_b is the battery voltage.

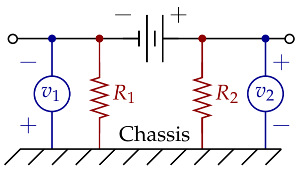


Figure 1.11: Step 1 of measuring isolation.

For the BMS to sense whether the pack is sufficiently isolated from the chassis, it must somehow determine R_i. To do so, we measure v_1 and v_2 using a high-impedance analog-to-digital measurement circuit, itself having impedance greater than 10 MΩ.[11] Inserting the sensor to make these measurements breaks strict isolation, but the high impedance of the sensor ensures that it will not violate the 2 mA current limit by itself.

[11] Note the polarity of the voltmeters—both v_1 and v_2 are positive.

Redrawing the circuit in Fig. 1.12, we see that R_1 and R_2 form a voltage divider across the total battery voltage. As we wish to find the smaller of the two resistances we solve for R_1 if $v_2 > v_1$; otherwise we solve for R_2. Note also that the current through both resistors in this circuit must be equal in order to satisfy Kirchhoff's current law. So, $v_1/R_1 = v_2/R_2$. We will use this identity shortly.

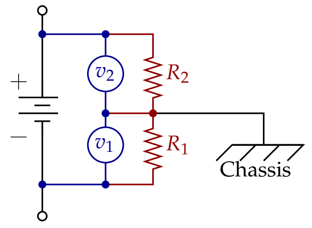


Figure 1.12: Step 1 of measuring isolation, redrawn.

1.7.1 *Potential isolation fault on negative side: Find R_1*

If there is a fault on the negative side of the battery pack, then $v_2 > v_1$. So, if we measure $v_2 > v_1$, we wish to solve for R_1 and then check to see whether it is sufficiently large to avoid an isolation fault.

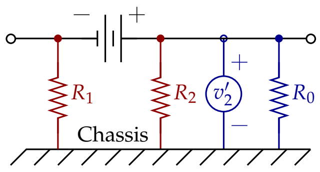


Figure 1.13: Step 2 of measuring isolation if fault is on the negative side.

To find R_1, we insert a known (large) resistance R_0 between the positive terminal of the battery and chassis ground via a transistor switch, as shown in Fig. 1.13. This again breaks strict isolation, but not enough to worry about if R_0 is "big enough" (i.e., $R_0 \gg 500V_b$).

We now measure v_2'. By Kirchhoff's current law, the sum of currents through R_0 and R_2 must equal the current through R_1, so

$$\frac{v_b - v_2'}{R_1} = \frac{v_2'}{R_2} + \frac{v_2'}{R_0}.$$

Substituting $v_b = v_1 + v_2$ and $R_2 = R_1(v_2/v_1)$ gives

$$\frac{(v_1 + v_2) - v_2'}{R_1} = \frac{v_2'}{R_2} + \frac{v_2'}{R_0}$$

$$= \frac{v_2'(v_1/v_2)}{R_1} + \frac{v_2'}{R_0}.$$

Finally, solving for R_1, we find

$$\begin{aligned} \frac{(v_1+v_2) - v_2' - v_2'(v_1/v_2)}{R_1} &= \frac{v_2'}{R_0} \\ R_1 &= \frac{R_0}{v_2'}(v_1 + v_2 - v_2' - v_2'(v_1/v_2)) \\ R_1 &= \frac{R_0}{v_2'}\left(1 + \frac{v_1}{v_2}\right)(v_2 - v_2'). \end{aligned} \tag{1.3}$$

In summary, if $v_2 > v_1$, there is a possible isolation fault on the negative side of the battery. We then measure v_2' and compute $R_i = R_1$ from v_1, v_2, and v_2' using Eq. (1.3). Isolation is deemed sufficient if $R_i > 500v_b$.

1.7.2 *Potential isolation fault on positive side: Find R_2*

The procedure is similar if $v_1 > v_2$, except that now we want to find $R_i = R_2$. We insert a known large resistance R_0 between the negative terminal of the battery and chassis ground using a transistor switch, as shown in Fig. 1.14, and measure v_1'.

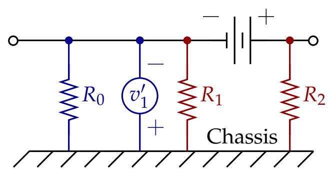


Figure 1.14: Step 2 of measuring isolation if fault is on the positive side.

Again, by Kirchhoff's current law,

$$\frac{v_b - v_1'}{R_2} = \frac{v_1'}{R_1} + \frac{v_1'}{R_0}.$$

Substituting $v_b = v_1 + v_2$ and $R_1 = R_2(v_1/v_2)$ gives

$$\begin{aligned} \frac{v_1 + v_2 - v_1'}{R_2} &= \frac{v_1'(v_2/v_1)}{R_2} + \frac{v_1'}{R_0} \\ \frac{v_1 + v_2 - v_1' - v_1'(v_2/v_1)}{R_2} &= \frac{v_1'}{R_0}. \end{aligned}$$

Solving for R_2, we find

$$\begin{aligned} R_2 &= \frac{R_0}{v_1'}(v_1 + v_2 - v_1' - v_1'(v_2/v_1)) \\ R_2 &= \frac{R_0}{v_1'}\left(1 + \frac{v_2}{v_1}\right)(v_1 - v_1'). \end{aligned} \tag{1.4}$$

In summary, if the potential isolation fault is on the positive side of the battery, we measure v_1, v_2, and v_1', and compute $R_i = R_2$ using Eq. (1.4). Isolation is deemed sufficient if $R_i > 500v_b$.

It is always wise to design with fault-tolerance and built-in test circuitry in mind. In the case of the isolation-sense requirement, we can test the function of the circuitry without changing the circuit design from what we have discussed already. We simply activate the transistor switches to insert resistance R_0 between both battery-pack terminals and chassis ground at the same time. We measure v'_1 and v'_2: they should both be equal to $v_b/2$ to within the tolerance of the resistors used for R_0.

1.8 Requirement 1f. Thermal control

In this book, we do not go into detailed thermal management and control strategies. However, they are very important aspects of BMS design as they address some significant battery-pack longevity and safety concerns.[12]

In general, lithium-ion cells last longest if they are maintained in a temperature band from about 10 °C to 40 °C.[13] Some of the degradation mechanisms that are accelerated by temperature effects are discussed qualitatively in Chap. 4.

[12] See, for example, Santhanagopolan, S., Smith, K., Neubauer, J., Kim, G-H, Keyser, M., and Pesaran, A., *Design and Analysis of Large Lithium-Ion Battery Systems*, Artech House, 2015, which has much more in-depth discussion of battery-pack thermal requirements.

[13] A good rule of thumb is "if you are comfortable at a certain temperature, the battery pack is also 'comfortable' at that temperature."

As we saw in Volume I of this series, models of heat generation in a battery cell can be complicated. However, irreversible and joule heating terms tend to dominate (especially over short timescales with random-looking input current), so heat generation can be crudely approximated as i^2R where i is cell current and R is a thermal resistance. Therefore, when the battery pack is sourcing or sinking a large amount of current per cell, heat generation is high.

In an EV, while the total battery-pack current is high, the relative C-rate per cell is low as the battery pack must have high total capacity for acceptable vehicle driving range. Internal heat generation is also therefore low. So, air cooling may be sufficient.

However, in an HEV, C-rates are high. Therefore, liquid cooling may be necessary. In any xEV application, careful coupled thermal/electrical simulation and analysis of battery-pack operation should be conducted in the design phase to determine the correct size and type of thermal management system.

While it is common (even universal) for a thermal-management system to be required to cool cells, it is rare for one to consider heating cells. Even if a vehicle were started when the ambient temperature is quite low, the i^2R self-heating tends to bring the battery pack to a reasonable operating temperature fairly quickly. Some plug-in vehicles, however, do heat a too-cold battery pack using grid power while plugged in, but to do so using the pack's own power when not plugged in would deplete the available energy and therefore driving range, so there is little reason to do so.

1.9 *Requirement 2. Protection*

Referring back to the list of BMS design requirements enumerated in Sect. 1.2, we have now completed our discussion of Requirement 1: Sensing and high-voltage control. We now quickly consider Requirement 2: Protection.

Energy storage in any form is potentially hazardous. If energy were to be released in an uncontrolled way, there could be catastrophic consequences. Because of long experience, we have grown accustomed to the risks associated with gasoline-powered vehicles, and understand very well how to minimize the likelihood and impact of those hazards.

We are still learning how to control the risks associated with energy storage in high-capacity battery packs. The electronics and software of the BMS are integral parts of an overall risk-management strategy. They must provide monitoring and control to protect cells from out-of-tolerance ambient operating conditions, and to protect the user from consequences of a battery failure. There are large challenges involved. For example, if a cell develops an internal short circuit, hundreds of amperes can develop in microseconds. Therefore, protection circuitry must act *very* quickly to isolate the fault; otherwise, the cell may go into thermal runaway, resulting in a battery-pack fire or explosion. Safety management for high-capacity battery packs is a developing field, as conventional methods using circuit elements such as fuses tend to act too slowly, and new devices and methodologies are needed.[14]

[14] If a single cell goes into thermal runaway, there may be no way to stop it if, for instance, there is an internal short in the cell caused by a impurity in the cell material. Opening the main contactor may not stop the event. However, the overall battery-pack design should be such that thermal runaway in one cell will not cascade to other cells and such that all gases are vented. This objective can be achieved by mechanically isolating cells. A clever fusing approach can also help: see Kim, G-H, Smith, K, Ireland, J, and Pesaran, A., "Fail-safe design for large capacity lithium-ion battery systems," *Journal of Power Sources*, 210, 2012, pp. 243–253.

In a battery pack, protection must address the following undesirable events or conditions: Excessive current during charging or discharging, short circuit, overvoltage or undervoltage, high ambient temperature or overheating, loss of isolation, and abuse. Whenever possible, fallback protection paths should be implemented.

For example, consider Fig. 1.15, which illustrates one way to think about designing protection mechanisms for a battery pack where current and temperature are the input variables. In the figure, the area shaded red corresponds to the cell-manufacturer-specified operating region where cells will most likely be subject to permanent damage. Anywhere else is considered to be "okay," but we need a margin of error in the design; hence, we generally design protection to limit the cell's operating conditions to smaller "safe" region, shown in green. Safety devices are then specified to constrain cells to the safe region leaving a safety margin, shown in white. Note that to traverse from the green region to the red region, two protection systems must fail. This redundancy helps provide a robust protection scheme.

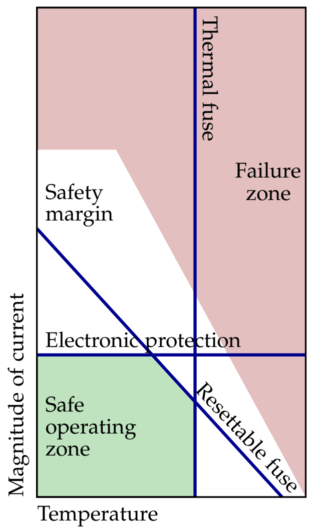


Figure 1.15: Designing current/temperature protection.

Fig. 1.16 is a similar diagram for a preliminary protection design where voltage and temperature are the input variables. In some cases, it is possible to cross only one boundary to move from the safe zone to the failure zone. To complete the design, additional protection elements should be added.

Examples of protection devices include thermal fuses (which open the contactor when temperature is above some limiting temperature), conventional fuses (which sever an electrical connection if too much current flows for too long, but may not operate quickly enough for some kinds of faults), and electronic fault detection. For the latter, the BMS continuously senses voltage, current, and temperature and takes design-specified action if a fault is detected. In Fig. 1.16 we also notice that the plug-in charger can independently monitor battery-pack voltage when the pack is being charged and automatically stop charging if it detects that the overall battery-pack voltage is too high. Similarly, the load controller can monitor voltage and stop utilizing the battery pack if the voltage becomes out of bounds.[15]

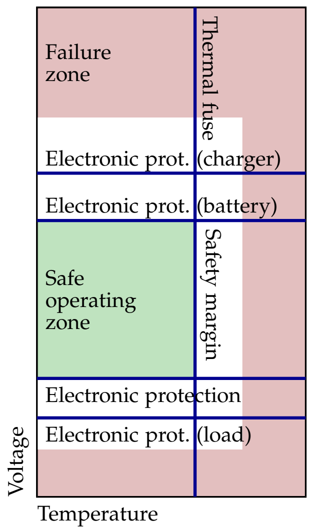


Figure 1.16: Designing current/temperature protection.

[15] Fault-detection and reaction strategies are outside the principal scope of this book. Some references that include a deeper discussion of these topics include: Weicker, P., *A Systems Approach to Lithium-Ion Battery Management*, Artech House, 2014, and Santhanagopolan, S., Smith, K., Neubauer, J., Kim, G-H, Keyser, M., and Pesaran, A., *Design and Analysis of Large Lithium-Ion Battery Systems*, Artech House, 2015.

1.10 Requirement 3a. Charger control

BMS requirement 3 considers communications interfacing between the battery management system and the application being powered by the battery pack. In an xEV, that application is the vehicle itself, which is managed via the vehicle control computer (which fills the role of the battery-load control computer in Fig. 1.4).

The first communication interface we consider is to the battery-pack charger. Battery packs in xEV can receive charge in two ways. All xEV packs experience *random charging*, where charge is delivered in unpredictable patterns; for example, via energy recovery because of regenerative braking. The BMS controls the maximum allowed level of random charging by continuously informing the vehicle of the present operating charge-power limit that can be supported by the battery pack. This is discussed more in Chap. 6. In addition, EV, PHEV, and E-REV have plug-in modes so can support *plug-in charging*. This is a carefully regulated process where charge power is provided from the electric utilities grid, and much finer control over the charging protocol can be implemented. Often, some sequence of constant-power steps (of decreasing magnitude) are applied until the battery pack is considered to be fully charged. Battery-pack equalization, as studied in Chap. 5, is also frequently performed during plug-in charging.

The speed with which a modern battery pack can be charged depends far less on the battery itself than it does on the utility service feeding the charger. A quick back-of-envelope example should illustrate this. Consider a small passenger vehicle: depending on aerodynamics and so forth, the rate of energy usage is on the order of 200 Wh mile^{-1} to 300 Wh mile^{-1}. For a 300-mile range, the battery pack must have between 60 kWh to 90 kWh useable capacity. If we were to charge this battery pack in 3 min, roughly the time required to fill a passenger-vehicle gasoline tank, the utilities service would need to provide power at a rate of 1.8 MW, continuously! Current mass-produced battery technologies cannot withstand such a fast charge but neither can residential utilities.

Approaching the problem from the opposite point of view, consider a standard domestic service of 110 V at 15 A, or about 1.5 kW. This is typical of most residential wall outlets. At this rate, it would take between 40 h and 60 h to fully charge the battery pack, which is not acceptable. However, consider an upgraded domestic service of 220 V at 30 A, or 6.6 kW. This is typical of an outlet to supply an electric stove or an electric clothes dryer. At this rate, it would take between 10 h and 15 h to fully charge the battery pack. This is actually quite a good compromise for daily driving, as the car is rarely driven its full range and often has overnight to recharge. So-called "fast charge" would be needed only on extended-length trips of greater distance than the vehicle range, and while stopping 10 or 15 min to charge a vehicle battery pack is longer than the wait with which the consumer is accustomed, it is still within reasonable limits if required only infrequently.

1.11 *Requirement 3b. Communication via CAN bus*

The second topic we consider under the heading of communications is the protocol used for the communications. This will, of course, vary with the application, but automotive applications use the *control-area network (CAN)* protocol almost exclusively for on-board vehicle messaging. CAN is designed to provide robust communications in the very harsh automotive operating environments with high levels of electrical noise.

CAN has an electrical specification and a packet protocol. Electrically, it comprises a differential two-wire serial bus designed to network intelligent sensors and actuators. Messaging can operate at two rates: high-priority messages are sent at a higher baud rate, while low-priority messages are sent at a lower rate. High-rate messaging is used for critical operations such as engine management, vehicle stability, and motion control, while low-rate messaging is used for simple switching and control of lighting, windows, mirror adjustments, and instrument displays, etc.

The communication protocol defines the following: The method of addressing the devices connected to the bus; transmission speed and

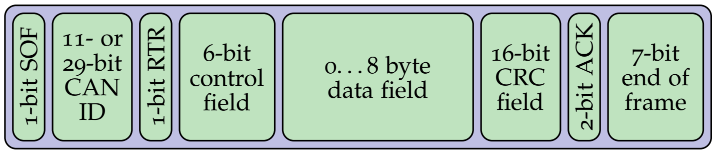


Figure 1.17: A CAN 2.0 data frame.

priority settings; transmission sequence; error detection and handling; and control signals. Data frames, such as illustrated in Fig. 1.17, are transmitted sequentially over the bus. In the frame, there is a 1-bit start-of-frame (SOF) marker, an 11- or 29-bit address for the intended recipient of the message, a 1-bit remote-transmit request (RTR) flag indicating whether this is a data frame or a remote frame, a 6-bit control field, up to eight bytes of message to be transmitted, a 16-bit cyclical-redundancy-check (CRC) for transmission error detection, a 2-bit acknowledgment (ACK) field, and a 7-bit end-of-frame (EOF) marker.[16]

[16] Fortunately, many automotive-grade microcontrollers have built-in CAN communications hardware that take care of the details involving the protocol, leaving the BMS designer free to design messages and responses to meet the requirements of the application.

1.12 *Requirement 3c. Log book function*

A third requirement that we list under the communication category is to provide a log-book function as well as a means of accessing these data. For warranty and diagnostic purposes, the BMS must store a log of atypical and abuse events. This log should include a record of the type of abuse (e.g., out-of-tolerance voltage, current, or temperature) and the duration and magnitude of the abuse. At a minimum, a vehicle technician should be able to gain access to this log for vehicle-diagnostics, warranty, and post-crash analysis purposes.

It is also necessary to be able to store persistent diagnostics regarding, for example, the number of charge/discharge cycles completed, the battery-state and SOH (battery-model parameter values) estimates for every cell at the end of each driving cycle, and more. These data are stored in a history chip (e.g., flash memory).

1.13 *Requirement 4a. State of charge estimation*

Referring back to Sect. 1.2, BMS design requirement 4 has to do with performance management. Under this umbrella, we introduce SOC, energy, and power estimation in this chapter. Later, in Chaps. 3 and 6, we will explore some of these in much greater depth.

1.13.1 *What needs to be estimated and why?*

Different applications impose different battery-pack performance-management priorities. To function reliably, xEVs need to have continuous updates for the estimates of two fundamental battery quantities: How much energy is available in the battery pack and how much power is available in the immediate future.

An estimate of energy is most important for EV. Energy is an ability to do work and is a total quantity measured in Wh or kWh. Energy is a fundamental input to vehicle range calculations—it tells you how far you can drive.

An estimate of power is most important for HEV. Power is the rate at which energy can be moved from the battery pack to the wheels (or vice versa) without exceeding cell or electronics design limits and is an instantaneous quantity $p = iv$ in W or kW. Power tells you whether you can accelerate or accept braking charge.

Estimates of both energy and power are important for E-REV/PHEV. The energy estimate is used when calculating remaining range in charge-depletion mode, and the power-limit estimate is used to balance engine and motor demand during charge-sustaining mode.

Ideally, we would like to be able to *measure* available energy and power directly and then report these values to the vehicle computer. Unfortunately, no such sensor exists.[17] Instead, we must compute estimates of available energy and power using more basic quantities as inputs to the computation.

[17] For a gasoline-powered vehicle, an (approximate) energy sensor does exist. A float in the gasoline tank measures the remaining fuel, which is proportional to available energy. Sadly, there is no such "charge float" for a battery cell.

To be able to compute energy, we must know (at least) the present value of all cell states of charge $z_k^{(i)}$ and total capacities $Q_k^{(i)}$, where subscript k denotes the present time index and superscript (i) denotes the ith cell in the battery pack. To compute power, we must know (at least) all cell states of charge and resistances $R_k^{(i)}$. This is illustrated in the right-hand-side of Fig. 1.18.

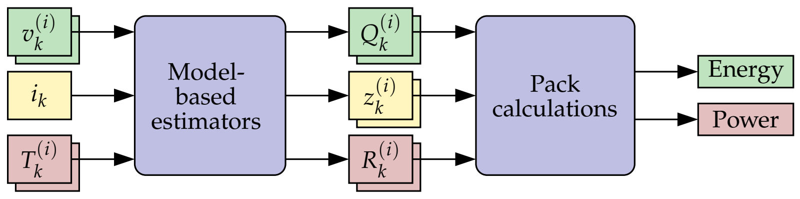


Figure 1.18: The data-flow diagram for estimating available energy and power.

However, we cannot measure these parameters directly, either; we must estimate them as well. Available inputs to this estimation process include all cell voltages $v_k^{(i)}$, pack current i_k, and cell temperatures $T_k^{(i)}$.

In Chaps. 3 and 4, we will see that there are both good and poor methods to produce state and parameter estimates from these measurable quantities. The poor methods are generally simpler to understand, code, and validate, but yield less accurate results. The impact of this can be very costly. For example, if you "floor" the accelerator pedal to pull out of your lane and pass a semitruck, and if the poor battery-controls algorithm has indicated to the vehicle that ample battery power is available, the vehicle will rely heavily on the battery pack to enable this acceleration. If the battery pack then discovers that some voltage or current limit has been exceeded because its power estimate was inaccurate, some kind of corrective action must be taken. The battery pack may decide to decrease allowed power abruptly, leading to customer perception of poor drivability (you certainly didn't pass the semi truck, and you may not even have survived the event if another vehicle was oncoming). Or, the battery pack may decide to allow the overcharge or overdischarge event in order to protect the safety of the driver, but this will degrade the battery cells prematurely. Most often, battery packs are overdesigned in the first place to compensate for uncertainty in the state of charge, energy, and power estimates.

All three of these consequences of a poor battery-controls algorithm have a cost in dollars, battery-pack weight and/or volume. *The fundamental underlying premise of this book is that investing in good battery management and control algorithms—and electronics capable of implementing them—can reduce pack size and return a considerable overall net savings.* For this reason, our focus is on the best available algorithms even though some of these are fairly complex.

1.13.2 What really is state of charge (SOC)?

We consider SOC estimation in detail in Chap. 3. However, it is helpful to have a good intuitive understanding of what this term means before we get to that point.

SOC is a concept that we explored in Volume I of this series in the contexts of both equivalent-circuit and physics-based battery-cell models. Physically, we recall that charging a cell moves lithium from the positive-electrode solid particles to the negative-electrode particles, and discharge does the opposite. Electrochemically, cell SOC is then positively related to the average concentration of lithium in the negative-electrode solid particles (and negatively related to the average concentration of lithium in the positive-electrode solid particles).

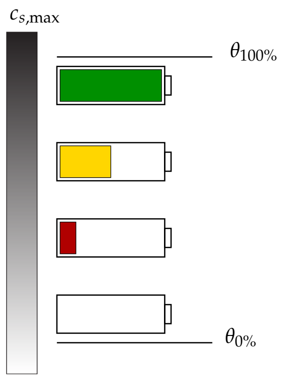


Figure 1.19: Relationship between negative-electrode average lithium concentration and cell SOC.

For simplicity, we consider the negative electrode only and define the present average lithium concentration stoichiometry as the ratio of the average lithium concentration to the maximum possible lithium concentration in the electrode materials, or $\theta_k =$

$c_{s,\text{avg},k}/c_{s,\max}$ at time index k. Based on cell-manufacturer-specified design limits, this stoichiometry is intended to remain between fixed values $\theta_{0\%}$ and $\theta_{100\%}$. Then, an equation for cell SOC is: $z_k = (\theta_k - \theta_{0\%})/(\theta_{100\%} - \theta_{0\%})$.

Since we cannot measure θ_k or $c_{s,\text{avg},k}$ directly, it is reasonable to wonder whether there exists a coupling between immeasurable SOC and measurable cell voltage. Perhaps we can infer SOC by measuring voltage? However, note that cell voltage depends on temperature and electrode particle *surface* concentrations but SOC depends on particle *average* concentrations. Surface and average concentrations will not generally be the same. Further, changing temperature changes cell voltage but not average concentrations so does not change SOC; resting a cell changes its voltage but not average concentrations so does not change SOC; and history of cell usage changes steady-state surface concentration versus average concentration (hysteresis), which makes determining SOC using only voltage problematic.

SOC changes only because of passage of current, either charging or discharging the cell via an external circuit or self-discharge within the cell. However, voltage changes both for these reasons and a number of others. We will find voltage useful as an indirect indicator of SOC but not as a direct measurement of SOC.

Then, how about inferring state of charge via measurements of current? We know from Volume I that SOC for cell i at time index k is related to its initial value and to cell current i_k via

$$z_k^{(i)} = z_0^{(i)} - \sum_{j=0}^{i-1} \frac{1}{Q_k^{(j)}} \eta_k^{(j)} i_k, \tag{1.5}$$

where cell current is positive on discharge and negative on charge, $\eta_k^{(i)}$ is cell coulombic efficiency (which is usually very close to but slightly less than unity), and $Q_k^{(i)}$ is the cell total capacity in ampere seconds (coulombs).[18] Estimating SOC via this relationship is called *coulomb counting*. While Eq. (1.5) is correct, we will see in Chap. 3 that SOC estimation via coulomb counting has some serious limitations.

[18] Note, that total capacity $Q_k^{(i)}$ is a measure of the number of locations in the electrode structure between $\theta_{0\%}$ and $\theta_{100\%}$ that could hold lithium. It is not a function of temperature, rate, etc., as we will discuss further in Chap. 4.

We make one final point in this section concerning the idea of "battery-pack SOC." Consider Fig. 1.20, which shows a simple two-cell battery pack with the lower cell having SOC equal to 100 % and the upper cell having SOC equal to 0 %. How should we define the "battery-pack SOC"?

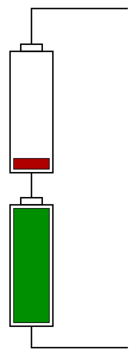


Figure 1.20: Illustrating that "battery-pack SOC" is a concept that does not make physical sense.

We might define "battery-pack SOC" to be 0 % because we cannot discharge without overdischarging the top cell. However, a value of 0 % implies that we can charge the battery pack, and we know that we cannot do so without overcharging the lower cell. Therefore, a definition of 0 % is not meaningful.

Or, we might define "battery-pack SOC" to be 100 % because we cannot charge without overcharging the bottom cell. However, that implies that we can discharge the battery pack, which we cannot do without overdischarging the top cell. Therefore, a definition of 100 % is not meaningful either.

Instead, we might define "battery-pack SOC" to be the average of the two, 50 %. This may be worst of all because it implies that we can both charge and discharge the battery pack. All of these definitions are deficient for the simple reason that "battery-pack SOC" is an ill-defined term and should not be used.

Of course, this example is extreme, but it does point out a need for cell balancing, which we will explore in Chap. 5. It also brings up the question regarding why "battery-pack SOC" might even be something we desire to know. One possible reason is for SOC setpoint control in HEV: the average of all cell SOCs will work well for this if the battery pack is reasonably well balanced (but we should call this a "battery-pack-*average* SOC" and not a "battery-pack SOC"). A second application is as an input to a dashboard fuel gauge. However, the real issue is not SOC but available energy, so the gauge should be an indicator of total energy instead.

1.14 Requirement 4b. Energy estimation

1.14.1 Cell total energy estimation

Cell total energy is equal to

$$e_k^{(i)} = Q_k^{(i)} \int_{z_{\min}}^{z_k^{(i)}} \mathrm{OCV}(\xi)\, \mathrm{d}\xi,$$

where $\mathrm{OCV}(\cdot)$ is the cell's *open-circuit-voltage (OCV)* relationship as a function of SOC. For example, Fig. 1.21 plots the OCV versus SOC relationship for six different lithium-ion cell chemistries. The integral computes the area under the OCV curve between the minimum permitted SOC and the present cell SOC. If $Q_k^{(i)}$ is measured in ampere-hours, then $e_k^{(i)}$ is measured in watt-hours.

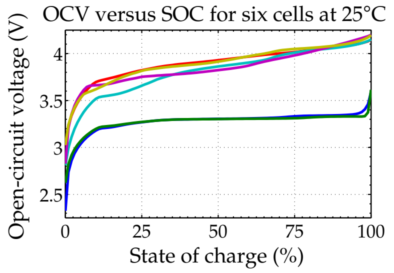


Figure 1.21: OCV versus SOC relationships for six cells.

The integral can be precomputed for different upper integration limits and stored in a lookup table for efficient implementation in real time. Or, if a slight inaccuracy can be permitted, we can approximate cell energy as

$$e_k^{(i)} \approx Q_k^{(i)} v_{\mathrm{nom}} \left(z_k^{(i)} - z_{\min} \right), \tag{1.6}$$

where v_{nom} is the nominal cell voltage.

Note that cell total energy is not a function of discharge rate, nor is it a strong function of temperature.[19] However, it is impossible to get all the of the total energy out of the battery pack at high rates and

[19] The only temperature dependence is via the small temperature-based variation in the open-circuit voltage versus state of charge relationship.

11:56:42.

cold temperatures without violating design voltage limits on the cell, which is why we need power estimates as well.

1.14.2 *Battery-pack total energy estimation*

To compute battery-pack total energy (as opposed to individual cell total energy, which we have just seen), we must take into consideration that cells may have different present SOCs and different total capacities. So, with reference to Fig. 1.22, we undertake a thought experiment asking, "How many ampere-hours may be discharged from the battery pack before the first cell reaches the lower SOC design limit z_{min}?" We can calculate this as

$$Q_k^{\text{dis}} = \min_i \left(Q_k^{(i)} \left(z_k^{(i)} - z_{\text{min}} \right) \right).$$

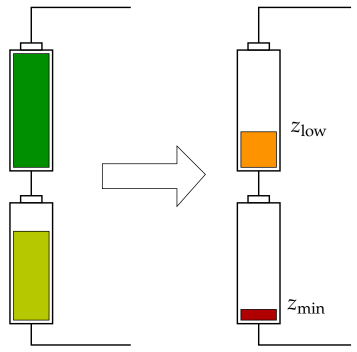


Figure 1.22: Illustrating the calculation of battery-pack total energy.

If Q_k^{dis} ampere-hours were discharged from the battery pack, only one cell would reach the lower SOC design limit; the general expression for the SOC that would be attained by any cell is

$$z_{\text{low},k}^{(i)} = z_k^{(i)} - \frac{Q_k^{\text{dis}}}{Q_k^{(i)}}.$$

Then, we can compute the total energy removed from the battery pack as the sum of the energy removed from each cell:

$$e_{\text{pack},k} = \sum_i Q_k^{(i)} \int_{z_{\text{low},k}^{(i)}}^{z_k^{(i)}} \text{OCV}(\xi)\, d\xi.$$

Again, we note that integrated OCV can be stored in a lookup table for instant computation in a real-time implementation.

1.15 *Requirement 4c. Power estimation*

1.15.1 *Cell power estimation*

Charging or discharging a cell at a high power level will accelerate cell degradation and lead to premature battery-pack failure. Therefore, power limits are computed to provide reasonable estimates of how much power is available over the next ΔT seconds without causing premature degradation to the battery pack. For now, we will preview a simple approach to computing power estimates to enforce design limits on cell voltage, known as the *Hybrid Pulse Power Characterization (HPPC)* method. In Chaps. 6 and 7, we will talk about more advanced methods.

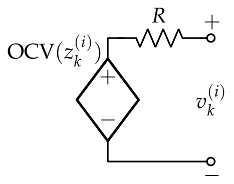


Figure 1.23: Crude equivalent-circuit model of a cell.

Here, we assume the simplified equivalent-circuit model of a cell shown in Fig. 1.23. For this model, we have that

$$v_k^{(i)} = \text{OCV}(z_k^{(i)}) - i_k R_k^{(i)},$$

or

$$i_k = \frac{\text{OCV}(z_k^{(i)}) - v_k^{(i)}}{R_k^{(i)}}.$$

To compute a power estimate, we first assume that we are concerned only with keeping the terminal voltage between $v_{\min}$ and $v_{\max}$. Then, discharge power is computed by clamping the cell terminal voltage to $v_{\min}$:

$$p_{\text{dis},k}^{(i)} = v_k^{(i)} i_k = v_{\min} \frac{\text{OCV}(z_k^{(i)}) - v_{\min}}{R_k^{(i)}}.$$

For this prediction to be reasonable, the value of $R_k^{(i)}$ should *not* be chosen equal to the instantaneous resistance R_0 from a more comprehensive equivalent-circuit model. Instead, the voltage drop $i_k R_k^{(i)}$ should be representative of the total voltage drop that would be observed over ΔT seconds if a constant-current pulse were applied. A laboratory cell test such as the one illustrated in Fig. 1.24 should be performed. In the figure, the cell has been allowed to rest for 10 seconds, then a constant-current discharge pulse is applied for ΔT seconds ($\Delta T = 10\,\text{s}$ in this example), then the cell voltage is allowed to recover and a constant-current charge pulse is applied for ΔT seconds. We measure ΔV_{dis} as the rest voltage minus the minimum voltage during the constant-current discharge, and ΔV_{chg} as the rest voltage minus the maximum voltage during the constant-current charge. Then, we compute (positive) effective discharge and charge resistances as

$$R_{\text{dis},\Delta T}^{(i)} = \left| \frac{\Delta v_{\text{dis}}^{(i)}}{i_{\text{dis}}} \right|, \qquad \text{and} \qquad R_{\text{chg},\Delta T}^{(i)} = \left| \frac{\Delta v_{\text{chg}}^{(i)}}{i_{\text{chg}}} \right|.$$

So, to compute cell discharge power, we set $R_k^{(i)} = R_{\text{dis},\Delta T}^{(i)}$ and clamp $v_k^{(i)} = v_{\min}$. Then,

$$p_{\text{dis}}^{(i)} = v_{\min} \frac{\text{OCV}(z_k^{(i)}) - v_{\min}}{R_{\text{dis},\Delta T}^{(i)}}. \tag{1.7}$$

Similarly, to compute cell charge power, we set $R_k^{(i)} = R_{\text{chg},\Delta T}^{(i)}$ and clamp $v_k^{(i)} = v_{\max}$. Then,

$$p_{\text{chg}}^{(i)} = v_{\max} \frac{\text{OCV}(z_k^{(i)}) - v_{\max}}{R_{\text{chg},\Delta T}^{(i)}}. \tag{1.8}$$

Note that this quantity is negative.

The power estimates produced by this method are fairly poor because of the underlying crude cell model and the implicit assumptions that the cell is at rest before power is demanded. Hence, it is common to derate the estimate by some constant multiplicative factor less than 1 to produce a more conservative estimate. Accuracy can be improved somewhat if the pulse testing is run at multiple cell SOC and temperature setpoints, and thus different $R_{\text{dis},\Delta T}$ and $R_{\text{chg},\Delta T}$ values are used for different operational conditions.

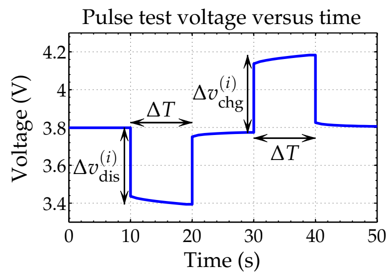


Figure 1.24: Determining the resistances $R_{\text{dis},\Delta T}$ and $R_{\text{chg},\Delta T}$ via a lab test.

1.15.2 *Battery-pack power estimation*

Assuming that the battery pack comprises cells wired in series, battery-pack power is computed by multiplying the minimum absolute cell power by the number of series-connected cells:

$$\begin{aligned} p_{\text{dis}} &= N_s \min_i p^{(i)}_{\text{dis}} \\ p_{\text{chg}} &= N_s \max_i p^{(i)}_{\text{chg}}. \end{aligned}$$

Note that we must use "max" instead of "min" when computing battery-pack charge power since charge power is negative.

1.16 *Requirement 5. Diagnostics*

The fifth category of battery-management-system requirements, per Sect. 1.2, is to compute diagnostics and report a SOH estimate for the battery pack. SOH estimation generically refers to a process whereby cell-model internal parameters are estimated and tracked as the cell ages and SOH is somehow a measure of the changes to these parameter values. While there is no universally agreed-upon definition of SOH, it is common to estimate two measurable indicators of aging: cell present total capacity and series resistance. Over the service life of a battery pack, capacity generally decreases between 20 % to 30 %, and resistance generally increases between 50 % to 100 %. Capacity change is also known as *capacity fade*, and resistance change is also known as *power fade* (as we have seen, resistance is an important factor in the power calculation). Therefore, estimating $R^{(i)}_k$ and $Q^{(i)}_k$ as the pack operates will give indicators of life. We study this in Chap. 4.

1.17 *Where to from here?*

The focus of the rest of the book is on how to estimate the battery internal status from simple voltage–current–temperature measurements and how to control battery operation for an optimal tradeoff between life and performance. These topics are organized as follows:

- All future discussion requires a more detailed understanding of how battery packs operate and how to represent that mathematically. So, our next step is to review some helpful battery-cell models from the Volume I text and see how to use them to simulate battery packs. This is the principal topic of Chap. 2.

- In Chap. 3, we will see how to use these models together with measurements of voltage, current, and temperature to estimate the internal state of all battery cells. We are particularly concerned with estimating SOC. There are some simple but poor ways to do this which we quickly survey. Most of our attention is focused on some higher-performance methods using nonlinear Kalman filters, with thought given to robustness and computational efficiency.
- In Chap. 4, we study how to estimate battery SOH. This is summarized primarily by the resistance increase and capacity decrease of the cells in the battery pack. We will discover that there are some straightforward ways to estimate resistance but that estimating total capacity well must be done very carefully.
- In Chap. 5, we look at the need for cell balancing (or equalization) and a number of different approaches to implement balancing. Design tradeoffs are considered.
- In Chap. 6, we examine methods for estimating available power when battery-cell voltage is the primary limiting concern. We study a simple but common static approach first, and then explore a more accurate dynamic algorithm.
- In Chap. 7, we discuss the essential problems with voltage-based power-limits computation and sketch out some ideas for more advanced power-limits calculations. (Detailed discussion is a topic of Vol. III of this series.)

Before proceeding we also note that many (or even most) of the methods we talk about for battery-state estimation and controls are patented and owned by xEV-related companies. This is true even of methods commonly found in the literature—most have been developed by companies for their own use. This fact strongly motivates research to develop methods that are sufficiently different from those that have been patented so that they may be implemented freely (or, so that you may patent them!). But, it also means that you may not use these methods commercially without license from the patent owner.

2
Simulating Battery Packs

2.1 Modeling battery cells

An important function of a battery management system is to compute estimates of a number of fundamental quantities including: SOC, SOH, available power, and available energy. The best methods to produce these estimates require high-fidelity but computationally simple mathematical sets of equations or *models* of cell *input/output* (current/voltage) dynamics. We believe that future applications will also require insight into cell *internal* electrochemical dynamics (e.g., to predict and minimize aging by understanding and controlling the internal degradation mechanisms).

There are two fundamentally different kinds of models that can be used by these estimation tasks. Both are able to describe the operation of lithium-ion battery cells in some regards.

Equivalent-circuit models (ECMs): ECMs represent the operation of a lithium-ion cell by proposing an electrical circuit as an analog to cell behaviors. Data collected from cells via lab tests are used to optimize circuit-element parameter values so that the current/voltage behaviors of the model closely match that of the true cell. Because ECMs amount to an empirical fit of data collected from a cell to a model structure comprising electronic circuit components, they share the same features of other types of curve fit. For example, ECMs tend to make good predictions when interpolating among the data seen when creating the model. This happens when the cell is exercised with profiles of current that are similar to those used to fit the parameter values. On the other hand, ECMs do not tend to extrapolate well. If the cell is being operated very differently from the lab-test scenario, its predictions should not be trusted. An ECM can predict input/output (current/voltage) behavior only; it cannot predict internal electrochemical states. But, ECMs yield fast and robust simulations.

Physics-based models (PBMs) : PBMs derive equations that model cell operation starting from first-principles physical laws that describe the response of internal cell electrochemical variables to an input-current stimulus. Cell voltage can then be computed from these internal electrochemical variables if desired. PBMs can predict over a wide range of operating conditions and can predict the internal electrochemical state of the cell (as is useful for aging prediction). However, these models are often expressed as coupled *partial-differential equations (PDEs)*, which yield slow simulations that can have robustness and convergence issues.

Volume I in this series studied both types of models in depth. Further, it showed how to compute a reduced-order PBM of similar complexity to an ECM that combines the advantages of both approaches.

Until now, we have explored modeling and simulation of lithium-ion battery *cells* only. We have not considered simulations of battery *packs*, which comprise many cells wired together in series and parallel combinations. As preparation for learning how to manage a battery pack, it is important then to understand how battery packs behave, as distinct from how battery cells behave on their own. Since the battery pack responds to the load to which it is connected, this also requires understanding how the battery-pack load operates.

This chapter develops methods to simulate battery packs and an EV load. Before we explore these topics, we first review the ECM and PBM. The majority of this book will use the ECM as its basis; however, we will require some concepts from the PBM in Chap. 7 when we consider optimal power-limits estimation.

2.2 *Modeling approach 1: Empirical*

The input/output (current/voltage) behaviors of a lithium-ion cell can appear very simple and are often well approximated by an equivalent circuit. That is, an electronic circuit—comprising a voltage source, resistors, capacitors and so forth—is used as an analog to predict the behavior of a physical lithium-ion cell. While the cell does not itself contain these electronic components internally, its voltage response to an input-current stimulus is similar to one that would be produced by the circuit model for the same input current. As most control-systems engineers have more familiarity with circuit design than they do with electrochemistry, equivalent-circuit type models are used extensively (almost exclusively) as the basis for real-time control algorithms in commercial battery packs.

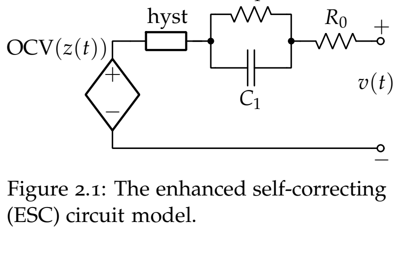


Figure 2.1: The enhanced self-correcting (ESC) circuit model.

Fig. 2.1 shows, as an example, the so-called *enhanced self-correcting (ESC)* equivalent-circuit cell model. In the example circuit, the cell's SOC-dependent open-circuit voltage is drawn as a dependent voltage

11:56:42.

source, “hyst” is a nonlinear hysteresis element, and R_0 is the cell’s ohmic resistance. A single resistor–capacitor pair is drawn to model diffusion voltages; however, additional resistor–capacitor parallel circuits could be added to improve model fidelity. This model structure was originally termed *enhanced* because it contains some description of hysteresis voltages to differentiate it from earlier models that did not. It is called *self-correcting* because the voltage converges to the correct value on both rest and constant-current input events. Even if the transient behavior of the model is imprecise, the model can be relied upon to yield reasonable voltage predictions after some time has elapsed.

In the model, the constants R_0, R_1, C_1, and so forth are termed *parameters*. The circuit elements described by these parameters are analogs to physical properties of diffusion processes when considered collectively but do not individually describe something physical. Therefore, we cannot measure values for the constants using laboratory test methods that isolate a particular physical property. Instead, when the model is being created, values of R_0, C_1, and R_1 are adjusted using an optimization procedure to make model predictions agree as well as possible to measured cell-test data. This process is known as *system identification*. The optimized parameter values are typically a function of state of charge and temperature.

The equations forming the ESC model comprise the following terms:[1]

[1] The derivation of these equations is covered in detail in Vol. I.

State of charge. State of charge at discrete-time index k is denoted as z_k. It evolves over time according to

$$z_{k+1} = z_k - \eta_k i_k \Delta t / Q, \tag{2.1}$$

where η_k is the unitless cell *coulombic efficiency* at time k, i_k is the input current at time k, Δt is the *sample period*, and Q is the cell *total capacity*. SOC is unitless, so if i_k is measured in amperes and Δt is measured in seconds, then Q must be expressed in ampere-seconds. Or, if Δt is measured in hours, then Q must have units of ampere-hours.

Diffusion-resistor current. Current through the resistor R_1 in the resistor–capacitor network at discrete-time index k is denoted as $i_{R_1,k}$ and evolves as

$$i_{R_1,k+1} = \exp\left(\frac{-\Delta t}{R_1 C_1}\right) i_{R_1,k} + \left(1 - \exp\left(\frac{-\Delta t}{R_1 C_1}\right)\right) i_k. \tag{2.2}$$

This term models the slow time constants of diffusion processes occurring within the cell.

Hysteresis voltage. A simple hysteresis model proposes that hysteresis

h_k at discrete-time index k evolves according to

$$h_{k+1} = \exp\left(-\left|\frac{\eta_k i_k \gamma \Delta t}{Q}\right|\right) h_k + \left(\exp\left(-\left|\frac{\eta_k i_k \gamma \Delta t}{Q}\right|\right) - 1\right) \operatorname{sgn}(i_k). \tag{2.3}$$

In this equation, the unitless constant γ adjusts how quickly the hysteresis state changes with a change in cell SOC and $\operatorname{sgn}(\cdot)$ is 1 if its input is positive, -1 if its input is negative, and 0 otherwise. The sample period and total capacity must have the same compatible units as in Eq. (2.1). Note that Eq. (2.3) does not capture observed hysteresis behavior as well as one might like but it is the best simple model of which we are aware.

We can somewhat extend this model (which describes slow changes in hysteresis as cell state of charge changes) by adding a term that describes an instantaneous change in hysteresis voltage when the sign of current changes.[2] To do so, we must define a state that contains memory of the previous sign of nonzero current

[2] This term is not always incorporated in the models in this book, but it is generally simple to add if it is important to the particular cell being modeled.

$$s_k = \begin{cases} \operatorname{sgn}(i_k), & |i_k| > 0; \\ s_{k-1}, & \text{otherwise.} \end{cases}$$

The model is extended easily to contain more than a single parallel resistor–capacitor pair. We do so by defining a vector valued extension to Eq. (2.2) as[3]

[3] Boldfaced lowercase characters are used in this book to denote quantities that may be vectors and boldfaced uppercase characters are used to denote quantities that may be matrices.

$$\boldsymbol{i}_{R,k+1} = \underbrace{\begin{bmatrix} \exp\left(\frac{-\Delta t}{R_1 C_1}\right) & 0 & \cdots \\ 0 & \exp\left(\frac{-\Delta t}{R_2 C_2}\right) & \\ \vdots & & \ddots \end{bmatrix}}_{\boldsymbol{A}_{RC}} \boldsymbol{i}_{R,k} + \underbrace{\begin{bmatrix} \left(1 - \exp\left(\frac{-\Delta t}{R_1 C_1}\right)\right) \\ \left(1 - \exp\left(\frac{-\Delta t}{R_2 C_2}\right)\right) \\ \vdots \end{bmatrix}}_{\boldsymbol{B}_{RC}} i_k.$$

Then, if we also define $A_{H_k} = \exp\left(-\left|\frac{\eta_k i_k \gamma \Delta t}{Q}\right|\right)$, we can combine all of the above relationships into a matrix–vector relationship:

$$\begin{bmatrix} z_{k+1} \\ \boldsymbol{i}_{R,k+1} \\ h_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \boldsymbol{A}_{RC} & 0 \\ 0 & 0 & A_{H_k} \end{bmatrix}}_{\boldsymbol{A}(i_k)} \begin{bmatrix} z_k \\ \boldsymbol{i}_{R,k} \\ h_k \end{bmatrix} + \underbrace{\begin{bmatrix} -\frac{\eta_k \Delta t}{Q} i_k \\ \boldsymbol{B}_{RC} i_k \\ (A_{H_k} - 1)\operatorname{sgn}(i_k) \end{bmatrix}}_{\mathbf{fn}(i_k)}.$$

We can condense notation even further if we define

$$\boldsymbol{x}_k = \begin{bmatrix} z_k \\ \boldsymbol{i}_{R_k} \\ h_k \end{bmatrix}.$$

Then, we have

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}(i_k)\boldsymbol{x}_k + \mathbf{fn}(i_k). \tag{2.4}$$

This is the ESC *state equation*, and describes all dynamic effects.

The ESC *output equation* computes voltage v_k at discrete-time index k as

$$v_k = \mathrm{OCV}(z_k) + Mh_k + M_0 s_k - \sum_i R_i i_{R_i,k} - R_0 i_k, \qquad (2.5)$$

where $\mathrm{OCV}(z_k)$ is the OCV as a function of SOC, M is the maximum absolute analog hysteresis voltage at this temperature, M_0 is the instantaneous hysteresis voltage, and R_0 is the pure ohmic resistance of the cell.

We can define $\boldsymbol{C} = \begin{bmatrix} 0, & -R_1, & -R_2, & \ldots & M \end{bmatrix}$ and $D = -R_0$ to arrive at

$$v_k = \mathrm{OCV}(z_k) + M_0 s_k + \boldsymbol{C}\boldsymbol{x}_k + Di_k.$$

So, we conclude that the ESC model looks similar to—but not identical to—a linear state-space system of the form

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k i_k \\ \boldsymbol{y}_k &= \boldsymbol{C}_k \boldsymbol{x}_k + \boldsymbol{D}_k i_k. \end{aligned}$$

The dependencies of $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ on i_k and the dependence of voltage on the OCV and instantaneous hysteresis terms make the equations nonlinear. Still, this nonlinear state-space form is conducive to applying control-systems concepts, which we do in the following chapters.

Fig. 2.2 illustrates the process for determining parameter values for an ESC cell model.[4] Two different laboratory tests are executed. The *OCV test* first fully charges a cell to arrive at a known initial SOC of 100 % before the main test begins. It then slowly discharges and then slowly recharges the cell. Roughly speaking, the OCV is the average of the discharge and charge voltages at every SOC.

[4] This description is a very abbreviated summary of the entire process described in detail in Vol. I.

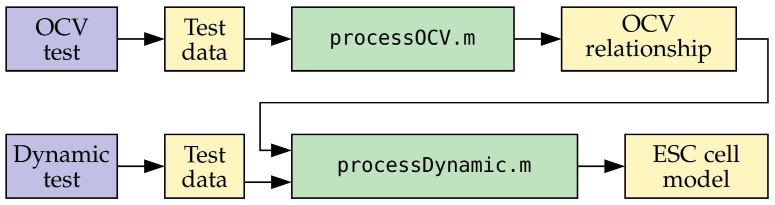


Figure 2.2: Procedure for determining parameters of an ESC cell model.

The *dynamic test* exercises the cell using a profile of current versus time that is similar to what would be expected in an application. The unknown parameters of the model are adapted to make the model predictions match the measured voltage as closely as possible for this input stimulus. In Vol. I of this series, MATLAB® codes were provided to accomplish the OCV processing (`processOCV.m`) and the dynamic parameter fitting (`processDynamic.m`).[5] You can download

[5] MATLAB is a registered trademark of The MathWorks. From now on, this product will be referred to simply as MATLAB.

them from `http://mocha-java.uccs.edu/BMS1/CH2/ESCtoolbox.zip`.

Fig. 2.3 illustrates the process for simulating a cell's response to an input-current stimulus. The model parameters are loaded and the initial state values (z_0, h_0, and $\boldsymbol{i}_{R,0}$) are set. Then, `simCell.m` iteratively evaluates the model equations Eqs. (2.4) and (2.5) for each value in the input-current vector, and produces a vector of predicted cell voltages.

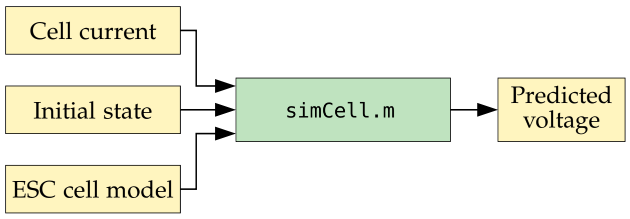


Figure 2.3: Procedure for simulating an ESC cell model.

2.3 *Modeling approach 2: Physics-based*

A fundamentally different approach can be taken to model battery cells by deriving equations—starting from first-principles physics relationships—that describe the electrochemical processes that occur inside the cell.[6] This type of model describes all internal cell processes using coupled partial-differential equations.

With reference to Fig. 2.4, which illustrates the primary components of a cell, the following internal electrochemical variables are of interest:[7]

- The concentration of lithium $c_s(x,r,t)$ in the solid active materials that comprise each electrode at spatial location x across the cell and at radial location r within a particle, and especially at the solid–electrolyte boundary at the surface of the solid: $c_{s,e}(x,t)$.
- The electric potential in the solid, $\phi_s(x,t)$.
- The flux density (normalized rate of lithium movement) between solid and electrolyte, $j(x,t)$. The electrode variables $c_{s,e}(x,t)$,

[6] This was the primary focus of Vol. I. We review the main results here, but refer the reader back to that volume for complete understanding of where these equations originate and what all the terms really mean.

[7] The equations presented in this section are for a continuum-scale model, which describes behavior in the neighborhood of a spatial location using a volume average of an underlying microscale model. We assume a simplified one-dimensional geometry where all electrode solid particles are spherical.

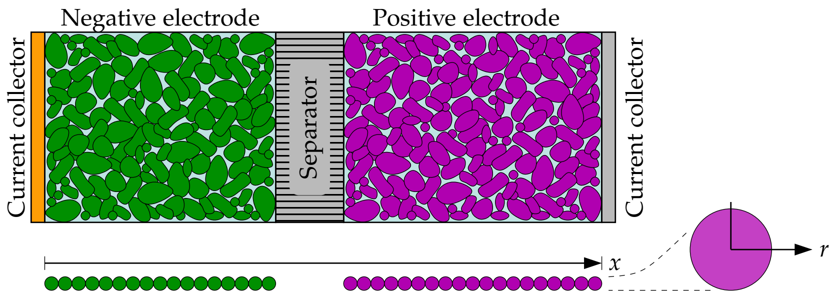


Figure 2.4: Illustration of cross-section of a cell, used by continuum physics-based models.
(Reproduced from Fig. 1 of Stetzel, K., Aldrich, L., Trimboli, M.S., and Plett, G., "Electrochemical State and Internal Variables Estimation using a Reduced-Order Physics-Based Model of a Lithium-Ion Cell and an Extended Kalman Filter," *Journal of Power Sources*, 278, 2015, pp. 490–505.)

11:56:42.

$\phi_s(x,t)$, and $j(x,t)$ are defined only at x locations corresponding to spatial locations in either the negative or positive electrodes. They are not defined in the separator region as no solid active materials are present in that region.

- The concentration of lithium $c_e(x,t)$ in the electrolyte at spatial location x, and
- The electric potential in the electrolyte, $\phi_e(x,t)$. The cell-scale variables $c_e(x,t)$ and $\phi_e(x,t)$ are defined at all x locations spanning the cell since the electrolyte permeates the electrode and separator regions.

The time-varying values for these five electrochemical variables can be found by solving four coupled continuum-scale partial-differential equations and one algebraic equation (along with associated boundary conditions). To determine the concentration of lithium in the spherically symmetric solid electrode particles, we solve the model radial-diffusion equation

$$\frac{\partial}{\partial t} c_s = \frac{D_s}{r^2} \frac{\partial}{\partial r}\left(r^2 \frac{\partial c_s}{\partial r}\right),$$

where D_s is the solid diffusivity. Charge balance in the solid active-material particles is modeled with aid of the solid-potential variable ϕ_s as

$$\nabla \cdot (\sigma_{\text{eff}} \nabla \phi_s) = a_s F j,$$

where σ_{eff} is the effective electronic conductivity of the electrode materials in the porous electrode and a_s is the specific interfacial surface area of the electrode active materials.[8] Mass balance of lithium in the electrolyte is captured by

$$\frac{\partial(\varepsilon_e c_e)}{\partial t} = \nabla \cdot (D_{e,\text{eff}} \nabla c_e) + a_s(1 - t_+^0) j,$$

where ε_e is the porosity of the electrode, $D_{e,\text{eff}}$ is the effective diffusivity of the electrolyte and t_+^0 is the transference number of the positively charged lithium ion with respect to the solvent in the electrolyte. Charge balance in the electrolyte is modeled by

$$\nabla \cdot (\kappa_{\text{eff}} \nabla \phi_e + \kappa_{D,\text{eff}} \nabla \ln c_e) + a_s F j = 0,$$

where κ_{eff} is the ionic conductivity of the electrolyte and $\kappa_{D,\text{eff}}$ is the ionic conductivity multiplied by a conversion factor. Finally, the rate of reaction at the surface of the particles is captured by the Butler–Volmer equation

$$j = k_0 {c_e}^{1-\alpha} (c_{s,\max} - c_{s,e})^{1-\alpha} {c_{s,e}}^{\alpha} \left\{ \exp\left(\frac{(1-\alpha)F}{RT}\eta\right) - \exp\left(-\frac{\alpha F}{RT}\eta\right) \right\},$$

[8] The $\nabla \cdot$ (argument) notation denotes a *divergence* operation and describes the net rate of flux of the quantity in its argument away from the point in space being modeled. Again, this notation is described in detail in Vol. I.

11:56:42.

where $\eta = (\phi_s - \phi_e) - U_{\text{ocp}}(c_{s,e}) - FR_{\text{film}}j$, k_0 is the reaction-rate constant and α is the asymmetric charge-transfer coefficient. Cell voltage is captured by the difference in solid potentials at the two current-collector locations.

These equations can be solved via simulation at any point in time and space to determine values for the five coupled electrochemical variables. However, these simulations require considerable processor resources and experience shows that some partial-differential-equation simulators can be fragile—they can have a difficult time converging to consistent solutions, especially when there are large abrupt changes in the cell input current.

Therefore, we desire to find methods that can create discrete-time *reduced-order models (ROMs)* based on the PDEs such that the ROMs are computationally simple and robust. In Volume I we did so via making two basic assumptions:[9]

1. We assumed linear behavior: We linearized nonlinear equations using Taylor series;
2. We assumed that the reaction current $j(x,t)$ was decoupled from (not a function of) the electrolyte concentration $c_e(x,t)$.

This allowed us to create Laplace-domain transfer functions from the linearized equations. We then used a method called the *discrete-time realization algorithm (DRA)* to create a discrete-time state-space model. The process for creating the ROM is illustrated in Fig. 2.5. The PDEs are linearized and mathematical manipulations are done to create transfer functions for the variables of interest in terms of the cell's physical constants. Values for these constants are found via laboratory tests and system-identification procedures and are then merged into the transfer functions. The DRA operates on the transfer functions to produce a linearized discrete-time state-space model. The steps of the DRA involve 1) converting the frequency responses into impulse responses; 2) converting the impulse responses into step responses; 3) converting the step responses into discrete-time unit-pulse responses; and 4) converting these unit-pulse responses into the final reduced-order state-space model form. The entire process is automatic and involves only common signal-processing and linear-algebra steps. No nonlinear optimization or curve fitting steps are needed.

[9] In simulations, we have found that the second assumption is quite good. The first assumption is less good, but nonlinear corrections help improve linear predictions.

The ROM must be created only once. However, as it is defined only in the neighborhood of a single temperature and state of charge linearization setpoint, multiple ROMs must be blended together in operation to represent the cell's behavior accurately over the entire functional range of the cell. The process for using the ROM in an application is shown in Fig. 2.6. As with the ECM, the fundamental

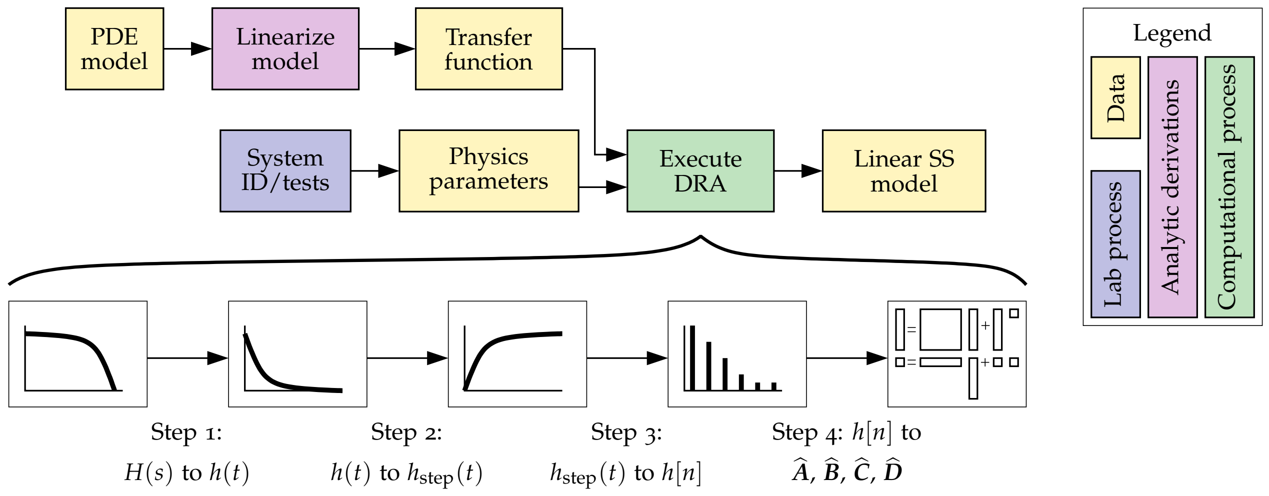

PDE model
Linearize model
Transfer function
System ID/tests
Physics parameters
Execute DRA
Linear SS model
Legend
Data
Lab process
Analytic derivations
Computational process
Step 1: $H(s)$ to $h(t)$
Step 2: $h(t)$ to $h_{\text{step}}(t)$
Step 3: $h_{\text{step}}(t)$ to $h[n]$
Step 4: $h[n]$ to $\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}, \widehat{\boldsymbol{C}}, \widehat{\boldsymbol{D}}$

Figure 2.5: Process for creating a reduced-order physics-based model.

inputs are the cell input current, an initial state, and the model structures. However, simulating the physics-based ROM has some additional steps. Four ROMs surrounding the present temperature/SOC operational point are blended to make a time-varying state-space model. One iteration of this model is simulated each simulated time step, producing linearized predictions of internal electrochemical cell variables. Nonlinear corrections are applied to these variables, producing high-fidelity predictions of cell electrochemical variables and cell voltage. Then, the process repeats until all values in the input-current profile have been considered.

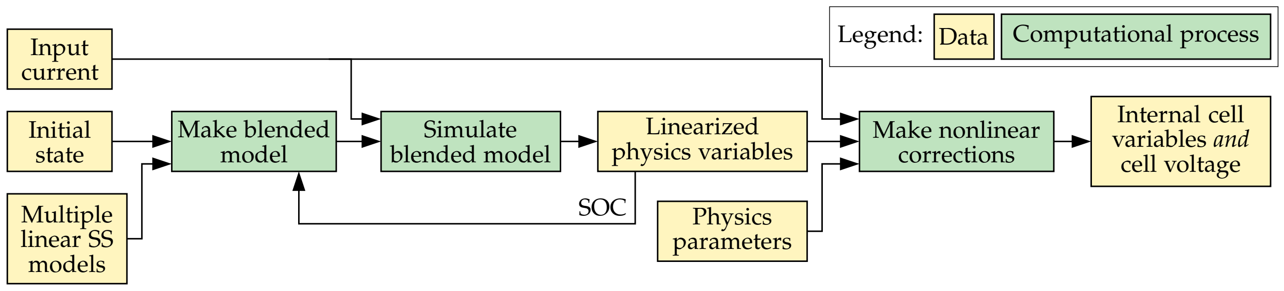

Legend: Data Computational process
Input current
Initial state
Multiple linear SS models
Make blended model
Simulate blended model
Linearized physics variables
SOC
Physics parameters
Make nonlinear corrections
Internal cell variables *and* cell voltage

Figure 2.6: Process for simulating a physics-based ROM.

Battery management and control algorithms require either ECMs or PBMs of the cells that they manage. The majority of this book describes methods for battery management based on ECMs. However, Chap. 7 introduces some potential benefits to using PBMs. A planned third volume in this series will focus on battery management and control using physics-based models and the benefits that can accrue from doing so.

2.4 *Simulating an EV*

When designing battery packs, it is important to be able to simulate the operation of the pack before proceeding too far down the design path. This helps to ensure that the pack will be able to meet all performance requirements before a large investment in electronic and mechanical design has been made.

Simulating a battery pack requires that we also have very good models of the battery-pack load. And, while large battery packs may be used in a variety of applications, here we consider the requirement for energy storage in xEVs as an illustrative example.

To predict battery demand from an xEV, we must simulate the vehicle over a number of real-world operating scenarios to determine profiles of power or current versus time required from the pack. HEV simulations are extremely complex because the internal-combustion engine, multispeed transmission, and hybrid blending control algorithms need to be modeled accurately for useful simulation results.[10] However, simulations of pure EVs (and PHEVs or E-REVs operating in electric-only charge-depletion mode) are fairly straightforward because there is no need to simulate an internal-combustion engine. Further, a single fixed-ratio transmission is used, and there is no blending of power sources. We consider this kind of simulation here as an example of how to determine a battery-pack load profile.

[10] The National Renewable Energy Laboratories (NREL) has developed a MATLAB/Simulink HEV simulator called ADVISOR, which is now open source. It can be downloaded from `http://sourceforge.net/projects/adv-vehicle-sim/files/ADVISOR/`.

To SIMULATE AN EV, two things are needed: an accurate description of the vehicle itself and knowledge of the task the vehicle is required to accomplish. The *vehicle description* includes all factors needed for a dynamic simulation, including battery-cell characteristics, battery-module characteristics, battery-pack characteristics, motor and inverter (motor-driving power electronics) characteristics, drivetrain characteristics, and so forth. The *vehicle task* is characterized by a *drive-cycle profile*: a function or table specifying desired vehicle speed versus time.

Fig. 2.7 illustrates four commonly encountered drive-cycle profiles. The *urban dynamometer driving schedule (UDDS)* profile is representative of city driving. It was originally defined by the Environmental Protection Agency (EPA) as a dynamometer test for internal-combustion vehicles to determine city-driving fuel efficiency but has been adopted by the EV community as a measure of electrical efficiency for urban driving as well. Likewise, the *highway fuel-efficiency test (HW-FET)* was originally intended to measure highway fuel efficiency, but is now also used to gauge EV performance in highway driving. Many other driving profiles exist—one needs only a GPS receiver that can record speed versus time in order to create custom

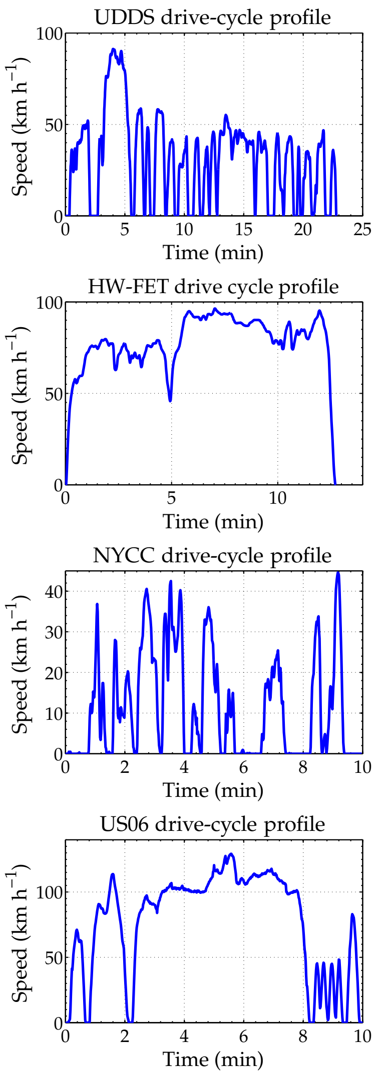


Figure 2.7: Commonly encountered drive-cycle profiles.

drive-cycle profiles tailored for a specific application. For example, the *New York City cycle (NYCC)* is representative of bus or taxicab driving in New York City; the US06 drive-cycle profile is representative of mixed highway and urban driving (it was recorded by the National Renewable Energy Laboratories on U.S. highway 6 near Golden, CO).[11]

[11] These drive-cycle profiles are included in the archive that can be downloaded from the URL given in footnote 14.

All drive-cycle profiles are simulated using the same basic equations by the process illustrated in Fig. 2.8 and as described in detail in the following pages. The approach is to compute, on a sample-by-sample basis, desired accelerations to match the desired speed profile, therefore desired road forces, therefore desired motor torques and power values. These desired torques and power values are restricted by the specifications of the motor chosen for the vehicle, and thus achievable torques and power values may be of lesser magnitude. Achievable torques are computed, and from them achieved road force, acceleration, and speed (this will not always match the desired speed).

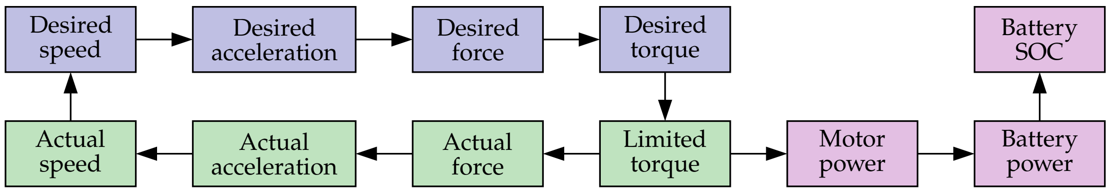


Figure 2.8: Simulation strategy.

The battery pack may be cosimulated with the motor and drivetrain to impose its own constraints on achievable performance. Here, however, we simplify analysis by assuming that the battery is always able to deliver the demanded power to the load. Battery power is computed based on motor power and battery average SOC is updated. Over the course of the driving profile, the battery average SOC is depleted a certain amount, and the projected range of the vehicle is extrapolated from this data.

2.5 *Equations for vehicle dynamics*

Our first step in creating an EV simulator is to derive equations of EV dynamics in more detail.[12] We begin by calculating the desired vehicle acceleration as

[12] Vehicle modeling equations are from T. Gillespie, "Fundamentals of Vehicle Dynamics," Society of Automotive Engineers Inc, 1992.

$$\begin{aligned}&\textit{desired acceleration}\,[\mathrm{m\,s^{-2}}] = \\ &\qquad (\textit{desired speed}\,[\mathrm{m\,s^{-1}}] - \textit{actual speed}\,[\mathrm{m\,s^{-1}}])\,/\,(\Delta t\,[\mathrm{s}]), \qquad (2.6)\end{aligned}$$

where Δt is the sampling period of the drive cycle. We then compute the net desired acceleration force that the motor must produce at the

11:56:42.

road surface:

$$\begin{aligned}&\textit{desired acceleration force}\,[\mathrm{N}] = \\ &\qquad \textit{equivalent mass}\,[\mathrm{kg}] \times \textit{desired acceleration}\,[\mathrm{m\,s^{-2}}].\end{aligned} \tag{2.7}$$

In this equation, the *equivalent mass* combines the maximum vehicle mass and the translational equivalent mass of the rotating inertias

$$\begin{aligned}&\textit{equivalent mass}\,[\mathrm{kg}] = \\ &\qquad \textit{maximum vehicle mass}\,[\mathrm{kg}] + \textit{rotating equivalent mass}\,[\mathrm{kg}]\end{aligned} \tag{2.8}$$

where

$$\begin{aligned}&\textit{rotating equivalent mass}\,[\mathrm{kg}] = \\ &\qquad ((\textit{motor inertia}\,[\mathrm{kg\,m^2}] + \textit{gearbox inertia}\,[\mathrm{kg\,m^2}]) \times N^2 \\ &\qquad + \textit{number of wheels} \times \textit{wheel inertia}\,[\mathrm{kg\,m^2}])/(\textit{wheel radius}\,[\mathrm{m}])^2,\end{aligned} \tag{2.9}$$

and where we assume a fixed gearbox ratio

$$N = (\textit{motor RPM})/(\textit{wheel RPM})$$

and measure the gearbox inertia at the motor side of the gearbox, not at the output side. Also, for most accurate results, the wheel radius is assumed to be that of the rolling wheel; that is, it takes into account the slight flattening of the tire because of load.

In addition to the force delivered by the motor, various other forces act on the vehicle. Aerodynamic drag is modeled as

$$\begin{aligned}&\textit{aerodynamic force}\,[\mathrm{N}] = \frac{1}{2}(\textit{air density } \rho\,[\mathrm{kg\,m^{-3}}])\times \\ &\qquad (\textit{aerodynamic frontal area}\,[\mathrm{m^2}])\times \\ &\qquad (\textit{drag coefficient } C_d\,[\text{unitless}]) \times (\textit{prior actual speed}\,[\mathrm{m\,s^{-1}}])^2,\end{aligned} \tag{2.10}$$

where the aerodynamic frontal area is the equivalent area experienced by the drag forces. Rolling drag is modeled as a kinetic friction force:

$$\begin{aligned}&\textit{rolling force}\,[\mathrm{N}] = (\textit{rolling friction coefficient } C_r\,[\text{unitless}]) \times \\ &\qquad (\textit{max. vehicle mass}\,[\mathrm{kg}]) \times (\textit{acceleration of gravity}\,[9.81\,\mathrm{m\,s^{-2}}]).\end{aligned} \tag{2.11}$$

Note that the rolling force is computed to be zero if the prior actual speed is 0. The force of gravity either assists or impedes progress due to a road grade and is modeled as

$$\begin{aligned}&\textit{grade force}\,[\mathrm{N}] = (\textit{maximum vehicle mass}\,[\mathrm{kg}]) \times \\ &\qquad (\textit{acceleration of gravity}\,[9.81\,\mathrm{m\,s^{-2}}]) \times \sin(\textit{grade angle}\,[\mathrm{rad}])\end{aligned} \tag{2.12}$$

11:56:42.

where *grade angle* is the present (or average) slope of the road (a positive angle represents an incline; a negative angle represents a decline). Finally, we model the possibility of a constant friction force (e.g., from sticking brake pads) as

$$\textit{brake drag}\,[\mathrm{N}] = \textit{constant road force input by user}.$$

We can now compute the demand torque at the motor to achieve the demanded acceleration for this time step:

$$\begin{aligned}\textit{demanded motor torque}\,[\mathrm{N\,m}] = (&\textit{desired acceleration force}\,[\mathrm{N}] + \\ &\textit{aerodynamic force}\,[\mathrm{N}] + \textit{rolling force}\,[\mathrm{N}] + \textit{grade force}\,[\mathrm{N}] + \\ &\textit{brake drag}\,[\mathrm{N}]) \times \textit{wheel radius}\,[\mathrm{m}] / N[\text{unitless}]. \end{aligned} \tag{2.13}$$

This completes the feed-forward steps illustrated in the blue boxes in Fig. 2.8.

It may not be possible for the vehicle's motor to produce this demanded torque. So, we must consider the motor's limitations in the simulation. Here, we will assume that the vehicle employs an AC induction motor, which has torque versus speed operational characteristics that are well modeled using the type of relationship plotted in Fig. 2.9.

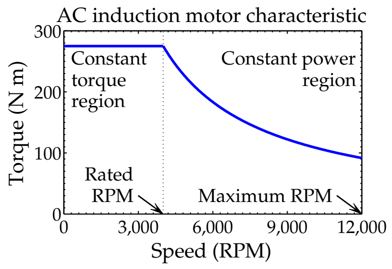


Figure 2.9: Operating characteristics of an example AC induction motor.

At low speeds, the motor can deliver any torque up to a constant maximum rated torque. At speeds higher than the motor's *rated RPM*, the torque is limited but the motor is able to deliver up to a constant maximum power (where power is the product of torque and speed). In all cases, the motor speed must be kept below the *maximum RPM*.

In the simulation, then, demanded torque must be compared to the maximum available motor torque based on the prior motor RPM. When positive torque (acceleration) is demanded, if prior actual motor speed is less than rated motor speed then the maximum available torque is the rated maximum available torque; otherwise, the maximum available torque is computed as (rated maximum available torque [N m]) $\times$ (rated motor speed [RPM]) / (prior actual motor speed [RPM]).

When negative torque (deceleration) is demanded, torque demand is split between friction brakes (assumed infinitely powerful) and the motor. Energy recovered from the motor replaces energy depleted from the battery (less inefficiency losses) in a *regeneration* event.[13] The maximum motor torque available for regeneration (in an unsigned sense) is calculated as a "regen fraction" times the rated maximum available torque. Finally, the limited torque at the motor is the lesser of the demanded motor torque and the maximum available torque (in an unsigned sense).

[13] *Regeneration* or *regen* is the process of recovering energy by using the motor as part of the braking system in place of friction brakes. This recovered energy is stored in the battery pack. The times when regen events will occur, and the magnitudes of the events are not predictable, which is a challenge for BMS design that is handled by imposing charge-power limits on the load, as discussed in Chap. 6.

Now that the motor torque limits have been established, we can compute the actual acceleration force that is available, the actual acceleration, and the actual speed. Starting with the actual acceleration force,

$$\begin{aligned}
&\textit{actual acceleration force}\,[\mathrm{N}] = \textit{limited torque at motor}\,[\mathrm{N\,m}] \times \\
&\quad N\,[\text{unitless}]/\textit{wheel radius}\,[\mathrm{m}] - \textit{aerodynamic force}\,[\mathrm{N}] - \\
&\quad \textit{rolling force}\,[\mathrm{N}] - \textit{grade force}\,[\mathrm{N}] - \textit{brake drag}\,[\mathrm{N}] \qquad (2.14)\\
&\textit{actual acceleration}\,[\mathrm{m\,s^{-2}}] = \\
&\quad \textit{actual acceleration force}\,[\mathrm{N}]/\textit{equivalent mass}\,[\mathrm{kg}]. \qquad (2.15)
\end{aligned}$$

The actual acceleration as just calculated may cause the motor to spin at a higher angular velocity than its maximum RPM. Therefore, we cannot compute actual speed as simply as follows:

$$\begin{aligned}
&\textit{actual speed}\,[\mathrm{m\,s^{-1}}] = \textit{prior actual speed}\,[\mathrm{m\,s^{-1}}]+ \\
&\qquad\qquad \textit{actual acceleration}\,[\mathrm{m\,s^{-2}}] \times \Delta t\,[\mathrm{s}].
\end{aligned}$$

Instead, we must compute a motor RPM first, then limit that RPM, and then compute the actual vehicle speed:

$$\begin{aligned}
&\textit{test speed}\,[\mathrm{m\,s^{-1}}] = \textit{prior actual speed}\,[\mathrm{m\,s^{-1}}]+ \\
&\qquad\qquad \textit{actual acceleration}\,[\mathrm{m\,s^{-2}}] \times \Delta t\,[\mathrm{s}] \qquad (2.16)\\
&\textit{motor speed}\,[\mathrm{RPM}] = \textit{test speed}\,[\mathrm{m\,s^{-1}}] \times N[\text{unitless}]\times \\
&\qquad\qquad 60\,[\mathrm{s\,min^{-1}}]/(2\pi \times \textit{wheel radius}\,[\mathrm{m}]). \qquad (2.17)
\end{aligned}$$

Motor speed is limited by the maximum rated motor speed to make a limited motor speed. Finally, actual vehicle speed is computed as

$$\begin{aligned}
&\textit{actual speed}\,[\mathrm{m\,s^{-1}}] = \textit{limited motor speed}\,[\mathrm{RPM}] \times 2\pi \times \\
&\quad \textit{wheel radius}\,[\mathrm{m}]/(60\,[\mathrm{s\,min^{-1}}] \times N[\text{unitless}]). \qquad (2.18)
\end{aligned}$$

The full circuit from desired to actual speed in Fig. 2.8—including both the blue and green boxes—has now been described.

THE EQUATIONS DEVELOPED SO FAR show whether the vehicle is able to develop the accelerations required to follow a specific drive-cycle profile. They assume that sufficient battery power is available to supply the motor demand at every time step.

To determine vehicle range based on battery capacity, the battery pack must be cosimulated with the vehicle. At the beginning of the simulation, the battery is initialized to be fully charged. As the simulation executes, the vehicle draws energy from the battery pack, reducing the remaining energy level. At some point, vehicle demand will cause the SOC or voltage of a cell in the battery pack to drop below some minimum design threshold. At that point, we consider the vehicle to have driven its maximum range.

To cosimulate the battery pack, we first compute instantaneous power required by the motor:

$$\begin{aligned}&\textit{motor power}\,[\mathrm{kW}] = 2\pi\,[\mathrm{rad\,revolution}^{-1}]\times\\&\quad\left(\frac{\textit{motor speed}\,[\mathrm{RPM}] + \textit{previous motor speed}\,[\mathrm{RPM}]}{2}\right)\times\\&\quad\textit{limited torque at motor}\,[\mathrm{N\,m}]\,/\,(60\,[\mathrm{s\,min}^{-1}]\times 1{,}000\,[\mathrm{W\,kW}^{-1}]).\end{aligned}\tag{2.19}$$

If motor power is positive, then battery power is calculated as

$$\begin{aligned}&\textit{battery power}\,[\mathrm{kW}] = \textit{overhead power}\,[\mathrm{kW}] +\\&\quad\textit{motor power}\,[\mathrm{kW}]\,/\,\textit{drivetrain efficiency}\,[\mathrm{unitless}],\end{aligned}\tag{2.20}$$

where the overhead power is the constant power drain from other vehicle systems, such as air conditioners, "infotainment" systems and so forth. If motor power is negative (during a regeneration event), battery power is calculated as

$$\begin{aligned}&\textit{battery power}\,[\mathrm{kW}] = \textit{overhead power}\,[\mathrm{kW}] +\\&\quad\textit{motor power}\,[\mathrm{kW}]\times\textit{drivetrain efficiency}\,[\mathrm{unitless}].\end{aligned}\tag{2.21}$$

We know how to make very precise battery-cell models. However, for now we will simply assume a constant battery voltage (a poor assumption, but sufficient for first-order approximate results). Then,

$$\begin{aligned}&\textit{battery current}\,[\mathrm{A}] = \textit{battery power}\,[\mathrm{kW}]\times 1{,}000\,[\mathrm{W\,kW}^{-1}]\,/\\&\qquad\textit{battery nominal voltage}\,[\mathrm{V}].\end{aligned}\tag{2.22}$$

Battery SOC is updated as

$$\begin{aligned}&\textit{battery SOC}\,[\%] = \textit{prior battery SOC}\,[\%] - \textit{battery current}\,[\mathrm{A}]\times\\&\quad\Delta t\,[\mathrm{s}]\,/\,(3{,}600\,[\mathrm{s\,hr}^{-1}]\times\textit{battery capacity}\,[\mathrm{A\,hr}])\times 100\,[\%].\end{aligned}\tag{2.23}$$

Finally, driving range is extrapolated from the drive-cycle calculations:

$$\begin{aligned}&\textit{range}\,[\mathrm{miles}] = \textit{total distance of simulated drive cycle}\,[\mathrm{miles}]\times\\&\quad(\textit{max. rated battery SOC}\,[\%] - \textit{min. rated battery SOC}\,[\%])\,/\\&\quad(\textit{SOC at beginning}\,[\%] - \textit{SOC at end of drive cycle}\,[\%]).\end{aligned}$$

This equation assumes that the vehicle repeats the same drive-cycle profile until the battery-pack energy is exhausted. If this repetition is

11:56:42.

not realistic, then a longer drive-cycle profile should be used as input, where this profile is sufficiently long that it is not repeated before the battery-pack energy is exhausted, and where the speed versus time data tabulated in the profile are more representative of the expected vehicle operational environment.

By running simulations with distinct drive-cycle profiles as input, we can learn what to expect for vehicle range under these different scenarios. As previously mentioned, custom profiles can be created by driving a specific route and logging GPS data, so this simulation mechanism is quite general.

2.6 *EV simulation code*

While an EV simulator could be written in any computer language, for purpose of illustration we now present some MATLAB code that implements these model equations. This code is divided into two functions: `setupSimVehicle.m` is an example of how to set up the parameter values that describe the vehicle and the drive cycle, and `simVehicle.m` is the code that executes the equations we've just derived to accomplish the simulation. These two functions are described in the following subsections.[14]

[14] This code is available at `http://mocha-java.uccs.edu/BMS2/CH2/EVsim.zip`.

2.6.1 `setupSimVehicle.m`

The `setupSimVehicle.m` function defines the parameters of the battery cell, module, and pack, and the parameters of the motor, wheels, and drivetrain. The parameter values are stored in structures, combined to make a vehicle description, and are later used when simulating the vehicle. The values in the example code are a rough description of a first-generation General Motors' Chevy Volt operating in pure-electric charge-depletion mode, based on public information (and speculation) prior to vehicle release. They are fairly reasonable parameter values, but are certainly not exact, nor are they verified for this vehicle.

The `setupSimVehicle.m` function is somewhat lengthy, so we do not present it as a monolithic block. Rather, we describe its operation section by section. To reproduce the code in MATLAB, one would need to reassemble the sections into a single function and save it to the file `setupSimVehicle.m`.

The file begins by defining the function and a list of data files containing drive-cycle profiles to be simulated:

```
function results = setupSimVehicle
  files = {'nycc.txt','udds.txt','us06.txt','hwfet.txt'};
```

Next, a data structure named `cell` is initialized via the `setupCell` function (to be defined later):

```
% set up the Chevy Volt:
% set up cell: capacity [Ah], mass [g], (vmax, vnom, vmin) [V]
cell = setupCell(15,450,4.2,3.8,3.0);
```

In this example, we are defining a battery cell that has capacity 15 Ah, mass 450 g, maximum operational voltage 4.2 V, nominal voltage 3.8 V, and minimum operational voltage 3.0 V.

Next, we define a battery module in terms of this cell:

```
% set up module: number of cells in parallel, number of cells in
% series, overhead of module by fraction of total cells' mass
module = setupModule(3,8,0.08,cell);
```

In this case, the module comprises three cells wired in parallel and eight cells wired in series. The module's mass is equal to the mass of the cells plus 8 % overhead for packaging, electronics, and so forth. The module uses cells described by the structure `cell`, which has already been defined.

Next, we define the battery pack in terms of this module:

```
% set up pack: number of modules in series, overhead of pack by
% fraction of total modules' mass, (full SOC, empty SOC) [%],
% efficiency for the pack
pack = setupPack(12,0.1,75,25,0.96,module);
```

In this example, the battery pack comprises 12 modules wired in series. The pack mass is equal to the mass of all modules combined plus 10 % overhead for packaging, electronics, cooling, and so forth. The pack design allows cell SOC to range from 75 % when "fully charged" to 25 % when "empty" (i.e., when switching from charge-depletion to charge-sustaining mode). The energy efficiency of the pack is 96 % and the pack comprises modules defined by the `module` variable.

With the battery pack now defined, we turn our attention to other aspects of the drivetrain. We next set up a structure that contains the motor parameters:

```
% set up motor: max torque "Lmax" [Nm], (RPMrated, RPMmax) [RPM],
% efficiency, inertia [kg/m2]
motor = setupMotor(275,4000,12000,0.95,0.2);
```

This motor provides maximum torque 275 Nm, has rated RPM of 4,000 RPM and has maximum RPM of 12,000 RPM. The motor's efficiency is 95 % and it has inertia of 0.2 $\text{kg}\,\text{m}^{-2}$.

The data structure containing information about the wheels is set up next:

```
% set up wheel: radius [m], inertia [kg/m2], rollCoef
wheel = setupWheel(0.35,8,0.0111);
```

This wheel has rolling (flattened) radius of 0.35 m, inertia of 8 $\text{kg}\,\text{m}^{-2}$ and rolling coefficient of $C_r = 0.0111$.

The drivetrain combines information from the pack, motor, and wheels:

```
% set up drivetrain: inverter efficiency, fractional regen torque
% limit, gear ratio, gear inertia [kg/m2], gear efficiency for this
% pack, motor, and wheel
drivetrain = setupDrivetrain(0.94,0.9,12,0.05,0.97,pack,motor,wheel);
```

In addition, we specify the inverter efficiency to be 94 %, the regeneration torque fraction to be 90 %, the fixed gear ratio to be 12:1, the gear inertia to be 0.05 kg m^{-2}, and the gearing efficiency to be 97 %.

Finally, we set up a data structure defining the entire vehicle:

```
% set up vehicle: # wheels, roadForce [N], Cd, frontal area [m2],
% mass [kg], payload [kg], overhead power [W] for this drivetrain
vehicle = setupVehicle(4,0,0.22,1.84,1425,75,200,drivetrain);
```

The vehicle has four wheels, a constant road force (brake drag) of 0 N, a drag coefficient $C_d = 0.22$, an aerodynamic frontal cross-sectional area of 1.84 m^2, a mass of 1425 kg, a payload capacity of 75 kg, and draws a constant overhead power of 200 W.

With the vehicle set up, we simulate the four drive-cycle profiles and display some results:

```
fprintf('\n\nStarting sims...\n');
for theCycle = 1:length(files),
  cycle = dlmread(files{theCycle},'\t',2,0);
  results = simVehicle(vehicle,cycle,0.3);
  range = (vehicle.drivetrain.pack.socFull - ...
           vehicle.drivetrain.pack.socEmpty) /...
          (vehicle.drivetrain.pack.socFull - ...
           results.batterySOC(end)) * ...
          results.distance(end);
  fprintf('Cycle = %s, range = %6.1f [km]\n',files{theCycle},range);
end
```

First, each drive-cycle profile is read from a tab-delimited text file (first column is time in seconds; second column is desired speed in mph). Then, the `simVehicle.m` function is invoked to simulate the vehicle (with a constant 0.3 % road grade). Finally, the vehicle range is extrapolated from the simulation.

The file `setupSimVehicle.m` is not yet complete. It contains some functions that compose the data structures for cell, module, pack, and so forth, that are nested *inside* the main `setupSimVehicle` function. We look at these briefly before proceeding to discuss `simVehicle.m`. First, `setupCell` creates a cell data structure from its input parameters:

```
function cell = setupCell(capacity,mass,vmax,vnom,vmin)
  cell.capacity = capacity; % ampere hours
  cell.mass = mass; % grams
  cell.vmax = vmax; % volts
  cell.vnom = vnom; % volts
  cell.vmin = vmin; % volts
  cell.energy = vnom * capacity; % Watt-hours
  cell.specificEnergy = 1000 * cell.capacity * cell.vnom/ ...
```

```
                          cell.mass; % Wh/kg
  end
```

Most of these operations simply copy the input values into the corresponding structure fields. However, two auxiliary calculations compute the cell nominal energy and specific energy.

Similarly, `setupModule` is a nested function that defines the fields of the module structure:

```
  function module = setupModule(numParallel,numSeries,overhead,cell)
    module.numParallel = numParallel;
    module.numSeries = numSeries;
    module.overhead = overhead;
    module.cell = cell;
    module.numCells = numParallel * numSeries;
    module.capacity = numParallel * cell.capacity;
    module.mass = module.numCells * cell.mass/(1-overhead)/1000; % kg
    module.energy = module.numCells * cell.energy/1000; % kWh
    module.specificEnergy = 1000 * module.energy / module.mass; % Wh/kg
 end
```

Besides directly assigning the input variables to their respective fields in the output structure, the function computes the total number of cells per module, the total module capacity in ampere-hours, the module mass, nominal energy, and specific energy.

The `pack`, `motor`, `wheel`, `drivetrain`, and `vehicle` structures are defined in a similar fashion using the following functions nested inside the `setupSimVehicle` function. The `setupPack` function computes auxiliary variables for total number of cells in the pack, pack mass, total energy capacity, specific energy, and battery-pack voltage ranges:

```
 function pack = setupPack(numSeries,overhead,socFull,socEmpty,...
                           efficiency,module)
    pack.numSeries = numSeries;
    pack.overhead = overhead;
    pack.module = module;
    pack.socFull = socFull;
    pack.socEmpty = socEmpty; % unitless
    pack.efficiency = efficiency; % unitless, captures I*I*R losses
    pack.numCells = module.numCells * numSeries;
    pack.mass = module.mass * numSeries * 1/(1 - overhead); % kg
    pack.energy = module.energy * numSeries; % kWh
    pack.specificEnergy = 1000 * pack.energy / pack.mass; % Wh/kg
    pack.vmax = numSeries*module.numSeries*module.cell.vmax;
    pack.vnom = numSeries*module.numSeries*module.cell.vnom;
    pack.vmin = numSeries*module.numSeries*module.cell.vmin;
  end
```

The `setupMotor` function sets the output fields in the `motor` structure, and computes motor maximum power. The `setupWheel` function simply sets output fields in the `wheel` structure:

```
  function motor = setupMotor(Lmax,RPMrated,RPMmax,efficiency,inertia)
    motor.Lmax = Lmax; % N-m
    motor.RPMrated = RPMrated;
    motor.RPMmax = RPMmax;
    motor.efficiency = efficiency;
    motor.inertia = inertia; %kg-m2
    motor.maxPower = 2*pi*Lmax*RPMrated/60000; % kW
  end

 function wheel = setupWheel(radius,inertia,rollCoef)
    wheel.radius = radius; % m
    wheel.inertia = inertia; % km-m2
    wheel.rollCoef = rollCoef;
  end
```

The `setupDrivetrain` function stores data in the fields of the `drivetrain` structure, including a computation of drivetrain efficiency:

```
  function drivetrain = setupDrivetrain(inverterEfficiency,regenTorque,...
        gearRatio,gearInertia,gearEfficiency,pack,motor,wheel)
    drivetrain.inverterEfficiency = inverterEfficiency;
    % regen torque is fraction of braking power that is used to charge
    % battery; e.g., value of 0.9 means 90% of braking power contributes
    % to charging battery; 10% lost to heat in friction brakes
    drivetrain.regenTorque = regenTorque;
    drivetrain.pack = pack;
    drivetrain.motor = motor;
    drivetrain.wheel = wheel;
    drivetrain.gearRatio = gearRatio;
    drivetrain.gearInertia = gearInertia; % kg-m2, measured on motor side
    drivetrain.gearEfficiency = gearEfficiency;
    drivetrain.efficiency = pack.efficiency * inverterEfficiency * ...
                            motor.efficiency * gearEfficiency;
  end
```

Finally, `setupVehicle` creates the `vehicle` structure containing all drivetrain data including some computed mass variables—including rotating mass via Eq. (2.9) and vehicle equivalent mass via Eq. (2.8)—and the vehicle maximum speed:

```
  function vehicle = setupVehicle(wheels,roadForce,Cd,A,mass,payload,...
                                  overheadPwr,drivetrain)
    vehicle.drivetrain = drivetrain;
    vehicle.wheels = wheels; % number of them
    vehicle.roadForce = roadForce; % N
    vehicle.Cd = Cd; % drag coeff
    vehicle.A = A; % frontal area, m2
    vehicle.mass = mass; % kg
    vehicle.payload = payload; % kg
    vehicle.overheadPwr = overheadPwr; % W
    vehicle.curbMass = mass + drivetrain.pack.mass;
    vehicle.maxMass = vehicle.curbMass + payload;
    vehicle.rotMass = ((drivetrain.motor.inertia + ...
                        drivetrain.gearInertia) * ...
                        drivetrain.gearRatio^2 + ...
                        drivetrain.wheel.inertia*wheels)/...
                        drivetrain.wheel.radius^2;
    vehicle.equivMass = vehicle.maxMass + vehicle.rotMass;
    vehicle.maxSpeed  = 2 * pi * drivetrain.wheel.radius * ...
                        drivetrain.motor.RPMmax * 60 / ...
                        (1000 * drivetrain.gearRatio); % km/h
  end
end
```

Note that the final "end" closes out the setupSimVehicle function. All of the other functions are defined as nested inside this overall function.

2.6.2 simVehicle.m

The equations that simulate the vehicle are implemented in the simVehicle.m function, which simulates a drive-cycle profile and returns a structure called results that contains comprehensive simulation outcomes.

The function begins with

```
% results = simVehicle(vehicle,cycle,grade)
%   - simulate a vehicle defined by "vehicle", perhaps created using
%     setupSimVehicle.m
%   - cycle is Nx2, where first column is time in seconds and second
%     column is desired speed in miles per hour
%   - grade is road grade in percent - either a constant grade for all
%     time, or a different grade value for every point in time
function results = simVehicle(vehicle,cycle,grade)
  rho = 1.225; % air density, kg/m3
  results.vehicle = vehicle;
  results.cycle = cycle; % time in s, desired speed in miles/hour
  results.time = cycle(:,1); % s
  results.desSpeedKPH = cycle(:,2) * 1.609344; % convert to km/h
  results.desSpeed = min(vehicle.maxSpeed,...
                         results.desSpeedKPH*1000/3600); % m/s
```

First, air density (at sea level) is defined, and then the function input variables are copied into the results data structure.

If a scalar is provided as the road grade, that scalar is replicated for all time samples in the simulation. Otherwise, the vector input grade defines a different road grade for every time sample in the drive cycle. In either case, the grade is converted from percent into a radian angle:

```
  if isscalar(grade),
    results.grade = repmat(atan(grade/100),size(results.time)); % rad
  else
    results.grade = atan(grade/100); % rad
  end
```

Next, fields in the results data structure are preallocated and set to 0. This is done to reserve memory before the simulation begins, so that MATLAB is not constantly growing vectors inside of the simulation loop (which is a very slow operation):

```
  % pre-allocate storage for fields of "results" data structure
  zeroInit = zeros(size(results.desSpeed));
  results.desAccel         = zeroInit; % m/s2
  results.desAccelForce    = zeroInit; % N
  results.aeroForce        = zeroInit; % N
  results.rollGradeForce   = zeroInit; % N
  results.demandTorque     = zeroInit; % N-m
  results.maxTorque        = zeroInit; % N-m
  results.limitRegen       = zeroInit; % N-m
```

```
results.limitTorque      = zeroInit; % N-m
results.motorTorque      = zeroInit; % N-m
results.demandPower      = zeroInit; % kW
results.limitPower       = zeroInit; % kW
results.batteryDemand    = zeroInit; % kW
results.current          = zeroInit; % A
results.batterySOC       = zeroInit; % 0..100
results.actualAccelForce = zeroInit; % N
results.actualAccel      = zeroInit; % m/s2
results.motorSpeed       = zeroInit; % RPM
results.actualSpeed      = zeroInit; % m/s
results.actualSpeedKPH   = zeroInit; % km/h
results.distance         = zeroInit; % km
```

Now, the main simulation loop begins. After initializing some variables defining the "previous" state before the simulation began, the equations of Sect. 2.5 are evaluated, and results are stored for future analysis:

```
prevSpeed = 0; prevMotorSpeed = 0;   prevDistance = 0;
prevSOC = vehicle.drivetrain.pack.socFull;
prevTime = 2*results.time(1) - results.time(2); % for even sampling
for k = 1:length(results.desSpeed),
  results.desAccel(k) = (results.desSpeed(k) - prevSpeed)/ ...
                        (results.time(k) - prevTime);
  results.desAccelForce(k) = vehicle.equivMass * results.desAccel(k);
  results.aeroForce(k) = 0.5 * rho * vehicle.Cd * vehicle.A * ...
                         prevSpeed^2;
  results.rollGradeForce(k) = vehicle.maxMass * 9.81 * ...
                              sin(results.grade(k));
  if abs(prevSpeed) > 0,
    results.rollGradeForce(k) = results.rollGradeForce(k) + ...
      vehicle.drivetrain.wheel.rollCoef * vehicle.maxMass * 9.81;
  end
  results.demandTorque(k) = (results.desAccelForce(k) + ...
                             results.aeroForce(k) + ...
                             results.rollGradeForce(k) + ...
                             vehicle.roadForce) * ...
                             vehicle.drivetrain.wheel.radius / ...
                             vehicle.drivetrain.gearRatio;
```

In this section, Eq. (2.6) is evaluated to compute the desired acceleration, Eq. (2.7) computes the desired acceleration force, Eq. (2.10) computes the aerodynamic force, Eq. (2.12) computes the grade force, Eq. (2.11) computes the rolling force, and Eq. (2.13) computes the total demanded torque.

The total demanded torque is now limited based on motor characteristics:

```
  if prevMotorSpeed < vehicle.drivetrain.motor.RPMrated,
    results.maxTorque(k) = vehicle.drivetrain.motor.Lmax;
  else
    results.maxTorque(k) = vehicle.drivetrain.motor.Lmax * ...
        vehicle.drivetrain.motor.RPMrated / prevMotorSpeed;
  end
  results.limitRegen(k) = min(results.maxTorque(k),...
                              vehicle.drivetrain.regenTorque * ...
                              vehicle.drivetrain.motor.Lmax);
  results.limitTorque(k) = min(results.demandTorque(k),...
                               results.maxTorque(k));
  if results.limitTorque(k) > 0,
    results.motorTorque(k) = results.limitTorque(k);
  else
    results.motorTorque(k) = max(-results.limitRegen(k),...
                                 results.limitTorque(k));
  end
```

We can now compute the actual acceleration force via Eq. (2.14), the actual acceleration via Eq. (2.15), the actual motor speed via Eqs. (2.16) and (2.17), and the actual vehicle speed via Eq. (2.18):

```
    results.actualAccelForce(k) = results.limitTorque(k) * ...
                                 vehicle.drivetrain.gearRatio / ...
                                 vehicle.drivetrain.wheel.radius - ...
                                 results.aeroForce(k) - ...
                                 results.rollGradeForce(k) - ...
                                 vehicle.roadForce;
    results.actualAccel(k) = results.actualAccelForce(k) / ...
                            vehicle.equivMass;
    results.motorSpeed(k) = min(vehicle.drivetrain.motor.RPMmax,...
                           vehicle.drivetrain.gearRatio * ...
                           (prevSpeed + results.actualAccel(k) * ...
                           (results.time(k) - prevTime)) * 60 / ...
                           (2*pi*vehicle.drivetrain.wheel.radius));
    results.actualSpeed(k) = results.motorSpeed(k) * ...
                            2*pi*vehicle.drivetrain.wheel.radius / ...
                            (60 * vehicle.drivetrain.gearRatio);
    results.actualSpeedKPH(k) = results.actualSpeed(k) * 3600/1000;
    results.distance(k) = prevDistance + ...
                         results.actualSpeedKPH(k)/3600;
```

The total distance traveled to date via this drive cycle is also computed.

Next, the power demand by the motor is computed via Eq. (2.19). This is converted to a battery demanded power via either Eq. (2.20) or Eq. (2.21) for the discharge and regen cases, respectively. This is converted to a battery current via Eq. (2.22) and then to a change in battery (average) SOC via Eq. (2.23). Finally, some intermediate results are saved to variables for the next iteration of the program loop:

```
    if results.limitTorque(k) > 0,
      results.motorPower(k) = results.limitTorque(k);
    else
      results.motorPower(k) = max(results.limitTorque(k),...
                                  -results.limitRegen(k));
    end % "motorPower" == motor torque until next line...
    results.motorPower(k) = results.motorPower(k) * 2*pi / 60000 * ...
                           (prevMotorSpeed + results.motorSpeed(k))/2;
    results.limitPower(k) = max(-vehicle.drivetrain.motor.maxPower,...
                           min(vehicle.drivetrain.motor.maxPower,...
                               results.motorPower(k)));
    results.batteryDemand(k) = vehicle.overheadPwr/1000;
    if results.limitPower(k) > 0,
      results.batteryDemand(k) = results.batteryDemand(k) + ...
            results.limitPower(k)/vehicle.drivetrain.efficiency;
    else
      results.batteryDemand(k) = results.batteryDemand(k) + ...
            results.limitPower(k)*vehicle.drivetrain.efficiency;
    end
    results.current(k) = results.batteryDemand(k)*1000/...
                         vehicle.drivetrain.pack.vnom;
    results.batterySOC(k) = prevSOC - results.current(k) * ...
                           (results.time(k) - prevTime) / ...
                      (36*vehicle.drivetrain.pack.module.capacity);

    prevTime = results.time(k);
    prevSpeed = results.actualSpeed(k);
    prevMotorSpeed = results.motorSpeed(k);
    prevSOC = results.batterySOC(k);
    prevDistance = results.distance(k);
  end
```

When the simulation has looped over all input data, the `results` data structure is returned to the calling program for further analysis and display.

2.7 *EV simulation results*

There are many opportunities for visualization and analysis of vehicle performance with this simulator by examining the data stored in the `results` structure. Some of these are displayed in Fig. 2.10.

The top frame of the figure shows battery power demand versus time for the US06 drive-cycle profile, using the default parameters in the simulation code. The second frame shows the battery current demand. Note that this is computed from the power demand by assuming a constant nominal voltage: a more accurate result could be achieved if a more accurate cell model were used, such as the ESC model reviewed earlier in this chapter. The third frame shows a histogram of demanded motor power. This kind of information is useful when selecting a motor for an EV application and for determining cooling requirements for the motor. Finally, the bottom frame shows a scatter plot of torque versus speed operational points visited by the US06 drive-cycle profile. We note that only one point falls on the motor limited boundary (denoted as black lines), so we conclude that this motor is well sized for this application.

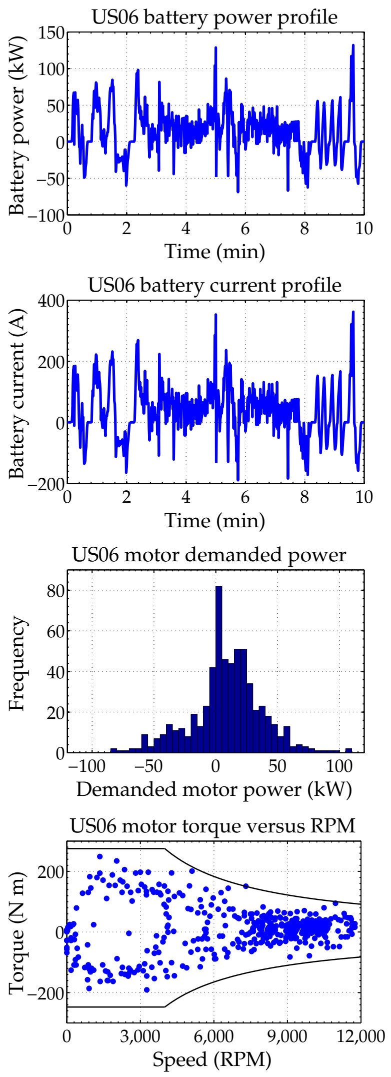


Figure 2.10: Some example results computed by the EV simulator.

This is just one example of how to simulate a battery-pack load. This is exactly, in fact, how the profiles of current versus time were computed for the UDDS drive-cycle profile used throughout Volume I to exercise battery cells in the laboratory and to demonstrate simulation results of model performance.

For any other battery application that is different from the EV application, we would either need to measure actual battery demand versus time, or we would need to develop a simulator for that application that could be used to predict battery power demand versus time to be able to simulate the performance of the battery pack under those conditions.

2.8 Simulating constant power and voltage

We have now completed our discussion of simulating an example battery-pack load and return to the task of simulating the battery pack itself. This will ultimately involve simulating multiple interconnected battery cells. We already know how to simulate a single cell's voltage response to an input-current profile as the battery-cell models that we have developed all have model input equal to applied cell electrical current. However, the EV simulator example we have just examined has demonstrated that demanded battery-cell *power* could also be considered a cell-model input. Other applications require that battery-cell *voltage* be the input and battery-cell current be the output.

Fortunately, both of these new scenarios can be accommodated easily by the ESC cell model, with no structural changes and minimal additional computation. We see how this is done in the next two subsections.

2.8.1 Constant-power simulation

To see how to simulate a cell where the input is equal to power and the output is equal to voltage, we first restate the ESC model of Eqs. (2.4) and (2.5):

$$\boldsymbol{x}_k = \boldsymbol{A}(i_{k-1})\boldsymbol{x}_{k-1} + \mathbf{fn}(i_{k-1}) \tag{2.24}$$
$$v_k = \underbrace{\text{OCV}(\boldsymbol{x}_k) + \text{hysteresis}(\boldsymbol{x}_k) - \text{diffusion}(\boldsymbol{x}_k)}_{\text{not a function of instantaneous current}} - R_0 i_k. \tag{2.25}$$

In Eq. (2.24), we make the important observation that the present state $\boldsymbol{x}_k$ is not a function of the present input current i_k; rather, it is a function of all prior input-current values i_{k-1}, i_{k-2}, and so forth. Therefore, all terms in the voltage equation Eq. (2.25) that are based on the present state are similarly independent of the present instantaneous value of input current. The only term in the voltage calculation that depends on present current is the ohmic voltage drop $-R_0 i_k$.

Therefore, we can say that at any point in time, cell voltage comprises a "fixed" part, which does not depend on the present cell current, and a "variable" part, $-R_0 i_k$, which does depend on present cell current. Of course, the fixed part changes from time sample to time sample as the state changes, but it is indeed fixed for any given time sample in that it is not a function of the present input current. To simplify our notation, we then define the fixed part to be $v_{f,k}$ and so we have $v_k = v_{f,k} - R_0 i_k$.

Our desire is for the cell to deliver a specified power at its terminals. We now have the ability to compute the input current required to deliver this desired power. Since power is equal to the product of terminal voltage and cell current, we can write

$$\begin{aligned} p_k &= v_k i_k \\ &= (v_{f,k} - R_0 i_k) i_k. \end{aligned}$$

We can rearrange these terms to obtain

$$R_0 i_k^2 - v_{f,k} i_k + p_k = 0.$$

This quadratic equation can be solved at every sample time to determine the value of cell input current that will meet the power demand

$$i_k = \frac{v_{f,k} \pm \sqrt{v_{f,k}^2 - 4R_0 p_k}}{2R_0}.$$

Which sign on the radical must we use? Both choices cause the cell-model simulation to produce the required power, but one turns out to require a negative cell voltage to do so. Because cell voltage must remain positive, the sign of the radical must be negative.

In summary, to achieve a profile of desired power versus time we may use the existing ESC cell model with input equal to applied cell current. At every time step we simply compute the required cell input current to be

$$i_k = \frac{v_{f,k} - \sqrt{v_{f,k}^2 - 4R_0 p_k}}{2R_0}.$$

We then update the state and output equations of the model (this is needed to update $v_{f,k}$ for the next time step) and repeat for every sample of desired power versus time.

2.8.2 Constant-voltage simulation

With this background, constant-voltage simulation is very straightforward. Every time step, we must determine i_k such that

$$v_k = v_{f,k} - R_0 i_k.$$

Therefore,

$$i_k = \frac{v_{f,k} - v_k}{R_0}.$$

2.8.3 Example requiring constant current, power, and voltage

Now that we know how to simulate a cell when demanded power is the input, we might revisit the EV simulator to incorporate a more accurate cell model to obtain higher-fidelity results. (We leave this as an exercise for the reader.) Another application for which constant-power and constant-voltage requirements naturally arise is during the process of charging a cell.

There are two common approaches to charging a battery cell:

1. *Constant-current/constant-voltage (CC/CV).* A constant charge current is applied to the cell until the cell reaches some predetermined maximum voltage. The cell's terminal voltage is then held at this maximum value until charging current becomes negligibly small.

2. *Constant-power/constant-voltage (CP/CV).* A constant charge power is applied to the cell until the cell reaches some predetermined maximum voltage. The cell's terminal voltage is then held at this maximum value until charging power (or current) becomes negligibly small.

The CC/CV mode is often used in laboratory tests of cells, but CP/CV is more commonly used in xEV chargers.[15] Both approaches may be simulated given what we now know about constant-current, constant-power, and constant-voltage simulations.

We will examine their behavior in an example where we consider charging a battery cell from 50 % to 100 % SOC (corresponding to a rest voltage of 4.15 V). The code begins by loading an ESC cell model from a data file, extracting operating parameters, and initializing storage for results.[16]

```
% --------------------------------------------------------------------
% simCharge: Simulate CC/CV and CP/CV charging of a battery cell
% --------------------------------------------------------------------
clear all; close all; clc;
load cellModel; % creates var. "model" with cell parameter values

% Get ESC model parameters
maxtime = 3001; T = 25; % Simulation run time, temperature
q  = getParamESC('QParam',T,model);   % total capacity
rc = exp(-1./abs(getParamESC('RCParam',T,model))); % time constant
r  = (getParamESC('RParam',T,model)); % diffusion resistance
m  = getParamESC('MParam',T,model);   % hysteresis max voltage
g  = getParamESC('GParam',T,model);   % hysteresis rate gamma
r0 = getParamESC('R0Param',T,model);  % series resistance
maxV = 4.15; % maximum cell voltage of 4.15 V

% Initialize simulation storage and state variables
storez = zeros([maxtime 1]);  % create storage for SOC
storev = zeros([maxtime 1]);  % create storage for voltage
storei = zeros([maxtime 1]);  % create storage for current
storep = zeros([maxtime 1]);  % create storage for power
z  = 0.5; irc = 0; h  = -1;   % initialize to 50% SOC, resting
```

We first simulate a CC/CV charge process. The code loops through `maxtime` iterations, updating the state and voltage predictions every time step. Cell current is computed every iteration to achieve a con-

[15] The constant-voltage segment is often omitted when charging a battery pack (as opposed to charging single cells). Instead, several constant-current or constant-power segments are performed in sequence, with decreasing input power, until the pack has achieved some desired level of charge.

[16] This code segment uses helper function `getParamESC.m` from the ESC model toolbox, available at `http://mocha-java.uccs.edu/BMS1/CH2/ESCtoolbox.zip`.

stant voltage but is limited to the maximum absolute value of 9 A to enforce the CC/CV operation.

```
% First, simulate CC/CV
CC = 9;  % constant absolute current of 9 A in CC/CV charge
for k = 1:maxtime,
  v = OCVfromSOCtemp(z,T,model) + m*h - r*irc; % fixed voltage

  ik = (v - maxV)/r0; % compute test ik to achieve maxV
  ik = max(-CC,ik);   % but limit current to no more than CC in mag.

  z = z - (1/3600)*ik/q;  % Update cell SOC
  irc = rc*irc + (1-rc)*ik; % Update resistor currents
  fac = exp(-abs(g.*ik)./(3600*q));
  h = fac.*h + (fac-1).*sign(ik); % Update hysteresis voltages
  storez(k) = z; % Store SOC for later plotting
  storev(k) = v - ik*r0;
  storei(k) = ik; % store current for later plotting
  storep(k) = ik*storev(k);
end % for k

time = 0:maxtime -1;
figure(1); clf; plot(time,100*storez); hold on
figure(2); clf; plot(time,storev);     hold on
figure(3); clf; plot(time,storei);     hold on
figure(4); clf; plot(time,storep);     hold on
```

Next, we simulate the CP/CV charge process. The code is very similar, except that within every iteration we first compute the voltage that would be achieved by a constant-power time step. If that results in voltage that exceeds the design voltage, we instead compute current for a constant-voltage time step.

```
% Now, simulate CP/CV
z  = 0.5; irc = 0; h  = -1; % initialize to 50% SOC, resting
CP = 35; % constant absolute power limit of 30 W in CP/CV charge
for k = 1:maxtime,
  v = OCVfromSOCtemp(z,T,model) + m*h - r*irc; % fixed voltage

  % try CP first
  ik = (v - sqrt(v^2 - 4*r0*(-CP)))/(2*r0);
  if v - ik*r0 > maxV, % too much!
    ik = (v - maxV)/r0; % do CV instead
  end

  z = z - (1/3600)*ik/q;  % Update cell SOC
  irc = rc*irc + (1-rc)*ik; % Update resistor currents
  fac = exp(-abs(g.*ik)./(3600*q));
  h = fac.*h + (fac-1).*sign(ik); % Update hysteresis voltages
  storez(k) = z; % Store SOC for later plotting
  storev(k) = v - ik*r0;
  storei(k) = ik; % store current for later plotting
  storep(k) = ik*storev(k);
end % for k

figure(1); plot(time,100*storez,'g--')
figure(2); plot(time,storev,'g--')
figure(3); plot(time,storei,'g--')
figure(4); plot(time,storep,'g--')
```

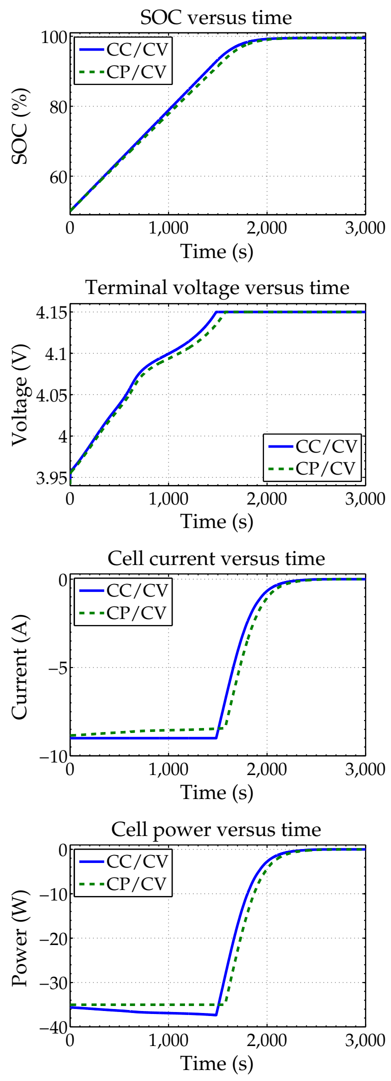


Figure 2.11: CC/CV and CP/CV charging.

Results of this simulation are shown in Fig. 2.11. In the constant-current part of the CC/CV simulation, a 1C charge rate is held constant and cell voltage responds to this input stimulus. Because voltage is increasing and current is kept constant, charge power *increases* in magnitude over this interval. SOC increases linearly. When voltage reaches the maximum value of 4.15 V, the simulation switches to a constant-voltage mode. The voltage is maintained constant at 4.15 V and, in response, the magnitude of the current continually decreases as the diffusion dynamics reach a steady-state value. SOC converges over this interval to 100 % and charge-power magnitude decreases.

In the constant-power part of the CP/CV simulation, a 35 W magnitude charge power is chosen to be similar to the CC/CV scenario. This applied power is held constant and cell voltage again responds to this stimulus. Since voltage is increasing but power is constant, charge current *decreases* in magnitude over this interval. The change in SOC is slower than the linear change in the CC/CV scenario for this chosen power level. When voltage reaches the maximum value of 4.15 V, the simulation switches to a constant-voltage mode. The voltage is maintained constant at 4.15 V and, in response, the magnitude of the charge power (and hence also charge current) continually decreases as the diffusion dynamics reach a steady-state value. SOC converges over this interval to 100 %.

2.9 *Simulating battery packs*

At last we arrive at the stated purpose of this chapter—to discuss how to simulate battery packs! To simulate *battery-cell* behavior, the ESC model voltage equation Eq. (2.5) is evaluated and the model state equation Eq. (2.4) is updated once per sample interval. Simulating *battery packs* must somehow involve simulating multiple interconnected battery cells.

2.9.1 *Series-connected cells*

Simulating a battery pack comprising cells that are connected only in series is straightforward as all cells must experience the same input current (by Kirchhoff's current law). If all cells have identical initial state and parameter values, then all cells have exactly the same state and voltage for all time, so we need to simulate only one cell (the others will have identical state and voltage).

In general, however, cells have different initial state and parameter values. Thus to simulate a general series-connected set of cells we must simulate all cells' individual dynamics by keeping independent state and model information for every cell, updating each cell's state and voltage computation once per sample interval.

When computing the battery-pack voltage, we then simply sum the individual cell voltages. Additionally, we can include an intercell *interconnect resistance* term when computing pack voltage, to arrive at the overall equation

$$v_{\text{pack},k} = \left(\sum_{j=1}^{N_s} v_{j,k} \right) - N_s R_{\text{interconnect}} i_k, \tag{2.26}$$

where $v_{j,k}$ is the voltage of cell j at time instant k. The interconnect resistance models external resistance of cell tabs and connections between cells in a battery pack.

2.9.2 *Packs comprising parallel-connected cell modules*

Series-connected packs are common for low-energy high-power applications such as HEV. High-energy applications, however, often have cells and even entire subpacks that are connected in parallel. We previewed this concept in Chap. 1, where we considered modules comprising N_p cells connected in parallel (cf. the PCM approach in Fig. 1.3).

If all cells in a PCM are identical in every respect, simulation is straightforward. Only one cell in each PCM must be simulated and its input current is equal to i_k/N_p. Pack voltage is computed using Eq. (2.26).

However, if cells in a PCM are not identical, the pack input current is not divided evenly among all cells in the PCM, and each cell in the PCM must be simulated individually.[17] But how?

[17] This would be the case for situations where capacities and resistances of cells are not the same in each parallel path, such as when cells have different states of health, and especially if there is a failure in one of the paths.

To see how to simulate a battery pack comprising PCMs, we refer to Fig. 2.12. In the figure, each blue rectangle delineates one battery cell. As discussed in conjunction with Eq. (2.25), each cell's voltage can be modeled as having a fixed part that does not depend on the present cell current, and a variable part that does depend on present cell current. Fig. 2.12 symbolizes the fixed part as a voltage source and the variable part as a resistance in each cell. Therefore the voltage source in the figure is not only OCV but also includes present hysteresis and diffusion voltages.

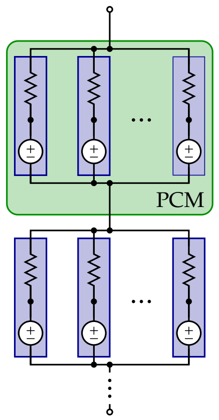


Figure 2.12: Schematic of a battery pack where modules comprise multiple cells wired in parallel.

By Kirchhoff's voltage law, all terminal voltages of cells connected in parallel must be equal; by Kirchhoff's current law, the sum of currents through all cells connected in parallel must equal the total battery-pack current. We define current through cell j of a PCM at time k as $i_{j,k}$, its fixed voltage as $v_{fj,k}$, the PCM overall voltage as v_k, and the resistance of the jth cell as $R_{0,j}$. Then, Ohm's law applied

11:56:42.

across each cell's resistance gives cell current as

$$i_{j,k} = \frac{v_{fj,k} - v_k}{R_{0,j}}, \tag{2.27}$$

which could be computed if we were able to find v_k.

We arrive at the total battery-pack current by summing all parallel cell currents

$$i_k = \sum_{j=1}^{N_p} \frac{v_{fj,k}}{R_{0,j}} - v_k \sum_{j=1}^{N_p} \frac{1}{R_{0,j}}.$$

By rearranging, we can solve for the PCM voltage

$$v_k = \frac{\sum_{j=1}^{N_p} \frac{v_{fj,k}}{R_{0,j}} - i_k}{\sum_{j=1}^{N_p} \frac{1}{R_{0,j}}}. \tag{2.28}$$

To summarize, we first evaluate Eq. (2.28) to determine the PCM terminal voltage. Then, we compute the individual cell currents using Eq. (2.27). Once we have the independent cell currents $i_{j,k}$, we can update the cell-model state associated with each cell. Pack voltage is computed by summing all PCM voltages and interconnect voltage drops in much the same way as is done in Eq. (2.26).

2.9.3 *Packs comprising series-connected cell modules*

When simulating packs comprising series-connected modules, the approach is very similar to how we simulate PCM. Referring again to Fig. 1.3, each module in an SCM comprises multiple cells wired in series. Each cell has a voltage with a fixed and variable part. Using elementary circuit analysis, we can lump together all individual fixed parts in an SCM as a single voltage source that sums the separate fixed voltages, and we can lump together all individual variable parts in the SCM as a single resistance that sums all separate cell equivalent-series resistances. Each SCM can then be modeled as a single high-voltage high-resistance cell. When the SCMs are connected in parallel to create a battery pack, the overall pack has schematic as shown in the lower portion of Fig. 2.13.

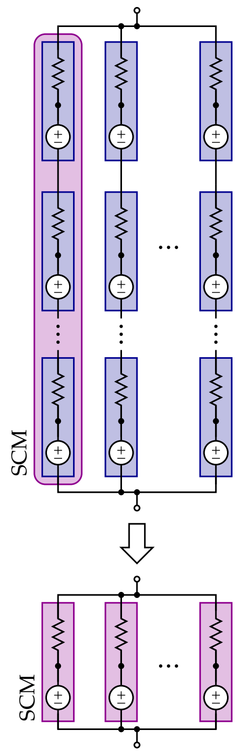


Figure 2.13: Collapsed schematic of a battery pack where modules comprise multiple cells wired in series.

If we denote the total lumped fixed voltage of SCM j by $v_{fj,k}$ and the total lumped resistance by $R_{0,j}$, then our prior analysis allows us to compute overall battery-pack voltage as

$$v_k = \frac{\sum_{j=1}^{N_p} \frac{v_{fj,k}}{R_{0,j}} - i_k}{\sum_{j=1}^{N_p} \frac{1}{R_{0,j}}}. \tag{2.29}$$

The individual SCM currents are then found as

$$i_{j,k} = (v_{fj,k} - v_k)/R_{0,j}. \tag{2.30}$$

11:56:42.

With these currents now known, we can update the state of every cell in the battery pack.

2.10 PCM simulation code

Before concluding this chapter, we present some MATLAB code to simulate PCMs and SCMs, and some simulation results.[18] The PCM-simulation code begins with some comments and definitions. It also loads the ESC cell-model structure `model` from the file `cellModel.mat` to define some default cell parameter values.

[18] This code may be downloaded from `http://mocha-java.uccs.edu/CH2/PCMSCM.zip`.

```
% ------------------------------------------------------------------------
% simPCM: Simulate parallel-connected-module packs (cells are connected in
% parallel to make modules; these modules are connected in series to make
% packs).
%
% The parameters for each cell may be different (e.g., capacity,
% resistance, etc.)
% ------------------------------------------------------------------------
clear all; close all; clc;

% Initialize some pack configuration parameters
load cellModel; % creates var. "model" with ESC cell parameter values
```

Any kind of simulation scenario could be considered. In the examples to be presented in Sect. 2.11, we simulate a battery pack comprising three cells in parallel and three cells in series. The simulation nominally begins with all cells fully rested at 25 % SOC. The simulation then predicts performance over one full hour of pack operation where the pack is repeatedly charged until the cell having highest state of charge reaches a maximum design SOC limit; the pack is then discharged until the cell having lowest SOC reaches a minimum SOC design limit. The pack rests after 2,700 s. The following code segment defines initial variables to this simulation.

```
% Initialize some simulation configuration parameters...
Ns = 3;         % Number of PCMs connected in series to make a pack
Np = 3;         % Number of cells connected in parallel in each PCM
maxtime = 3600; % Simulation run time in simulated seconds
t0 = 2700;      % Pack rests after time t0
storez = zeros([maxtime Ns Np]);  % create storage for SOC
storei = zeros([maxtime Ns Np]);  % create storage for current

% Initialize states for every battery-pack cell ESC cell model
z  = 0.25*ones(Ns,Np);
irc = zeros(Ns,Np);
h  = zeros(Ns,Np);
```

The next code segment loads the default parameter values from the structure `model`. Note that the simulation considers a constant cell temperature of 25 °C and defines a 125 $\mu\Omega$ interconnect resistance per cell tab.

```
% Default initialization for cells within the pack
```

```
q    = getParamESC('QParam',25,model)*ones(Ns,Np);
rc   = exp(-1./abs(getParamESC('RCParam',25,model)))'*ones(Ns,Np);
r    = (getParamESC('RParam',25,model))';
m    = getParamESC('MParam',25,model)*ones(Ns,Np);
g    = getParamESC('GParam',25,model)*ones(Ns,Np);
r0   = getParamESC('R0Param',25,model)*ones(Ns,Np);
rt   = 0.000125; % 125 microOhm interconnect resistance for each cell tab
```

At this point, all cells have identical parameters. The next section introduces different kinds of cell variability so that we can explore how the battery pack responds to them. In the following, the code is configured to overwrite initial SOC for every cell with a random value between 30 % and 70 %, to overwrite total capacity for each cell with a random value between 4.5 Ah and 5.5 Ah, and to overwrite resistance with a random value between 5 mΩ and 25 mΩ. To disable any of these random initializations, simply replace the corresponding "`if true` " statement with "`if false`."

```
% Modified initialization for cell variability
% Set individual random "initial SOC" values
if true, % set to "if true," to execute, or "if false," to skip this code
  z=0.30+0.40*rand([Ns Np]); % rand. init. SOC for ea. cell
end

% Set individual random cell-capacity values
if true, % set to "if true," to execute, or "if false," to skip this code
  q=4.5+rand([Ns Np]);      % random capacity for ea. cell
end

% Set individual random cell-resistance relationships
if true, % set to "if true," to execute, or "if false," to skip this code
  r0 = 0.005+0.020*rand(Ns,Np);
end
r0 = r0 + 2*rt; % add tab resistance to cell resistance
```

One important capability of a battery-pack simulator that would be difficult and possibly unsafe to reproduce in a physical battery pack is the ability to simulate fault scenarios. These capabilities are disabled in the code below, but can be enabled by uncommenting the indicated lines in the following segment. An open-circuit fault can be simulated by setting a cell's `r0` value to `Inf` (infinity), and a short-circuit fault can be simulated by setting a cell's SOC to `NaN` ("not a number"). In the example below, a short-circuit cell has no dynamics beyond a series resistance of value `Rsc`.

```
% Add faults to pack: cells faulted open- and short-circuit
%   To delete a PCM (open-circuit fault), set a resistance to Inf
%   r0(1,1) = Inf; % for example...

%   To delete a cell (short-circuit fault), set its SOC to NaN
%   z(1,2) = NaN; % for example, delete cell 2 in PCM 1
Rsc = 0.0025; % Resistance value to use for cell whose SOC < 0%
```

The cells are now configured. We get ready to simulate by computing the approximate 10C rate of the battery pack at which the cells will be cycled in the simulation (this is not a very realistic rate, but it does give some qualitative results in a short simulation).

```
% Get ready to simulate... first compute pack capacity in Ah
totalCap = min(sum(q,2)); % pack capacity = minimum module capacity
I = 10*totalCap; % cycle at 10C... not realistic, faster simulation
```

The simulation now begins. All the battery-pack variables are stored in individual matrices having N_s rows and N_p columns. The $N_s \times N_p$ matrix v containing the fixed part of each cell's voltage is computed as open-circuit voltage plus hysteresis minus the diffusion voltages. The N_s PCM terminal voltages V are then computed using Eq. (2.28) and the $N_s \times N_p$ PCM individual cell currents ik are computed using Eq. (2.27). Every cell's state is then updated (note that if any state of charge somehow becomes less than zero, that cell is converted to a short-circuit fault). If the minimum state of charge is less than 5 %, then the simulation switches from discharge to charge; if the maximum SOC is greater than 95 %, then the simulation switches from charge to discharge. If the 2,700 s activity period has been exceeded, the cell rests. Finally, all cell states of charge and input current are stored for later analysis and visualization.

```
% Okay... now to simulate pack performance using ESC cell model.
for k = 1:maxtime,
  v = OCVfromSOCtemp(z,25,model); % get OCV for each cell: Ns * Np matrix
  v = v + m.*h - r.*irc; % add in hysteresis and diffusion voltages
  r0(isnan(z)) = Rsc; % short-circuit fault has "short-circuit" resistance
  V = (sum(v./r0,2) - I)./sum(1./r0,2);
  ik = (v-repmat(V,1,Np))./r0;
  z = z - (1/3600)*ik./q;  % Update each cell SOC
  z(z<0) = NaN; % set over-discharged cells to short-circuit fault
  irc = rc.*irc + (1-rc).*ik; % Update diffusion currents
  Ah = exp(-abs(g.*ik)./(3600*q));
  h = Ah.*h + (Ah-1).*sign(ik); % Update hysteresis voltages
  if min(z(:)) < 0.05, I = -abs(I); end % stop discharging
  if max(z(:)) > 0.95, I =  abs(I); end % stop charging
  if k>t0, I = 0; end % rest
  storez(k,:,:) = z; % Store SOC for later plotting
  storei(k,:,:) = ik; % store current for later plotting
end % for k
```

2.11 *Example PCM results*

Every time the code in Sect. 2.10 is executed, the results differ due to the random cell parameter values. Fig. 2.14 shows individual cell states of charge versus time, and current versus time for one representative simulation as corresponding colored lines. We see that cell states of charge in a PCM can be quite different during cycling due to the different capacities and resistances, but that they converge to the same value during rest. That is, there is an automatic self-balancing mechanism due to the parallel electrical connection of the cells comprising a PCM.

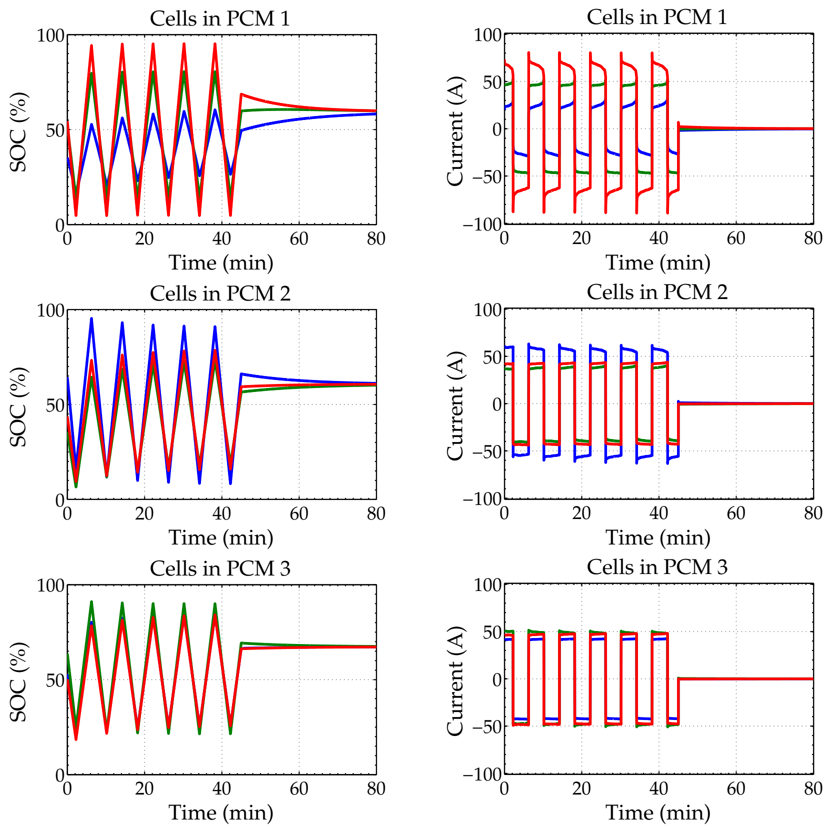


Figure 2.14: Representative state of charge and current profiles produced by the PCM simulator.

We also see that the current experienced by individual cells in a PCM can be quite different from each other due to the varying resistances. Also, cell current is not necessarily zero when applied battery-pack current is zero. This is because the cells may have unequal states of charge, causing circulating balancing currents because of the parallel electrical connections of cells within a PCM.

Fig. 2.15 plots average cell SOC in each PCM, and the maximum of the averaged SOCs minus the minimum of the averaged SOCs. The first plot shows that while there is self-balancing within a PCM due to the parallel electrical connection of the cells, there is no balancing among the set of PCMs. The second plot reinforces this observation by showing that the divergence in state of charge does not decay to zero. It can increase or decrease temporarily during cycling as cells differentially absorb the load, but in steady state the individual PCMs do not converge to the same global average SOC level.

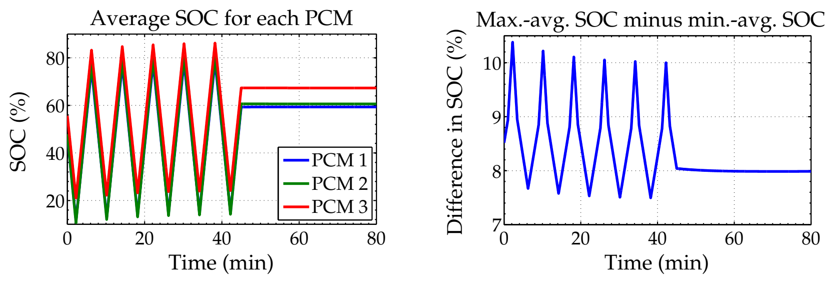


Figure 2.15: Average SOC results from the PCM simulator.

2.12 *SCM simulation code*

The code from Sect. 2.10 can be modified in a straightforward way to simulate a battery pack comprising series-connected cell modules rather than parallel-connected cell modules. The only revision needed is to the main simulation loop. Now, Eq. (2.29) is evaluated to produce the (scalar) battery-pack voltage `V`. Then, Eq. (2.30) is used to determine the N_p distinct SCM currents `ik`. These currents are then replicated for every cell in every SCM to make the final $N_s \times N_p$ matrix of currents `ik`. The rest of the code is unchanged.

```
% ------------------------------------------------------------------
% Okay... now to simulate pack performance using ESC cell model.
% ------------------------------------------------------------------
for k = 1:maxtime,
  v = OCVfromSOCtemp(z,25,model); % get OCV for each cell: Ns * Np matrix
  v = v + m.*h - r.*irc; % add in hysteresis and diffusion voltages
  r0(isnan(z)) = Rsc; % s-c fault has "short-circuit" resistance
  V = (sum(sum(v,1)./sum(r0,1),2)-I)./sum(1./sum(r0,1),2); % Bus V
  ik = (sum(v,1)-repmat(V,1,Np))./sum(r0,1); % 1 * Np cell currents
  ik = repmat(ik,Ns,1);
  z = z - (1/3600)*ik./q;  % Update each cell SOC
  z(z<0) = NaN; % set over-discharged cells to short-circuit fault
  irc = rc.*irc + (1-rc).*ik; % Update diffusion currents
  Ah = exp(-abs(g.*ik)./(3600*q));
  h = Ah.*h + (Ah-1).*sign(ik); % Update hysteresis voltages
  if min(z(:)) < 0.05, I = -abs(I); end % stop discharging
  if max(z(:)) > 0.95, I = abs(I);  end % stop charging
  if k>t0, I = 0; end % rest
  storez(k,:,:) = z; % Store SOC for later plotting
  storei(k,:,:) = ik; % store current for later plotting
end % for k
```

2.13 *Example SCM results*

Every time the code in Sect. 2.12 is executed, the results vary due to the random cell parameter values. Fig. 2.16 shows individual cell states of charge versus time as different colored lines, and corresponding current versus time for one representative simulation where each SCM had eight cells connected in series and the pack comprised three SCMs connected in parallel. We see that current experienced by all cells within any given SCM is the same due to the series connection, but that cells in different SCMs experience different current. We notice that cells in an SCM do not self-balance since they are not electrically connected in parallel. The overall bus voltage of the three SCMs is the same, but that does not force individual cell voltages to be the same.

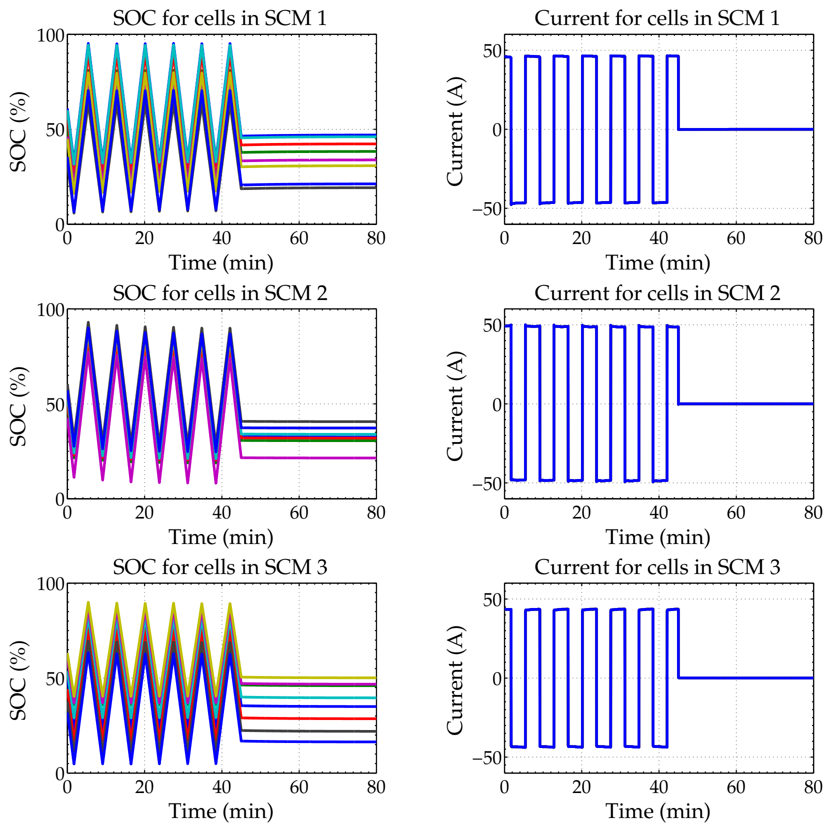


Figure 2.16: Representative state of charge and current profiles produced by the PCM simulator.

2.14 *Where to from here?*

We have now reviewed two types of battery-cell models: the empirical equivalent-circuit model and the physics-based model. Our focus in this book is on applying the empirical model to define algorithms to meet the controls requirements of a battery-management system. A planned Volume III in this series will show how to apply the physics-based models to solve battery-management problems.

To be able to simulate the performance of a battery pack under realistic conditions, it is important to understand how to model the load connected to the battery pack. In this chapter, we have derived equations for an electric-vehicle load and have seen simulation results that, in part, include profiles of battery-pack power or current versus time that can be used as input to a battery-pack simulation.

Further, as a battery pack comprises many cells—possibly configured in parallel and/or series—we have seen that it is possible and even vital to be able to simulate every cell in the battery pack to ensure that no cell goes outside of operational design limits as the battery pack operates. Even cells connected electrically in parallel, which we know have identical terminal voltage due to Kirchhoff's voltage law, can actually exhibit very different internal behaviors during transient operation.

With this review and background, we are now prepared to embark on the first major new topic of this book: learning how to estimate the dynamic values of the internal state vector of every cell in a battery pack as it operates. As this vector includes state of charge as one of its components, this process will be key to any state of charge estimation task. However, we will see that it is also possible to estimate all other model states, and will find in Chap. 6 that these states can be helpful when computing aggressive but safe power limits for the battery pack.

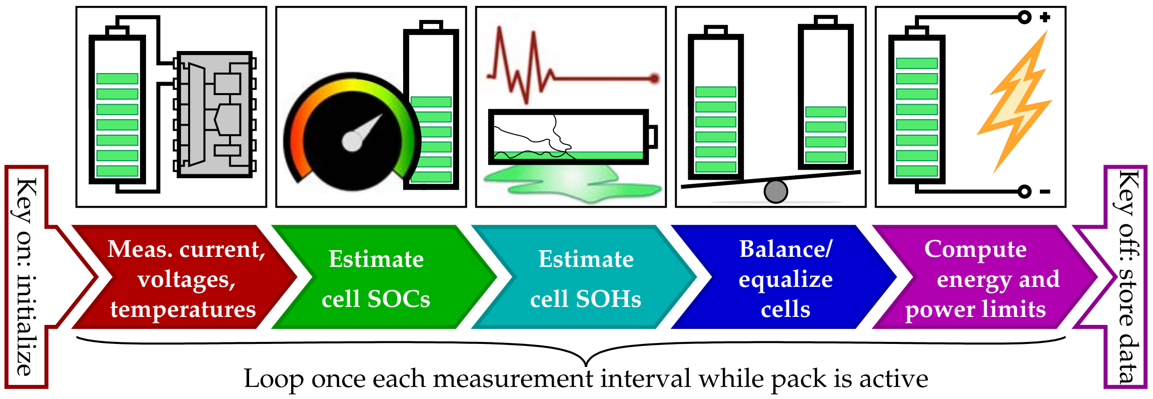
3
Battery-State Estimation

The primary focus of this book is on developing equivalent-circuit-based algorithms for battery management. These algorithms are invoked repeatedly within the main battery-management system control loop, as illustrated in Fig. 3.1.

When the battery-management application is launched, the different algorithms are initialized. In an automotive application, this occurs when the driver turns the key to the "on" position and may require making some initial measurements, loading saved parameter values from a nonvolatile memory, performing safety checks, and/or closing the contactors.

After initialization, the BMS enters its main control loop, which executes at a regular frequency. First, the battery-pack current, cell voltages, and temperatures are measured. Then, the state of every battery cell is estimated, which includes computing an estimate of SOC. Next, a health estimate for every cell is updated. Cells having unequal SOCs may be balanced at this point, and battery-pack energy and power limits are calculated and transmitted to the load-management system.

Finally, when the application is terminated, the contactors are



Key on: initialize
Meas. current, voltages, temperatures
Estimate cell SOCs
Estimate cell SOHs
Balance/ equalize cells
Compute energy and power limits
Key off: store data
Loop once each measurement interval while pack is active
+
-

Figure 3.1: The battery-management-system main algorithm control loop.

11:56:42.

opened and data representing the current battery-pack state are saved to nonvolatile memory to be reloaded and used the next time the battery pack is needed.

Fig. 3.1 can be seen as a roadmap for the bulk of this book. Chap. 1 has already considered the measurement requirements of a battery-management system. This chapter explores the task of battery-state estimation using these measurements and an equivalent-circuit model of the battery-pack cells. Chap. 4 focuses on the challenges of state-of-health estimation for the cells in the battery pack. Chap. 5 discusses balancing, and Chaps. 6 and 7 investigate different approaches to computing battery-pack power limits.

3.1 SOC estimation

The algorithms of a battery-management system are required to estimate quantities that describe the present battery pack condition, but which may not be measured directly. Some of these, such as cell SOC, diffusion currents, and hysteresis states vary relatively quickly, with significant changes possible over the course of seconds or minutes. Others tend to vary slowly, if at all. For example, cell capacities and resistances might change only a few percent in value over years of battery-pack use.

We refer to the quickly changing quantities as the *states* of the cells, and to the slowly changing quantities as the *parameters* of the cell model. In the ESC cell model, elements of x_k are the model states; quantities such as total capacity, diffusion capacitance and resistance, equivalent-series resistance and so forth are the parameters. This chapter considers battery-cell state estimation and Chap. 4 investigates estimation of select battery-cell parameter values.

An important element of the battery-model state vector is the cell's SOC z_k. A SOC estimate is required as input to balancing strategies and to both energy and power calculations. While there can be considerable value in estimating the entire battery-model state vector, we focus first on estimating SOC only.

Recall that SOC is something like a dashboard fuel gauge that reports a value between "empty" (0 %) and "full" (100 %). While there exist sensors that can measure a gasoline level in a tank accurately, there are (presently) no sensors available to measure SOC. So we must somehow combine measured current, voltage, temperature, and knowledge from a cell model to calculate estimates of SOC.

There are some simple methods that give poor estimates, and more complex methods that give very good estimates.[1] Complexity has a cost: more engineering time is required to develop and validate the better methods, and a more capable BMS processor is required to

[1] For a good survey, see S. Piller, M. Perrin, and A. Jossen, "Methods for state of charge determination and their applications," *Journal of Power Sources,* 96(1), 2001, pp. 113–120.

execute them. However, accurate SOC estimates also provide numerous benefits:

Longevity: If the gasoline tank in a standard vehicle is overfilled or runs empty, the tank itself is not damaged. However, overcharging or overdischarging a battery cell may cause permanent damage and result in a reduced lifetime. An accurate SOC estimate is necessary to guarantee that the battery pack is never overcharged or overdischarged.

Performance: Without a good SOC estimator, one must be overly conservative when using the battery pack to avoid overcharge or overdischarge due to trusting the poor estimate. With a good estimate, especially one that is accompanied by a reliable confidence interval on its value, we can safely and aggressively use the entire pack capacity.

Reliability: Poor estimators behave differently for different battery-pack usage profiles. A good SOC estimator is consistent and dependable for any profile, enhancing overall power-system reliability.

Density: Accurate state of charge and battery-state information allow the battery pack to be used aggressively within its design limits, so the pack does not need to be overengineered. This allows smaller, lighter battery packs.

Economy: Smaller battery systems cost less. Reliable battery systems incur lower levels of warranty-servicing costs.

These benefits often outweigh the added cost of implementing an advanced SOC-estimation algorithm.

3.2 *A careful definition of state of charge*

Chap. 1 introduced an electrochemical definition of SOC. With reference to Fig. 3.2, we define the present average lithium concentration stoichiometry as $\theta_k = c_{s,\text{avg},k}/c_{s,\max}$ at time index k. This stoichiometry is intended to remain between $\theta_{0\%}$ and $\theta_{100\%}$, although it is possible to violate these limits in an overdischarge or overcharge situation.

Cell SOC z_k is then computed as

$$z_k = \frac{\theta_k - \theta_{0\%}}{\theta_{100\%} - \theta_{0\%}}.$$

The issue addressed here is that there is (presently) no direct way to measure the concentrations that would allow us to calculate the stoichiometries and from them the SOC. So, we must infer or estimate SOC using only measurements of cell terminal voltage, current, and temperature.

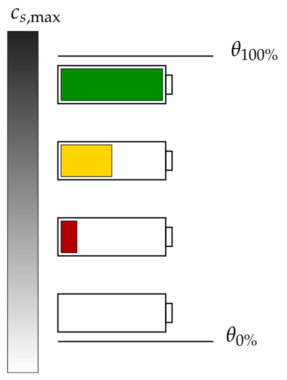


Figure 3.2: Relationship between negative-electrode average lithium concentration and cell SOC. (Duplicated from Fig. 1.19.)

We have already noticed that while cell open-circuit voltage is closely related to state of charge, the terminal voltage under load is a poor predictor of open-circuit voltage unless the cell is in electrochemical equilibrium (and hysteresis is negligible).

This brings up two problems. The first is, how do we estimate SOC in the first place? We will see a number of answers to this problem in this chapter. The second is, how can we know the true state of charge against which to evaluate our estimators?

To address the second problem, we introduce some careful definitions that will ultimately motivate procedures to calibrate laboratory results so that the truth values can be known.

DEFINITION*:* A cell is *fully charged* when its open-circuit voltage reaches $v_h(T)$, a manufacturer-specified voltage that may be a function of temperature T. For example, $v_h(25\,^\circ\mathrm{C}) = 4.2\,\mathrm{V}$ for some lithium-manganese-oxide chemistries and $v_h(25\,^\circ\mathrm{C}) = 3.6\,\mathrm{V}$ for some lithium-iron-phosphate chemistries.

A common method to bring a cell to a fully charged state is to execute a constant-current charge profile until the terminal voltage is equal to $v_h(T)$ followed by a constant-voltage profile until the charging current becomes infinitesimal. This can be done readily in a laboratory setting, allowing calibration of a dataset at the beginning of a test. We define the SOC of a fully charged cell to be 100 %.

DEFINITION*:* A cell is *fully discharged* when its open-circuit-voltage reaches $v_l(T)$, a manufacturer specified voltage that may be a function of temperature. For example, $v_l(25\,^\circ\mathrm{C}) = 3.0\,\mathrm{V}$ for some lithium-manganese-oxide chemistries and $v_l(25\,^\circ\mathrm{C}) = 2.0\,\mathrm{V}$ for some lithium-iron-phosphate chemistries.

A cell may be fully discharged by executing a constant-current discharge profile until its terminal voltage is equal to $v_l(T)$ followed by a constant-voltage profile until the discharge current becomes infinitesimal. Again, this can be done readily in a laboratory setting, allowing calibration of a dataset at the end of a test. We define the SOC of a fully discharged cell to be 0 %.

DEFINITION*:* The *total capacity* Q of a cell is the quantity of charge removed from a cell as it is brought from a fully charged state to a fully discharged state.[2] While the SI unit for charge is coulombs (C), it is more common in practice to use units of ampere-hours (Ah) or milliampere-hours (mAh) to measure the total capacity of a battery cell. The total capacity of a cell is a parameter of our models but is not strictly a fixed quantity: it generally decays slowly over time as the cell ages (we will discuss this further in Chap. 4).

[2] In this book, the term *capacity* refers to a charge capacity, not an energy capacity. This is the more relevant concept for our study since the inputs to our battery-cell models are the cell current; that is, a rate-of-change of charge.

DEFINITION: The *discharge capacity* $Q_{[\text{rate}]}$ of a cell is the quantity of charge removed from a cell as it is discharged at a constant rate from a fully charged state until its loaded terminal voltage reaches $v_l(T)$. Because the discharge capacity is determined based on loaded terminal voltage rather than open-circuit voltage, it is strongly dependent on the cell's internal resistance, which itself is a function of rate and temperature. Hence, the discharge capacity of a cell is rate-dependent and temperature-dependent, while the total capacity is not.

Because of the resistive $i_k \times R_0$ drop, discharge capacity is less than total capacity unless the discharge rate is infinitesimal. Likewise, the SOC of the cell is nonzero when the terminal voltage reaches $v_l(T)$ at a noninfinitesimal rate. The discharge capacity of a cell at a particular rate and temperature is not a fixed quantity: it also generally decays slowly over time as the cell ages.

DEFINITION: The *nominal capacity* Q_{nom} of a cell is a manufacturer-specified quantity that indicates the amount of charge that the cell is rated to hold. Its value tends to be determined by the anticipated application of the cell: for long-duration batteries, it is often the C/8-rate discharge capacity $Q_{0.125C}$, for UPS applications, it is often the 4C-rate discharge capacity Q_{4C}, and for automotive, it is usually close to the 1C-rate discharge capacity Q_{1C} of a particular manufactured lot of cells at room temperature, 25 °C. The nominal capacity is a constant value. Because the nominal capacity is representative of a manufactured lot of cells and the discharge capacity is representative of a single individual cell, $Q_{\text{nom}} \neq Q_{1C}$ in general, even at beginning of life. Also, because Q_{nom} is representative of a discharge capacity and not of a total capacity, $Q_{\text{nom}} \neq Q$.

DEFINITION: The *residual capacity* of a cell is the quantity of charge that would be removed from a cell if it were brought from its present state to a fully discharged state.

DEFINITION: The SOC of a cell is the ratio of its residual capacity to its total capacity.

These definitions are consistent with the continuous-time and discrete-time relationships presented earlier:

$$z(t) = z(0) - \frac{1}{Q}\int_0^t \eta(t)i(t)\,\mathrm{d}t, \quad \text{and} \quad z_{k+1} = z_k - \eta_k i_k \Delta t/Q. \qquad (3.1)$$

True SOC may then be calibrated in a laboratory setting using high-accuracy sensing and the following approach. First, before a test is run, the cell is fully charged to 100 % SOC. This is often done at a controlled room temperature (25 °C). Then, ambient temperature is modified to the test temperature and the cell is allowed to soak at the test temperature until thermal equilibrium of cell internal temperature is achieved. The desired test procedure is executed and net accumulated discharge ampere-hours are recorded continuously as the test runs. The temperature is then returned to 25 °C and the cell temperature is allowed to equilibrate. Finally, the cell is fully discharged. From these data, we can determine total capacity Q via the total net ampere-hours discharged and cell SOC at every point via Eq. (3.1), if we assume that $\eta = 1$. If $\eta \neq 1$, then a full charge and discharge can be performed either prior to or subsequent to the main test to estimate its value; in this case

$$\eta = \frac{\textit{total capacity measured while discharging}}{\textit{total capacity measured while charging}}.$$

3.3 *Some approaches to estimate SOC*

This procedure can work well in a laboratory setting but is not feasible to implement in an embedded application since we cannot afford highly calibrated sensors, nor can we perform this complex process of calibrating cell starting and ending states without interfering with the primary mission of the battery pack application. So, how do we estimate state of charge in practice? In this section, we consider methods that are based primarily on voltage measurements, a method that is based primarily on current measurements, and a more general approach that uses both voltage and current measurements together with an accurate cell model.

3.3.1 *Poor, voltage-based methods to estimate SOC*

Based on the ESC cell model, we know that a cell's terminal voltage is a function of its SOC via:

$$v_k = \mathrm{OCV}(z_k) + Mh_k + M_0 s_k - \sum_i R_i i_{R_i} - i_k R_0.$$

If the cell is at rest and hysteresis is ignored, then we have the very simple relationship $v_k \approx \mathrm{OCV}(z_k)$.

By now, we are accustomed to computing open-circuit voltage as a function of SOC, perhaps using a lookup table. Now, to solve $v_k \approx \mathrm{OCV}(z_k)$ for SOC as a function of OCV, we need to perform the inverse operation. To see what this result looks like, consider the OCV versus SOC relationships that are plotted for six different lithium-ion cells in Fig. 1.21. These can be inverted by simply interchanging plot axes to produce the plots of SOC versus OCV shown in Fig. 3.3. These inverse relationships, again, can be tabulated for later use so that output values can be computed via table lookup. For example, in the figure an OCV of 3.5 V corresponds to an SOC of around 4 % for three of the cells, 9 % for one of the cells, and 99 % for two of the cells. We denote this inverse-table lookup to compute z_k from v_k as $z_k = \text{OCV}^{-1}(v_k)$.

Approximating state of charge via table lookup using the present terminal voltage is truly accurate only when the cell is resting and when hysteresis is negligible; however, it is nonetheless a very simple operation. It is tempting to apply this method even when the cell is under load to find an approximate state of charge from the loaded terminal voltage. However, doing so gives very poor results. It misses the effects of hysteresis, diffusion voltages, and $i_k R_0$ losses. Further, the wide, flat areas of the OCV relationship in Fig. 1.21 translate into steep segments in Fig. 3.3, which dilute the accuracy of the estimate. For example, for the two lithium-iron-phosphate cells in the figure, an open-circuit voltage of 3.3 V corresponds to a state of charge of 42 %. However, changing this voltage by as little as 10 mV in either direction results in states-of-charge of 32 % and 66 %. That is, a relatively small ±10 mV error in computing the OCV produces an enormous 34 % range of error in the estimated SOC.

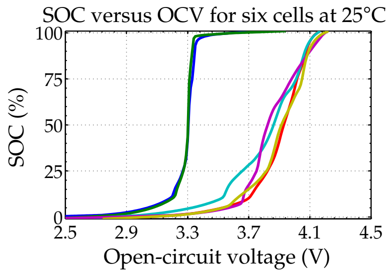


Figure 3.3: SOC as a function of OCV for six lithium-ion chemistries.

For reasonable values of current and cell resistances, the magnitude of the ohmic voltage term alone can be much more than 10 mV. However, this term is also the easiest to compute. So, we can modify the prior overly simplistic state of charge estimation scheme to compensate for the ohmic voltage

$$\begin{aligned} v_k &\approx \text{OCV}(z_k) - i_k R_0 \\ v_k + i_k R_0 &\approx \text{OCV}(z_k) \\ z_k &\approx \text{OCV}^{-1}(v_k + i_k R_0). \end{aligned} \tag{3.2}$$

This produces better results, but still misses the effects of hysteresis and diffusion voltages. Fig. 3.4 shows an example of using this improved voltage-based estimate on a cell having an SOC versus OCV relationship similar to the red line in Fig. 3.3. The cell is subjected to a highly dynamic current profile and voltage is recorded. At every time step, Eq. (3.2) is executed to estimate SOC. This is plotted as the blue line in Fig. 3.4. The data labeled "True SOC" are computed by using the laboratory method described at the end of Sect. 3.2.

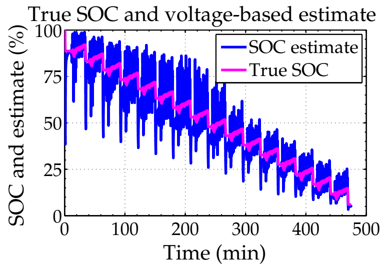


Figure 3.4: Voltage-based estimate of SOC.

We see that the estimates produced using even this improved method are very noisy. It is possible to filter the results to reduce the noise, but filtering adds group delay (a time lag) to the estimates that must then be accounted for. Alternatively, if we knew the diffusion-current state and the hysteresis state of the cell, we could modify Eq. (3.2) to be

$$z_k \approx \text{OCV}^{-1}(v_k - M h_k - M_0 s_k + \sum_i R_i i_{R_i} + i_k R_0).$$

However, we don't have a very good way of knowing these values open-loop, so this approach appears to be a dead end.

In summary, using voltage as the primary estimator of state of charge, perhaps with an $i_k R_0$ correction, produces very noisy estimates that are not useful by themselves. However, in Chap. 4 we will find a use for Eq. (3.2) to stabilize a method for simultaneous SOC and SOH estimation. For now, we turn to looking at current-based methods for estimating state of charge instead.

3.3.2 *Poor, current-based method to estimate SOC*

To use current as the primary estimator of state of charge, we recall that

$$z_k = z_0 - \frac{\Delta t}{Q} \sum_{j=0}^{k-1} \eta_j i_j. \tag{3.3}$$

This equation is, in fact, precise. It measures the amount of charge added to or removed from the cell, normalizes this net amount by total capacity, and updates the state of charge based on this net flow of charge.

If we use this relationship as the basis of an estimator, we are said to be *coulomb counting*. However, we must recognize that what is implemented in practice is really

$$\hat{z}_k = \hat{z}_0 - \frac{\Delta t}{\widehat{Q}} \sum_{j=0}^{k-1} \hat{\eta}_j i_{\text{meas},j}, \tag{3.4}$$

where the hat decoration "ˆ" on a variable indicates an estimate of the value of that variable and where

$$i_{\text{meas},j} = i_{\text{true},j} + i_{\text{noise},j} + i_{\text{bias},j} + i_{\text{nonlin},j} - i_{\text{self-discharge},j} - i_{\text{leakage},j}. \tag{3.5}$$

In Eq. (3.4), $\hat{z}_0$ is the estimate of initial SOC. If this estimate is incorrect, there is no feedback mechanism (e.g., based on voltage) to correct this error over time. All else being ideal, the SOC estimation error over time is fixed at the constant value equal to the error in the $\hat{z}_0$ estimate.

However, not all else is ideal. We don't know the cell total capacity exactly, so must estimate that as well, approximating $Q \approx \widehat{Q}$. Neither do we know the coulombic efficiency η_j exactly, so we must approximate its value with $\hat{\eta}_j$. Both of these approximations contribute to SOC estimation error.

Perhaps most importantly, we don't know the exact current experienced by the cell. We approximate the true cell current in Eq. (3.3) with the measured battery-pack current in Eq. (3.5). This measured current does contain the true cell current $i_{\text{true},j}$, but it also includes random measurement noise $i_{\text{noise},j}$, measurement dc bias $i_{\text{bias},j}$, and nonlinear errors $i_{\text{nonlin},j}$ introduced by the measurement circuit. Further, the measurement does not reflect the self-discharge current of the cell $i_{\text{self-discharge},j}$, nor does it measure how much current $i_{\text{leakage},j}$ is being drawn from the cell to power the electronic circuitry that monitors its performance.

Eq. (3.4) integrates these errors over time. The noise and nonlinear errors might be considered to have zero mean and so would not affect the expected value of the state of charge estimate. They do, however, cause the uncertainty of the estimate to grow continually. The bias, self-discharge, and leakage errors do not have zero mean so they will cause the state of charge estimate to degrade continually, and uncertainty in the measurement error will also cause the uncertainty of the SOC estimate to grow.

So, we conclude that coulomb counting is a risky method to use to estimate SOC. It can be acceptable for short periods of operation when initial conditions are well known. Alternatively, if the application has frequent excursions into voltage regions where the SOC can be estimated reasonably well from measured voltage, the coulomb count can be reset at these points using Eq. (3.2), which tends to make it more reliable.

Coulomb counting, with some kind of reset mechanism, is sometimes the only viable option for estimating SOC. In Fig. 3.3, we saw that the two cells having lithium-iron-phosphate chemistries have voltage curves that yield almost no information on SOC around 3.3 V. However, the other cells do contain significant state of charge information in their OCV, so there should be benefit in somehow combining information from both the current sensor and voltage sensor. We explore this idea next.

3.3.3 *Model-based state estimation*

An alternative to a voltage-only method or a current-only method is to combine the approaches somehow. We can do so by using a model of cell input/output (current/voltage) behaviors, within a *model-based estimation* approach. The resulting method will be able to estimate SOC and all other internal states of the model as well, which yields some additional benefits that we explore in Chap. 6.

The model-based estimation approach is illustrated in Fig. 3.5. The top branch of the diagram shows the operation of the actual cell, which we denote as the "true system." The input to the cell is the electrical current that it experiences and the output from the cell is its terminal-voltage response. Inside the cell, there is a true SOC and a true set of diffusion currents and hysteresis voltages (assuming that our model of the cell makes physical sense). However, these quantities are not measurable, which is why we must estimate their values.

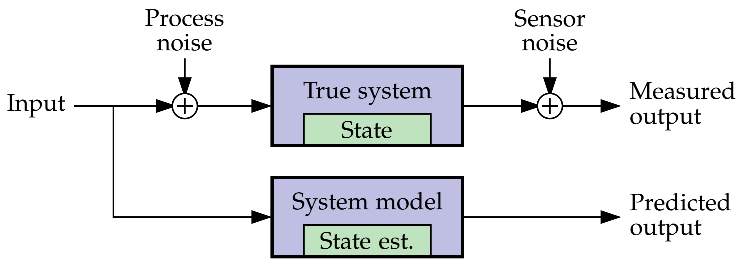


Figure 3.5: The model-based estimation approach.

Complicating the issue is the fact that our current-sensor measurements and our voltage-sensor measurements are both noisy. We model the uncertainty in current as a *process noise*. This unknown and unmeasurable component to cell current does in fact cause the cell's state to change, but we cannot predict the change it effects as we don't know the variation between true and measured current. We model the uncertainty in the voltage as a *sensor noise*. This sensor noise does not affect the true state, but its presence does mean that we cannot place full trust in the accuracy of the voltage measurement.

Model-based estimation realizes that we cannot measure the true system's state, so instead we first measure the input to the true system (the current), and propagate that same input through a model of the system, as shown in the bottom branch of Fig. 3.5. Because our model is implemented in software, the state vector for the model is simply a computer variable and we can do with it whatever we like. We can store it, print it, or use it for other computations. The state-estimate variable in the model is a surrogate for a measurement of the true state.

As described so far, the model-based approach is identical to coulomb counting; however, we're not finished. In the model-based approach, we continue by predicting the system output (the voltage) based on our state estimate and the measured system input. This predicted output is compared to the measured output. If the two are the same, we have confirmation that the model's state estimate is good. If the two are very different, we have an indicator that the model's state estimate is poor. So, the difference between the predicted and measured output can be used in a feedback mechanism to update the model's state estimate. This incorporation of feedback is the vital step to making a good state estimate that is missing from coulomb counting.

We must be very careful, however, when applying the feedback. The voltage-prediction error can be due to a number of factors. These include: state-estimation error (which we would like to correct), measurement errors (due to sensor noise), and modeling errors (because our model is not a perfect representation of the true cell dynamics). We must compute our state-estimate update carefully to account for these sources of error.

Under some specific conditions, the *Kalman filter* is an algorithm that computes a provably optimal state estimate despite these uncertainties. The Kalman filter is a special case of a general solution framework known as *sequential probabilistic inference.* We will look at the linear Kalman filter and some of its variants throughout the remainder of this chapter. But, to do so, we must first explore the more general problem of sequential probabilistic inference, which we do next.

From this point in the chapter, the mathematical level increases. Not every BMS engineer needs to understand every detail. In general, what follows is good background for the BMS hardware engineer to be able to understand sensing requirements, and it is helpful for the BMS software engineer to be able to understand how to interface the main BMS code to the algorithm code. However, we believe that it is essential for the BMS algorithm engineer to study and understand the remainder of this chapter in detail. Real-world implementations of Kalman filters almost always require some modifications to the generic steps presented here to make them work better in an environment where the assumptions made when deriving the filter equations are violated. This is why we take time to derive the steps of the Kalman filter in this book instead of simply listing them. The algorithm designer must know where the equations come from and what they mean in order to modify them or augment them to work in a practical application.

The derivations of the Kalman filter in this chapter are self-contained; however, the reader who is unfamiliar with this topic may wish to consult other references to get a broader perspective.[3] An Internet search will also uncover online courses (including notes and videos of lectures) on the subject of Kalman filtering in general, which may be helpful.[4]

[3] One example is: Simon, D., *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*, Wiley Interscience, 2006, but there are many other excellent texts on the subject.

[4] For one example, see `http://mocha-java.uccs.edu/ECE5550/`.

3.3.4 *Sequential probabilistic inference*

We start by assuming a general, possibly nonlinear, state-space model of a system whose state we would like to estimate:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \tag{3.6}$$

$$y_k = h(x_k, u_k, v_k), \tag{3.7}$$

where $\boldsymbol{u}_k$ is a known (deterministic or measured) input signal, $\boldsymbol{x}_k$ is the model state vector, $\boldsymbol{w}_k$ is an unknown and unmeasurable process-noise random input signal, and $\boldsymbol{v}_k$ is an unknown and unmeasurable sensor-noise random input signal. The output from the system is $\boldsymbol{y}_k$. In the case of the ESC cell model, $\boldsymbol{u}_k$ is the measured cell input current $i_{\text{meas},k}$, and $\boldsymbol{y}_k$ is the measured cell voltage.

Notice that we no longer use v_k to denote voltage, since the name $\boldsymbol{v}_k$ now describes sensor noise. This can be confusing, but we make the switch at this point to be compatible with the majority of the Kalman-filter literature.[5] Note also that $\boldsymbol{y}_k$ is not cell voltage; rather, it is a noisy measurement of cell voltage. The distinction is important.

[5] The literature seems split as to whether the system's output should be denoted as either $\boldsymbol{y}_k$ or $\boldsymbol{z}_k$. However, because we are already using z_k to refer to the cell's SOC, we will refer to the output as $\boldsymbol{y}_k$ in this book.

The functions $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ compute the state and output equations of the model, respectively. We note that they may be time-varying, but we generally omit the time dependency from the notation for ease of understanding. The ESC-model state equation in Eq. (2.4) will later be modified with the inclusion of $\boldsymbol{w}_k$ to form $\boldsymbol{f}(\cdot)$, and the ESC-model output equation in Eq. (2.5) will be modified with the inclusion of $\boldsymbol{v}_k$ to form $\boldsymbol{h}(\cdot)$.

The *sequential-probabilistic-inference* problem seeks to find an efficient recursive estimate of the present state $\boldsymbol{x}_k$ of the dynamic system using knowledge of all inputs and measurements of all outputs up until time k. For compact notation, we define $\mathbb{U}_k$ to be this set of inputs and $\mathbb{Y}_k$ to be this set of outputs. Mathematically, we can write these continuously growing sets as

$$\mathbb{U}_k = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_k\} \tag{3.8}$$

$$\mathbb{Y}_k = \{\boldsymbol{y}_0, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_k\}. \tag{3.9}$$

The solution will be *sequential* in the sense that it implements a recursion that computes the new estimate based on the prior estimate and on the new information measured at this time step; hence, a sequence of steps provides a sequence of estimates. The solution is *probabilistic* in the sense that it must take into account the randomness of the process noise and the sensor noise when computing the estimates.

Fig. 3.6 illustrates the operation of the system under consideration in a way that is helpful to gain insight into the sequential-probabilistic-inference solution. The true system has a state vector whose values evolve over time. This evolution is partially deterministic, via the known $\boldsymbol{u}_k$ input, but it is also partially random via the unknown process-noise input $\boldsymbol{w}_k$. Therefore, we must model the state at a particular point in time as a *vector random variable*, and the state sequence as a *vector random process*. The uncertainty when moving from state $\boldsymbol{x}_{k-1}$ to $\boldsymbol{x}_k$ due to process noise $\boldsymbol{w}_{k-1}$ is modeled by the *conditional probability density function* $f_{\boldsymbol{X}|\boldsymbol{X}}(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$.

The state is unmeasurable, but we are able to observe the noisy

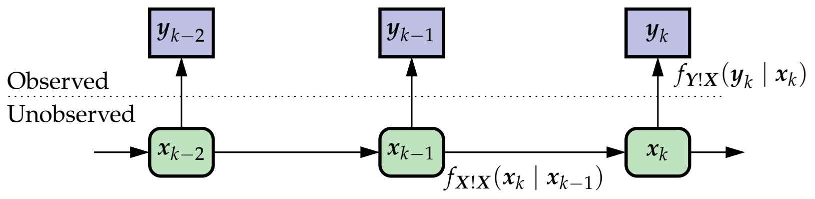


Figure 3.6: The sequential-probabilistic-inference concept.

system outputs $\boldsymbol{y}_k$. These observations allow us some insight into what is happening in the true system. Based on the observations and our model, we estimate the state. However, these measurements are not a deterministic function of the state. Uncertainty due to sensor noise $\boldsymbol{v}_k$ is modeled by the conditional probability density function $f_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}_k \mid \boldsymbol{x}_k)$. Due to the uncertainty of the process-noise and sensor-noise randomness, we will never to be able to compute the state exactly. Instead, we must make estimates of the state that will always have some error. Thus, we will find value in having an associated state-estimation confidence interval computed by the estimator in addition to the estimate itself. This allows the other battery-management-system algorithms that use the estimates produced by the sequential-probabilistic-inference solution to know how aggressively the estimates may be used.

The randomness of process noise and sensor noise have naturally brought up subjects in probability, random variables, and random processes. We assume that the reader has some background in these topics but will now review the most important concepts.

3.4 *Review of random processes*

3.4.1 *Random variables*

By definition, noise is not deterministic: it is random in some sense. So, to discuss the impact of noise on system dynamics, we must understand how to work with numeric quantities whose values change at least to some degree every time we repeat the identical experiment. Such quantities are known as *random variables (RVs)*. We cannot predict beforehand exactly what we will get each time we measure or sample the random variable, but we are able to characterize the relative likelihood of different outcomes by the RV's *probability density function (pdf)*.

THE PDF of random variable X is denoted as $f_X(x)$ and represents the relative likelihood that a measurement of X will result in the value x.[6] Even though the precise value of X is unknown until we make a measurement, we often do have *some* knowledge about which

[6] Note that we use capital letters such as X for the names of RVs, and lowercase letters such as x for the values that these variables might take on.

outcomes are more likely and which outcomes are less likely to occur. Thus, the pdf is somehow a description of what we *do* know about what the uncertain X will be. Values of x for which $f_X(x)$ is large are those we believe are more likely to be observed than values of x for which $f_X(x)$ is small.

To be more precise, a pdf can be any function for which the following three properties are true.[7]

[7] These properties are direct consequences of the *axioms of probability* applied to a continuous random variable.

1. A pdf can never take on negative values. Therefore, $f_X(x) \geq 0$ for all x.
2. The probability that X will take on a value that is less than or equal to x_0 is written as $\Pr(X \leq x_0)$ and is evaluated using its pdf as
$$\Pr(X \leq x_0) = \int_{-\infty}^{x_0} f_X(x)\,\mathrm{d}x.$$
This is easily generalized to find the probability that X is in a finite range as
$$\begin{aligned}\Pr(x_1 < X \leq x_2) &= \Pr(X \leq x_2) - \Pr(X \leq x_1)\\ &= \int_{-\infty}^{x_2} f_X(x)\,\mathrm{d}x - \int_{-\infty}^{x_1} f_X(x)\,\mathrm{d}x\\ &= \int_{x_1^+}^{x_2} f_X(x)\,\mathrm{d}x.\end{aligned}$$
As long as $f_X(x)$ does not contain Dirac delta (impulse) functions, this probability is the same as $\Pr(x_1 \leq X \leq x_2)$ and can be evaluated as $\int_{x_1}^{x_2} f_X(x)\,\mathrm{d}x$. If the pdf has a Dirac delta function exactly at x_1, then we must be careful to integrate from a point just to the right of x_1—denoted as x_1^+—up to x_2 instead. So, assuming that the pdf does not contain Dirac delta functions and that $f_X(x)$ is continuous in the neighborhood of x_0, we can come up with an intuitive understanding of the meaning of $f_X(x)$ by evaluating
$$\begin{aligned}\Pr(x_0 \leq X \leq x_0 + \mathrm{d}x) &= \int_{x_0}^{x_0+\mathrm{d}x} f_X(x)\,\mathrm{d}x\\ &= f_X(x_0)\,\mathrm{d}x\end{aligned}$$
for infinitesimal $\mathrm{d}x$. Therefore, strictly speaking, we cannot say that $f_X(x_0)$ is the probability that $X = x_0$ since the probability that continuous random variable X takes on any particular real value out of the infinite set of possible real values is zero. However, we can say that it is proportional to the probability that X will take on a value in a small neighborhood of x_0. Most correctly, we state that $f_X(x)$ is the relative likelihood that X will take on value x_0.
3. Every experiment must result in some real value. Therefore, we have
$$\Pr(-\infty \leq X \leq \infty) = \int_{-\infty}^{\infty} f_X(x)\,\mathrm{d}x = 1.$$

This normalizing equation states that the area under $f_X(x)$ must be one.

Applying the mathematical abstraction of $f_X(x)$ to a real problem is sometimes very challenging. What should we use as the pdf of process noise for our battery-cell model? How about the pdf of sensor noise? We find that, apart from simple contrived textbook-style examples, it is often difficult to determine the pdf $f_X(x)$ of a real-life random variable with any accuracy. Instead, we use approximations to capture the dominant behavior. To do so, we will need to define some key characteristics of $f_X(x)$.

The *expected value* or *mean* of RV X can be written either as $\bar{x}$ or $\mathbb{E}[X]$, and is defined as

$$\bar{x} = \mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x.$$

This definition can be extended to find the expected value of any function $g(X)$ of X as

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x)\,\mathrm{d}x.$$

A very important property of expectation is that it is linear. Therefore, we can write things like

$$\mathbb{E}[aX + b] = \mathbb{E}[aX] + \mathbb{E}[b] = a\bar{x} + b,$$

if a and b are constants. Similarly, the first moment of X about its mean is

$$\begin{aligned}\mathbb{E}[X - \bar{x}] &= \mathbb{E}[X] - \bar{x} \\ &= \bar{x} - \bar{x} \\ &= 0,\end{aligned}$$

where the first line is true because $\bar{x}$ is a deterministic constant computed from the deterministic function $f_X(x)$ and so can come outside of the expectation operation. The linearity of expectation makes it a much nicer quantity to work with than the pdf itself. We will find that we very rarely need to consider the pdf functional form of an RV after this review section.

The *variance* of an RV is its second central moment around the mean. We define

$$\begin{aligned}\mathrm{var}(X) &= \mathbb{E}[(X - \bar{x})^2] \\ &= \int_{-\infty}^{\infty} (x - \bar{x})^2 f_X(x)\,\mathrm{d}x \\ &= \int_{-\infty}^{\infty} (x^2 - 2\bar{x}x + \bar{x}^2) f_X(x)\,\mathrm{d}x\end{aligned}$$

$$\begin{aligned} &= \mathbb{E}[X^2] - 2\bar{x}\mathbb{E}[X] + \bar{x}^2 \\ &= \mathbb{E}[X^2] - 2\bar{x}\bar{x} + \bar{x}^2 \\ &= \mathbb{E}[X^2] - \bar{x}^2, \end{aligned}$$

or is equal to the mean-square minus the square-mean.[8] Related to the variance, we define the *standard deviation* of RV X as $\sigma_X = \sqrt{\text{var}(X)}$. The standard deviation of X has the same units as X itself, so it is possible to write things like $\bar{x} \pm 3\sigma_X$, as we shall need to do in the future.

[8] Be very careful to note that $\mathbb{E}[X^2] \neq (\mathbb{E}[X])^2$ because the X^2 operation is not linear.

While the mean of an RV captures in a single value the center of the random outcomes (since $\mathbb{E}[X - \bar{x}] = 0$), the variance of a random variable captures the spread or range of random outcomes that we might expect to see. This can be argued via *Chebychev's inequality*, which states (for positive ε)

$$\Pr(|X - \bar{x}| \geq \varepsilon) \leq \frac{\text{var}(X)}{\varepsilon^2}.$$

If the variance of X is small compared to ε, then the probability that X will be further than ε from its mean is small. Conversely, the probability that X will be closer than ε to its mean is large. This implies that probability is concentrated around the mean, and that $\text{var}(X)$ is a relative measure of the spread of the pdf around the mean.

That is, variance informs us of how uncertain we are about the value that an RV will take on. Low variance means that we can predict the value of the RV with very narrow error bounds; high variance means that our prediction will have wide error bounds. The mean and variance together allow us to predict the value of an RV and to state how certain we are of that prediction.

Thus, expectation and variance capture two key features of the actual pdf. While higher-order moments are available, we won't use them in this book.

The most important pdf for the applications examined this book is the *Gaussian* or *normal* distribution. (We will see why this is the case when we consider the *central limit theorem* on page 90.) Several Gaussian pdfs having mean $\bar{x}$ and a number of different variances are shown in Fig. 3.7.

The pdf of a Gaussian random variable X having mean $\bar{x}$ and variance σ_X^2 is defined as

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{(x-\bar{x})^2}{2\sigma_X^2}\right). \tag{3.10}$$

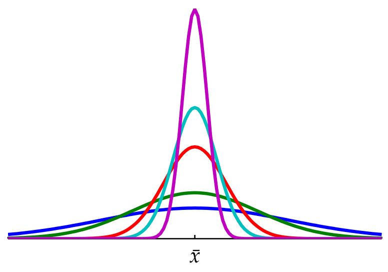


Figure 3.7: Five different Gaussian pdfs, all having mean $\bar{x}$, but with different variances.

By evaluating this equation, or by examining Fig. 3.7, we see that the pdf is symmetric about $\bar{x}$, has peak proportional to $1/\sigma_X$ located at

11:56:42.

$x = \bar{x}$, and has relative width proportional to σ_X. As a short form for this pdf, we often write

$$X \sim \mathcal{N}(\bar{x}, \sigma_X^2).$$

In this notation, the "$\sim$" symbol means "is distributed as" and $\mathcal{N}(a, b)$ denotes a Gaussian pdf with mean a and variance b.[9] Therefore, we read this notation as "Random variable X is distributed as a Gaussian having mean $\bar{x}$ and variance σ_X^2."

[9] The symbol $\mathcal{N}$ is a common notation and stands for "Normal." However, while we use this notation, we will refer to such RVs as Gaussian in this book.

The Gaussian pdf is not easy to integrate, but other properties make it very nice to work with as we shall see. Integration tables are available, and many engineering software toolboxes have Gaussian integration built in. For our purposes, it is sufficient to know that

$$\begin{aligned} \Pr(\bar{x} - \sigma_X \le X \le \bar{x} + \sigma_X) &= 0.683 \\ \Pr(\bar{x} - 2\sigma_X \le X \le \bar{x} + 2\sigma_X) &= 0.955 \\ \Pr(\bar{x} - 3\sigma_X \le X \le \bar{x} + 3\sigma_X) &= 0.997. \end{aligned}$$

Therefore, we can say that a $\pm 3\sigma_X$ interval centered at $\bar{x}$ will almost certainly contain the observed measurement. Again, this confirms that a narrow distribution with small σ_X will have a sharp peak and we have high confidence when predicting X, and that a wide distribution results in poor knowledge in what to expect for X.

3.4.2 *Vector RVs*

With very little change in what we've seen, we can extend the RV paradigm to describe jointly a collection of multiple related RVs using a single *vector RV*. Suppose that we have scalar RVs X_1 that might take on value x_1, X_2 that might take on value x_2, and so forth up to X_n, which might take on value x_n. We can write random vector $\boldsymbol{X}$ and sample vector $\boldsymbol{x}_0$ as

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}, \qquad \boldsymbol{x}_0 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Random vector $\boldsymbol{X}$ is described by a *joint probability density function* $f_{\boldsymbol{X}}(\boldsymbol{x})$, which computes a scalar output for every vector input $\boldsymbol{X}$. Joint pdfs have properties analogous to those of pdfs of scalar RVs. Namely:

1. The joint pdf is always nonnegative: $f_{\boldsymbol{X}}(\boldsymbol{x}) \ge 0$ for all vectors $\boldsymbol{x}$. Note that $f_{\boldsymbol{X}}(\boldsymbol{x}_0)$ is a short form for

$$f_{\boldsymbol{X}}(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n)$$

   where the commas can be read as "and at the same time."

2. The probability $\Pr(\boldsymbol{X} \le \boldsymbol{x}_0)$—which means that within the vector $\boldsymbol{X}$ we have $X_1 \le x_1$ and at the same time $X_2 \le x_2$ and at the same time… $X_n \le x_n$—can be written as
$$\Pr(\boldsymbol{X} \le \boldsymbol{x}_0) = \int_{-\infty}^{x_1}\int_{-\infty}^{x_2}\cdots\int_{-\infty}^{x_n} f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\,\mathrm{d}x_n.$$
3. Similarly, the pdf is normalized by the constraint that every experiment must result in some real vector. Therefore, we have
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\,\mathrm{d}x_n = 1.$$

From these results, we can deduce that $f_{\boldsymbol{X}}(\boldsymbol{x}_0)$ is the relative likelihood that $\boldsymbol{X} = \boldsymbol{x}_0$.

We compute the expected value of random vector $\boldsymbol{X}$ in much the same way as we did for scalar RVs,
$$\bar{\boldsymbol{x}} = \mathbb{E}[\boldsymbol{X}] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \boldsymbol{x} f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\,\mathrm{d}x_n.$$

This operation is linear, as we found before.

However, we cannot compute variance in exactly the same way as before, because the operation X^2 does not make sense for vectors. We must decide whether the correct generalization of X^2 is $\boldsymbol{X}^T\boldsymbol{X}$ (an inner product) or $\boldsymbol{X}\boldsymbol{X}^T$ (an outer product). It turns out that the outer product is much more useful. Accordingly, we define the square *correlation matrix* for random vector $\boldsymbol{X}$ to be
$$\begin{aligned}\boldsymbol{\Sigma}_{\boldsymbol{X}} &= \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T]\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \boldsymbol{x}\boldsymbol{x}^T f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\,\mathrm{d}x_n.\end{aligned}$$

We also define $\widetilde{\boldsymbol{X}} = \boldsymbol{X} - \bar{\boldsymbol{x}}$ to be the variation in $\boldsymbol{X}$ about its mean $\bar{\boldsymbol{x}}$. The correlation of this variation gives us the square *covariance matrix* for random vector $\boldsymbol{X}$ as
$$\begin{aligned}\boldsymbol{\Sigma}_{\widetilde{X}} &= \mathbb{E}[(\widetilde{\boldsymbol{X}})(\widetilde{\boldsymbol{X}})^T]\\ &= \mathbb{E}[(\boldsymbol{X} - \bar{\boldsymbol{x}})(\boldsymbol{X} - \bar{\boldsymbol{x}})^T]\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} (\boldsymbol{x} - \bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}})^T f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\,\mathrm{d}x_n.\end{aligned}$$

Covariance is the generalization of variance, applied to a random vector. The covariance matrix $\boldsymbol{\Sigma}_{\widetilde{X}}$ is symmetric and positive-semi-definite (psd). This means that all its eigenvalues are nonnegative and that we can write
$$\boldsymbol{y}^T\boldsymbol{\Sigma}_{\widetilde{X}}\boldsymbol{y} \ge 0$$
for any vector $\boldsymbol{y}$ having same dimension as $\boldsymbol{X}$. Notice that covariance and correlation are identical for zero-mean random vectors.

The entries of the covariance matrix have specific meaning. The diagonal elements—those having row index i and identical column index i—are equal to the variances of the scalar RV X_i. That is,

$$(\mathbf{\Sigma}_{\widetilde{X}})_{ii} = \sigma^2_{X_i}.$$

The off-diagonal elements—those having different row index i and column index j—are related to the product of the standard deviations of X_i and X_j via

$$(\mathbf{\Sigma}_{\widetilde{X}})_{ij} = \rho_{ij}\sigma_{X_i}\sigma_{X_j} = (\mathbf{\Sigma}_{\widetilde{X}})_{ji},$$

where the *correlation coefficient* ρ_{ij} is a measure of linear dependence between X_i and X_j. The magnitude of the correlation coefficient is bounded, $|\rho_{ij}| \leq 1$. When $\rho_{ij} = 0$, there is no linear dependence between X_i and X_j, meaning that knowledge of the specific value of either one of these RVs gives us no greater ability to predict the other using a linear relationship than we already have from the joint pdf. When $\rho_{ij} = 1$, X_j can be computed exactly from X_i using a linear equation where the slope of this equation is positive; when $\rho_{ij} = -1$, X_j can be computed exactly from X_i using a linear equation where the slope of this equation is negative. Nonzero values of $-1 < \rho_{ij} < 1$ indicate that a linear relationship can be of some value in predicting one of the quantities from the other, but that the prediction will not be perfect—there will still be random error between the prediction and the truth.

There are infinite variety of pdfs for random vectors. However, we will need only the multivariable Gaussian pdf for topics we consider in this book. Similar to the scalar random-variable case, we say $\boldsymbol{X} \sim \mathcal{N}(\bar{\boldsymbol{x}}, \mathbf{\Sigma}_{\widetilde{X}})$, which means that "random vector $\boldsymbol{X}$ is distributed as a multivariable Gaussian having mean $\bar{\boldsymbol{x}}$ and covariance $\mathbf{\Sigma}_{\widetilde{X}}$." The proper generalization from the scalar pdf of Eq. (3.10) to a vector pdf is

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}\left|\mathbf{\Sigma}_{\widetilde{X}}\right|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \mathbf{\Sigma}_{\widetilde{X}}^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}})\right), \qquad (3.11)$$

where $|\mathbf{\Sigma}_{\widetilde{X}}| = \det(\mathbf{\Sigma}_{\widetilde{X}})$.[10] The reader can verify that Eq. (3.10) is a degenerate form of Eq. (3.11) when vector $\boldsymbol{X}$ has only a single component.

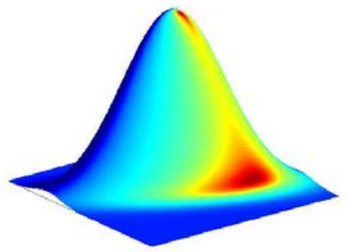


Figure 3.8: A two-dimensional Gaussian pdf.

[10] In this formulation, computing the matrix inverse $\mathbf{\Sigma}_{\widetilde{X}}^{-1}$ requires positive-definite $\mathbf{\Sigma}_{\widetilde{X}}$. It is possible to generalize this definition further to allow for any valid covariance matrix—that is, for any positive-semidefinite $\mathbf{\Sigma}_{\widetilde{X}}$—but we will not focus on that here.

3.4.3 *Properties of jointly distributed RVs*

There are limited analysis opportunities available when considering a single scalar RV, but there is a richness to the kinds of understanding that can come from considering multiple RVs jointly. Some of this can be summarized by determining whether certain properties are

11:56:42.

true of a specific set of RVs. Since we will make extensive use of these properties, we address the most important ones here.

Jointly distributed RVs are said to be *independent* if and only if the joint pdf can be written as

$$f_{\boldsymbol{X}}(x_1, x_2, \ldots, x_n) = f_{X_1}(x_1) f_{X_2}(x_2) \cdots f_{X_n}(x_n)$$

where the *marginal pdf* $f_{X_k}(x_k)$ is the pdf of scalar RV X_k considered by itself. That is, if the joint pdf of $\boldsymbol{X}$ can be written as a product of the marginal pdfs, then the elements of $\boldsymbol{X}$ are independent, and if the elements of $\boldsymbol{X}$ are independent, then the joint pdf of $\boldsymbol{X}$ can be written as a product of the marginal pdfs. This property is not true for the majority of pdfs. However, if it is true for a specific pdf, the implication is that the particular value of the RV X_i has no impact on what values we could obtain for any of the other RVs X_j in $\boldsymbol{X}$. When RVs in $\boldsymbol{X}$ are independent, there is no linear *or nonlinear* dependence between X_i and X_j, meaning that knowledge of the specific value of either one of these RVs gives no greater ability to predict the other using any kind of linear or nonlinear relationship than we already have directly from the joint pdf.

Jointly distributed RVs X_i and X_j are said to be *uncorrelated* if their second moments are finite and

$$\text{cov}(X_i, X_j) = \mathbb{E}[(X_i - \bar{x}_i)(X_j - \bar{x}_j)] = 0,$$

for $i \neq j$. If we consider the covariance matrix $\boldsymbol{\Sigma}_{\widetilde{X}}$ of $\boldsymbol{X}$, we realize that $\text{cov}(X_i, X_j) = (\boldsymbol{\Sigma}_{\widetilde{X}})_{i,j}$. Thus if two RVs are uncorrelated, we must have that $\rho_{ij} = 0$. If all RVs within random vector $\boldsymbol{X}$ are uncorrelated, then $\boldsymbol{\Sigma}_{\widetilde{X}}$ is a diagonal matrix. Based on our previous discussion, we recognize that if two RVs are uncorrelated, then there is no linear dependence between them and that knowledge of the specific value of either one of the RVs gives no greater ability to predict the other using any kind of *linear* relationship than we already have directly from the joint pdf.

The condition for jointly distributed RVs to be independent is much stronger than for them to be uncorrelated. So, if two RVs are independent, then they are also uncorrelated. However, if they are uncorrelated they are not necessarily independent. One very important exception is when jointly distributed RVs have a Gaussian distribution. It turns out that uncorrelated Gaussian RVs are also independent. This is a very special case.

When considering the interactions between multiple RVs, we can also define a *conditional pdf*[11]

$$f_{X_1|X_2}(x_1 \mid x_2) = \frac{f_{\boldsymbol{X}}(x_1, x_2)}{f_{X_2}(x_2)}$$

[11] The marginal probability $f_{X_2}(x_2)$ may be calculated from the joint pdf as

$$f_{X_2}(x_2) = \int_{-\infty}^{\infty} f_{\boldsymbol{X}}(x_1, x_2)\, \mathrm{d}x_1.$$

For each x_2, we integrate out the effect of X_1.

as the relative likelihood that $X_1 = x_1$ given that we know $X_2 = x_2$. The question that is being addressed by a conditional pdf is this: if we have two jointly distributed RVs and we somehow obtain information as to the value of one of them, does that change the probability distribution of what we would expect of the other, which is still unknown? If the two RVs are independent, then we have

$$f_{X_1|X_2}(x_1 \mid x_2) = \frac{f_{X_1}(x_1) f_{X_2}(x_2)}{f_{X_2}(x_2)} = f_{X_1}(x_1),$$

which is the prior marginal distribution for X_1. In this case, knowing the value of X_2 gives no additional insight into what we would expect X_1 to be. However, if the two RVs are not independent, then knowing the value for X_2 *does* give some additional insight, which we can leverage when predicting a value for X_1.

Likewise, we can define *conditional expectation* for jointly distributed random vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ as

$$\mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y} = \boldsymbol{y}] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \boldsymbol{x} f_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x} \mid \boldsymbol{y})\, \mathrm{d}x_1\, \mathrm{d}x_2 \cdots \mathrm{d}x_n, \quad (3.12)$$

which computes what to expect for the value of $\boldsymbol{X}$ given that we know $\boldsymbol{Y} = \boldsymbol{y}$. This is somewhat different from $\mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y}]$. That is, $\boldsymbol{m} = \mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y} = \boldsymbol{y}]$ gives us a deterministic constant vector of values based on the specific knowledge that $\boldsymbol{Y} = \boldsymbol{y}$. On the other hand $\boldsymbol{M} = \mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y}]$ uses the same definition with an unspecified $\boldsymbol{Y}$ and gives us a deterministic function of random variable $\boldsymbol{Y}$. This function computes the random variable $\boldsymbol{M}$, which states what to expect for $\boldsymbol{X}$ if we were to measure a specific value $\boldsymbol{Y} = \boldsymbol{y}$. Because $\mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y}]$ is an RV, we can take its expectation via the rule of *iterated expectation* and find that

$$\mathbb{E}\big[\mathbb{E}[\boldsymbol{X} \mid \boldsymbol{Y}]\big] = \mathbb{E}[\boldsymbol{X}]. \quad (3.13)$$

At this point it is impossible to overemphasize the importance of conditional expectation. Sequential probabilistic inference is an algorithm to compute the expected present value of the system's state vector given all past and present output measurements, $\mathbb{E}[\boldsymbol{x}_k \mid \mathbb{Y}_k]$. We will be making extensive use of expectations and conditional expectations in the remainder of this book.

WHEN WE PERFORM MATHEMATICAL OPERATIONS on one or more RVs, the result is itself a random variable. One important example is the seemingly simple operation that computes

$$Y = X_1 + X_2.$$

Generally speaking, it is quite difficult to find the pdf of Y—it is not simply the summation of the pdfs of X_1 and X_2.[12] If X_1 and X_2 are

[12] This wouldn't even make sense because it would integrate to 2 and violate the conditions for a valid pdf.

independent, then the result is somewhat simpler: the pdf of Y is the convolution of the pdfs of X_1 and X_2. This is still a challenging mathematical operation to perform, but not nearly as difficult as solving the general case for dependent RVs.

Something very interesting happens when we add together *a lot* of RVs that are mutually independent and identically distributed, with each individual variable having finite mean and variance. Then, the summation Y will be approximately normally distributed, and the approximation improves as the number of summed RVs gets large. This result is known as the *central limit theorem* and is the main reason why a Kalman filter makes any sense at all. To completely specify the distribution of Y, we need only find the expected value of the summation—which is the summation of the expected values of the individual vectors being summed—and the covariance matrix of the summation—which is the summation of the covariances of the individual vectors, under the assumed conditions.

Since the state of our dynamic system adds up the effects of lots of independent random inputs in the process-noise signal $\boldsymbol{w}_k$, it is reasonable to assume that the distribution of the state tends to the normal distribution via the central limit theorem. So, we will assume that the state $\boldsymbol{x}_k$ of the system is a normally distributed random vector. We will also ultimately assume that the process noise $\boldsymbol{w}_k$ is a normally distributed random vector, that the sensor noise $\boldsymbol{v}_k$ is a normally distributed random vector, and that $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are uncorrelated with each other. Even when these assumptions are broken in practice, the Kalman filter often works quite well.

For a more general function of the RVs within random vector $\boldsymbol{X}$ that produces random vector $\boldsymbol{Y}$, a useful result allows us to compute the pdf of $\boldsymbol{Y}$ from the pdf of $\boldsymbol{X}$. Let $\boldsymbol{Y} = \boldsymbol{g}(\boldsymbol{X})$ and assume that the inverse function exists such that $\boldsymbol{X} = \boldsymbol{g}^{-1}(\boldsymbol{Y})$. If $\boldsymbol{g}(\cdot)$ and $\boldsymbol{g}^{-1}(\cdot)$ are continuously differentiable, then

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = f_{\boldsymbol{X}}(\boldsymbol{g}^{-1}(\boldsymbol{y})) \left|\left|\left| \frac{\partial \boldsymbol{g}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}} \right|\right|\right|, \tag{3.14}$$

where the notation $|||\cdot|||$ means to take the absolute value of the determinant of the matrix argument.

An important example is the case where $\boldsymbol{Y} = A\boldsymbol{X} + \boldsymbol{B}$ where A is a constant nonsingular matrix, $\boldsymbol{B}$ is a constant vector, and $\boldsymbol{X} \sim \mathcal{N}(\bar{\boldsymbol{x}}, \Sigma_{\tilde{\boldsymbol{X}}})$. We can rewrite this expression to solve for $\boldsymbol{X}$ as

$$\boldsymbol{X} = A^{-1}\boldsymbol{Y} - A^{-1}\boldsymbol{B},$$

which gives us $\boldsymbol{g}^{-1}(\boldsymbol{y}) = A^{-1}\boldsymbol{y} - A^{-1}\boldsymbol{B}$. Then, the *Jacobian matrix* $\partial \boldsymbol{g}^{-1}(\boldsymbol{y})/\partial \boldsymbol{y} = A^{-1}$.

11:56:42.

To find the pdf of $\boldsymbol{Y}$, we begin with the known pdf for $\boldsymbol{X}$, which is

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_{\widetilde{X}}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^T\boldsymbol{\Sigma}_{\widetilde{X}}^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}})\right].$$

Therefore, substituting into Eq. (3.14),

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = \frac{\left\| |\boldsymbol{A}^{-1}| \right\|}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_{\widetilde{X}}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{A}^{-1}(\boldsymbol{y}-\boldsymbol{B})-\bar{\boldsymbol{x}})^T\boldsymbol{\Sigma}_{\widetilde{X}}^{-1}(\boldsymbol{A}^{-1}(\boldsymbol{y}-\boldsymbol{B})-\bar{\boldsymbol{x}})\right].$$

Recognizing that $|\boldsymbol{A}^{-1}| = 1/|\boldsymbol{A}|$, that $|\boldsymbol{A}| = |\boldsymbol{A}^T|$, and that $\bar{\boldsymbol{y}} = \boldsymbol{A}\bar{\boldsymbol{x}} + \boldsymbol{B}$,

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = \frac{1}{(2\pi)^{n/2}(|\boldsymbol{A}||\boldsymbol{\Sigma}_{\widetilde{X}}||\boldsymbol{A}^T|)^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{y}-\bar{\boldsymbol{y}})^T(\boldsymbol{A}^{-1})^T\boldsymbol{\Sigma}_{\widetilde{X}}^{-1}\boldsymbol{A}^{-1}(\boldsymbol{y}-\bar{\boldsymbol{y}})\right].$$

In addition, we note that

$$\begin{aligned}
\boldsymbol{\Sigma}_{\widetilde{Y}} &= \mathbb{E}[(\boldsymbol{Y}-\bar{\boldsymbol{y}})(\boldsymbol{Y}-\bar{\boldsymbol{y}})^T] \\
&= \mathbb{E}[(\boldsymbol{AX}+\boldsymbol{B}-\boldsymbol{A}\bar{\boldsymbol{x}}-\boldsymbol{B})(\boldsymbol{AX}+\boldsymbol{B}-\boldsymbol{A}\bar{\boldsymbol{x}}-\boldsymbol{B})^T] \\
&= \mathbb{E}[(\boldsymbol{A}\widetilde{\boldsymbol{X}})(\widetilde{\boldsymbol{X}}^T\boldsymbol{A}^T)] \\
&= \boldsymbol{A}\boldsymbol{\Sigma}_{\widetilde{X}}\boldsymbol{A}^T.
\end{aligned}$$

Therefore,

$$f_{\boldsymbol{Y}}(\boldsymbol{y}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_{\widetilde{Y}}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{y}-\bar{\boldsymbol{y}})^T\boldsymbol{\Sigma}_{\widetilde{Y}}^{-1}(\boldsymbol{y}-\bar{\boldsymbol{y}})\right].$$

Examining this result, we recognize that it is the pdf of a multivariable Gaussian. That is, $\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{A}\bar{\boldsymbol{x}}+\boldsymbol{B}, \boldsymbol{A}\boldsymbol{\Sigma}_{\widetilde{X}}\boldsymbol{A}^T)$.

This is an extremely important result. It shows us that linear functions of Gaussian random vectors yield a resulting Gaussian random vector. And because the pdf of a Gaussian random vector is uniquely defined by the vector's mean and covariance matrix, which can be found using expectations, we will never need to work with the pdf of the Gaussian itself in the future. That should be reassuring!

For example, let $\boldsymbol{X}$ and $\boldsymbol{W}$ be uncorrelated vector RVs, $\boldsymbol{A}$ and $\boldsymbol{B}$ be constant matrices, and $\boldsymbol{C}$ be a constant vector. If $\boldsymbol{X} \sim \mathcal{N}(\bar{\boldsymbol{x}}, \boldsymbol{\Sigma}_{\widetilde{X}})$ and $\boldsymbol{W} \sim \mathcal{N}(\bar{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\widetilde{W}})$, then we know that $\boldsymbol{Z} = \boldsymbol{AX}+\boldsymbol{BW}+\boldsymbol{C}$ is also Gaussian because it is a linear combination of Gaussians. The pdf of $\boldsymbol{Z}$ is then completely determined if we can find its mean and covariance. First, the mean is

$$\mathbb{E}[\boldsymbol{Z}] = \bar{\boldsymbol{z}} = \boldsymbol{A}\bar{\boldsymbol{x}} + \boldsymbol{B}\bar{\boldsymbol{w}} + \boldsymbol{C}.$$

The covariance is

$$\begin{aligned}
\boldsymbol{\Sigma}_{\widetilde{Z}} &= \mathbb{E}[(\boldsymbol{Z}-\bar{\boldsymbol{z}})(\boldsymbol{Z}-\bar{\boldsymbol{z}})^T] \\
&= \mathbb{E}[(\boldsymbol{AX}+\boldsymbol{BW}+\boldsymbol{C}-\boldsymbol{A}\bar{\boldsymbol{x}}-\boldsymbol{B}\bar{\boldsymbol{w}}-\boldsymbol{C})
\end{aligned}$$

$$
\begin{aligned}
&\quad \times (AX + BW + C - A\bar{x} - B\bar{w} - C)^T] \\
&= \mathbb{E}[(A\widetilde{X} + B\widetilde{W})(A\widetilde{X} + B\widetilde{W})^T] \\
&= \mathbb{E}[A(\widetilde{X})(\widetilde{X})^T A^T + A(\widetilde{X})(\widetilde{W})^T B^T + B(\widetilde{W})(\widetilde{X})^T A^T \\
&\quad + B(\widetilde{W})(\widetilde{W})^T B^T] \\
&= A\Sigma_{\widetilde{X}} A^T + B\Sigma_{\widetilde{W}} B^T,
\end{aligned}
$$

where the last line holds due to X and W being uncorrelated, hence (for example) $\mathbb{E}[(\widetilde{X})(\widetilde{W})^T] = \mathbb{E}[\widetilde{X}]\mathbb{E}[(\widetilde{W})^T] = 0$, because $\mathbb{E}[\widetilde{X}] = \mathbb{E}[X - \bar{x}] = 0$. As the final result for this example, we conclude that Z is a Gaussian random vector and $Z \sim \mathcal{N}(\bar{z}, \Sigma_{\widetilde{Z}})$.

Another valuable example considers creation of correlated Gaussian random vectors from uncorrelated Gaussian random vectors. For example, MATLAB's `randn.m` returns Gaussian random vectors $X \sim \mathcal{N}(\mathbf{0}, I)$ having zero mean and covariance matrix equal to the identity matrix of the appropriate size. Sometimes, we would like to generate Gaussian random vectors $Y \sim \mathcal{N}(\bar{y}, \Sigma_{\widetilde{Y}})$ in a computer program. We can do so via the expression $y = \bar{y} + Ax$ where A is a square matrix such that $AA^T = \Sigma_{\widetilde{Y}}$. Using either of the two previous examples, we have that $\mathbb{E}[Y] = \mathbb{E}[\bar{y} + Ax] = \bar{y}$ since X is zero mean, and that $\Sigma_{\widetilde{Y}} = A\Sigma_{\widetilde{X}}A^T$, which gives the desired result since $\Sigma_{\widetilde{X}} = I$.

To compute the A matrix for this application, we can use one of two different matrix-factoring algorithms. For symmetric positive-definite $\Sigma_{\widetilde{Y}}$, the *Cholesky decomposition* computes $AA^T = \Sigma_{\widetilde{Y}}$, where A is a square lower-triangular matrix. In MATLAB, the `chol.m` command computes the Cholesky decomposition, but we must be careful to invoke it with the `'lower'` optional argument. The following code produces the data in Fig. 3.9.

```
ybar = [1; 2]; covar = [2, 0.75; 0.75, 1];
A = chol(covar,'lower');
for k = 1:5000,
  x = randn([2, 1]);
  y = ybar + A*x;
  plot(y(1),y(2),'.'); hold on
end
```

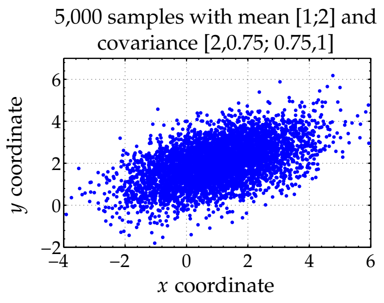


Figure 3.9: Correlated, nonzero-mean Gaussian random vectors.

An alternate approach, which must be used when $\Sigma_{\widetilde{Y}}$ is only positive-semi-definite, is to use the *LDL decomposition*, which computes $LDL^T = \Sigma_{\widetilde{Y}}$, where L is a square lower-triangular matrix and D is a diagonal matrix. In MATLAB, the prior example changes to

```
ybar = [1; 2]; covar = [2, 0.75; 0.75, 1];
[L,D] = ldl(covar);
for k = 1:5000,
  x = randn([2, 1]);
  y = ybar + (L*sqrt(D))*x;
  plot(y(1),y(2),'.'); hold on
end
```

3.4.4 *Vector random (stochastic) processes*

A random variable is a scalar quantity. A random vector is a collection of RVs. A *random process* or *stochastic process* is a family of RVs or random vectors indexed by a parameter set. In our case, that parameter set is time. For example, we might refer to a random process $\boldsymbol{X}_k$ for generic time index k. The value of the random process at any specific time $k = m$ is a random variable or vector $\boldsymbol{X}_m$.

We will usually assume that the random processes we encounter are *stationary*. This means that the behavior has reached a steady-state where the pdfs (and therefore any derived statistics) of the random vectors making up the random process are time-shift invariant. Therefore, for example, $\mathbb{E}[\boldsymbol{X}_k] = \bar{x}$ for all k and $\mathbb{E}[\boldsymbol{X}_{k_1}\boldsymbol{X}_{k_2}^T] = \boldsymbol{\Sigma}_{\boldsymbol{X}_{k_1},\boldsymbol{X}_{k_2}} = \boldsymbol{\Sigma}_{\boldsymbol{X},k_2-k_1}$.[13]

[13] This latter result means that the correlations are not a function of the absolute time indices k_1 and k_2. Instead, they are functions only of the interval length $k_2 - k_1$ between the indices.

Random processes have some properties that are similar to those of random vectors. For example, we define the *autocorrelation function* for a random process to be

$$\boldsymbol{\Sigma}_{\boldsymbol{X}_{k_1},\boldsymbol{X}_{k_2}} = \mathbb{E}[\boldsymbol{X}_{k_1}\boldsymbol{X}_{k_2}^T].$$

If the random process is stationary, then

$$\boldsymbol{\Sigma}_{\boldsymbol{X}_k,\boldsymbol{X}_{k+\tau}} = \mathbb{E}[\boldsymbol{X}_k\boldsymbol{X}_{k+\tau}^T] = \boldsymbol{\Sigma}_{\boldsymbol{X},\tau},$$

for all k. The autocorrelation function provides a measure of correlation between elements of the process having time displacement equal to τ.

We also define the *autocovariance function*

$$\boldsymbol{\Sigma}_{\widetilde{\boldsymbol{X}}_{k_1},\widetilde{\boldsymbol{X}}_{k_2}} = \mathbb{E}[(\boldsymbol{X}_{k_1} - \mathbb{E}[\boldsymbol{X}_{k_1}])(\boldsymbol{X}_{k_2} - \mathbb{E}[\boldsymbol{X}_{k_2}])^T].$$

If the process is stationary,

$$\boldsymbol{\Sigma}_{\widetilde{\boldsymbol{X}}_k,\widetilde{\boldsymbol{X}}_{k+\tau}} = \mathbb{E}[(\boldsymbol{X}_k - \mathbb{E}[\boldsymbol{X}_k])(\boldsymbol{X}_{k+\tau} - \mathbb{E}[\boldsymbol{X}_{k+\tau}])^T] = \boldsymbol{\Sigma}_{\widetilde{\boldsymbol{X}},\tau}$$

for all k.

The autocovariance always has maximum value at $\tau = 0$, where it is equal to the covariance matrix of the underlying random vectors $\boldsymbol{X}_k$ making up the process. If the autocovariance is large for some time shift $\tau \neq 0$, two samples of the random process separated in time by τ have a high correlation and a linear relationship can make a good prediction of one of these quantities using a measured value of the other. If the autocovariance is zero, there is no linear relationship between the samples for that value of and so one cannot be predicted from the other using a linear relationship with any more accuracy than if the other were completely unknown.

We can use these properties to define *white noise* as a stationary random process having zero mean and autocorrelation function

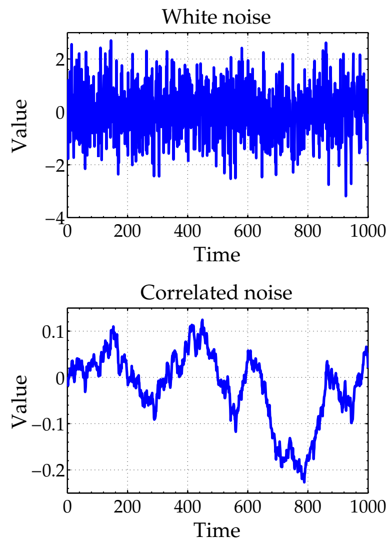


Figure 3.10: Examples of uncorrelated and correlated random processes.

11:56:42.

$\Sigma_{X,\tau} = S_X \delta_\tau$, where δ_τ is the discrete-time Dirac delta function. Therefore, the process is uncorrelated in time. Fig. 3.10 shows examples of white noise and correlated random processes. Values of a white-noise process are completely unpredictable from past samples of the same process, but it is possible to predict the value of a correlated random process with some accuracy given only past samples from the same process.

A white-noise process whose underlying pdf at every point in time is Gaussian is termed a *white Gaussian noise process*. When deriving the sequential-probabilistic-inference solution, we will assume that the noise inputs to the dynamic system are such white Gaussian noise processes.

We have already justified the Gaussian assumption via the central limit theorem. However, this new restriction of whiteness might appear to be overly limiting. Fortunately it is one that can be fixed easily if the random input to the true system is not, in fact, white. We can cascade our system model with a second linear system that is used to shape the noise as desired, as shown in Fig. 3.11. Therefore, we can drive our system $G(z)$ with noise that has desired characteristics by introducing a shaping filter $H(z)$ that itself is driven by white noise. The combined system $GH(z)$ behaves as if $G(z)$ had a shaped-noise input, but $GH(z)$ itself is driven by white noise. The model $GH(z)$ is then used inside the sequential-probabilistic-inference solution rather than $G(z)$ itself.

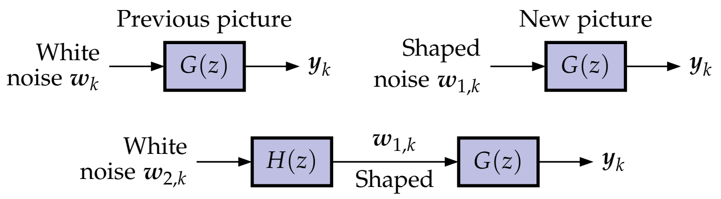


Figure 3.11: Modeling a correlated-noise input via a shaping filter.

This analysis *augments* the original system model with filter states. Suppose that the original system has $G(z)$ with nonwhite input $w_{1,k}$

$$\begin{aligned} x_{k+1} &= Ax_k + B_w w_{1,k} \\ y_k &= Cx_k. \end{aligned}$$

We introduce a shaping filter $H(z)$ with fictitious white input $w_{2,k}$ that produces shaped-noise output $w_{1,k}$

$$\begin{aligned} x_{s,k+1} &= A_s x_{s,k} + B_s w_{2,k} \\ w_{1,k} &= C_s x_{s,k}. \end{aligned}$$

Then, we combine both models into a single system $GH(z)$ by augmenting the state vector

$$\begin{bmatrix} x_{k+1} \\ \hdashline x_{s,k+1} \end{bmatrix} = \begin{bmatrix} A & B_w C_s \\ \hdashline 0 & A_s \end{bmatrix} \begin{bmatrix} x_k \\ \hdashline x_{s,k} \end{bmatrix} + \begin{bmatrix} 0 \\ \hdashline B_s \end{bmatrix} w_{2,k}$$

$$y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \hdashline x_{s,k} \end{bmatrix}.$$

This augmented system, which is the one used in sequential probabilistic inference, is simply a higher-order system driven by white noise.

Before concluding this section we make a remark concerning notation. Until now, we have always used capital letters to denote random variables and vectors. So, as the state of a dynamic system driven by a random process is itself a random vector, we could denote it by X_k. However, it is more common to retain the standard notation x_k and understand from the context that we are discussing a random vector. In the remainder of this book, x_k is to be understood as a random vector sampled from a stochastic process.

3.5 *Sequential probabilistic inference*

With this background, we are ready to derive the sequential-probabilistic-inference solution. Before we do, however, we need to introduce some new notation. So, in the following,

- A superscript "$-$" indicates a *predicted* quantity based only on *past* measurements.
- A superscript "$+$" indicates an *estimated* quantity based on both *past and present* measurements. So, technically, an estimated quantity is different from a predicted quantity.
- The hat decoration "$\hat{\ }$" on a variable indicates either a predicted or an estimated value of that variable. For example, $\hat{x}_k^-$ indicates the predicted value of x_k and $\hat{x}_k^+$ indicates the estimated value of x_k, both at time step k.
- The tilde decoration "$\tilde{\ }$" on a variable indicates an error; that is, the difference between a true and predicted or estimated quantity. For example, $\tilde{x}_k^- = x_k - \hat{x}_k^-$ and $\tilde{x}_k^+ = x_k - \hat{x}_k^+$.
- Consistent with our discussion of random vectors, the symbol "Σ" is used to denote the correlation between the two arguments in its subscript (autocorrelation if only one is given). For example,

$$\Sigma_{xy} = \mathbb{E}[xy^T] \qquad \text{and} \qquad \Sigma_x = \mathbb{E}[xx^T].$$

11:56:42.

- Further, the correlation of mean-subtracted quantities gives covariance

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{x}\tilde{y}} &= \mathbb{E}[\tilde{\boldsymbol{x}}\tilde{\boldsymbol{y}}^T] \\ &= \mathbb{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}])^T].\end{aligned}$$

The sequential-probabilistic-inference problem attempts to find a state estimate that minimizes the *mean-squared error* between the true state and the estimated state, given measurements of the input and output from time zero until the present time. Recall from Eqs. (3.8) and (3.9) that we denote the set of all known inputs up until time k by $\mathbb{U}_k$ and the set of all measurements of the output up until time k by $\mathbb{Y}_k$. The elements of $\mathbb{U}_k$ are not RVs since they are assumed to be measured perfectly (any imperfections in the input measurement are modeled by the process noise $\boldsymbol{w}_k$). However, the output measurements are RVs since they are corrupted by measurement noise $\boldsymbol{v}_k$.

Therefore, we write the *minimum-mean-squared-error (MMSE)* state estimate as the argument vector of values that minimizes the expected norm squared of the error, given all measurements up to and including the present time step:[14]

[14] We will see that the deterministic known values of $\mathbb{U}_k$ are factored into this result as well, but we do not condition on them since they are not RVs.

$$\begin{aligned}\hat{\boldsymbol{x}}_k^{\text{MMSE}}(\mathbb{Y}_k) &= \arg\min_{\hat{\boldsymbol{x}}_k^+} \Big(\mathbb{E}\big[\, \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^+\|_2^2 \mid \mathbb{Y}_k\big]\Big) \\ &= \arg\min_{\hat{\boldsymbol{x}}_k^+} \Big(\mathbb{E}\big[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^+)^T(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^+) \mid \mathbb{Y}_k\big]\Big) \\ &= \arg\min_{\hat{\boldsymbol{x}}_k^+} \Big(\mathbb{E}\big[\boldsymbol{x}_k^T\boldsymbol{x}_k - 2\boldsymbol{x}_k^T\hat{\boldsymbol{x}}_k^+ + (\hat{\boldsymbol{x}}_k^+)^T\hat{\boldsymbol{x}}_k^+ \mid \mathbb{Y}_k\big]\Big).\end{aligned}$$

We solve for $\hat{\boldsymbol{x}}_k^+$ by differentiating the cost function that we wish to minimize with respect to this estimate and setting the result to 0:

$$0 = \frac{\mathrm{d}}{\mathrm{d}\hat{\boldsymbol{x}}_k^+}\mathbb{E}\big[\boldsymbol{x}_k^T\boldsymbol{x}_k - 2\boldsymbol{x}_k^T\hat{\boldsymbol{x}}_k^+ + (\hat{\boldsymbol{x}}_k^+)^T\hat{\boldsymbol{x}}_k^+ \mid \mathbb{Y}_k\big].$$

To do so, note the following identities from vector calculus (for generic vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ and matrix $\boldsymbol{A}$),

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{X}}\boldsymbol{Y}^T\boldsymbol{X} = \boldsymbol{Y}, \quad \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{X}}\boldsymbol{X}^T\boldsymbol{Y} = \boldsymbol{Y}, \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{X}}\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{X}.$$

Further, if the $\boldsymbol{A}$ matrix is symmetric, we have

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{X}}\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X} = 2\boldsymbol{A}\boldsymbol{X}.$$

Then, because the derivative operation is linear, we can take the derivative inside the expectation to find

$$\begin{aligned}0 &= \mathbb{E}\big[-2(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^+) \mid \mathbb{Y}_k\big] = 2\hat{\boldsymbol{x}}_k^+ - 2\mathbb{E}\big[\boldsymbol{x}_k \mid \mathbb{Y}_k\big] \\ \hat{\boldsymbol{x}}_k^+ &= \mathbb{E}\big[\boldsymbol{x}_k \mid \mathbb{Y}_k\big], \qquad (3.15)\end{aligned}$$

where we have also made use of the fact that $\mathrm{d}x_k/\mathrm{d}\hat{x}_k^+ = 0$ because the true system state is not a function of the state estimate. The final result says that the MMSE state estimator is the conditional expectation of the value of the true state given all measurements up to and including the present time.

In general, it is very difficult to evaluate Eq. (3.15). We would need to know all joint probability density functions of all variables (which grow in dimension as time proceeds, due to the growth in $\mathbb{Y}_k$). Then, we would need to compute or approximate the multidimensional integral of Eq. (3.12). *Particle filters* attempt to approximate this result directly, at great computational expense. Thankfully, unless the problem being addressed is very nonlinear, particle filters are probably not necessary to achieve adequate state estimates.

Instead, we proceed by making the assumption that all RVs in our system have a Gaussian distribution. This assumption can be argued from the central limit theorem, assuming that states and noises arise from linear combinations of independent identically distributed noise samples (plus deterministic inputs, which serve only to shift the mean of the uncertainty). The Gaussian assumption breaks down in practice when nonlinear combinations of states and noises are computed, which will be the case for our battery-cell models. Nonetheless, we will see that the simplifying assumptions that we make lead to an algorithm that still gives very good results.

When we make the Gaussian assumption, we can develop a very efficient algorithm for computing $\hat{x}_k^+ = \mathbb{E}\left[x_k \mid \mathbb{Y}_k\right]$ that involves two steps that are executed repeatedly. First, at every time step k, we compute a prediction $\hat{x}_k^- = \mathbb{E}[x_k \mid \mathbb{Y}_{k-1}]$ of the state using prior measurements. Then, we correct the prediction with an update that includes the present measurement y_k to find the estimate $\hat{x}_k^+ = \mathbb{E}[x_k \mid \mathbb{Y}_k]$. The prediction and correction operations require retaining knowledge of results from only the prior step, so memory storage is finite and the computational requirements for every iteration are the same. This makes the resulting algorithm very well suited for implementation on an embedded system such as a BMS.

To proceed with the derivation, we define *prediction error* $\tilde{x}_k^- = x_k - \hat{x}_k^-$.[15] Notice that we cannot compute this error in practice, since the truth value is not known. If it were known, we would not need an estimator! However, we can prove statistical relationships using this definition of prediction error that result in an algorithm for estimating the truth using only measurable values.

[15] Error is always computed as "truth minus prediction" or "truth minus estimate."

We also define the measurement *innovation* as $\tilde{y}_k = y_k - \hat{y}_k$ where $\hat{y}_k = \mathbb{E}\left[y_k \mid \mathbb{Y}_{k-1}\right]$. Because $\hat{y}_k$ is what we predict the measurement value to be, the difference between the measurement y_k and the prediction $\hat{y}_k$ is a surprise to us. We expect $\tilde{y}_k$ to be 0. But, if it

11:56:42.

is not, then there is new information in $\tilde{y}_k$ that we should be able to use somehow to modify our state prediction to make an improved estimate. The $\tilde{y}_k$ term is called the innovation because of this new information it contains.

We can show that both $\tilde{x}_k^-$ and $\tilde{y}_k$ have 0 mean using the method of iterated expectation (cf. Eq. (3.13)):

$$\begin{aligned}\mathbb{E}\big[\tilde{x}_k^-\big] &= \mathbb{E}\big[x_k - \hat{x}_k^-\big]\\ &= \mathbb{E}\big[x_k\big] - \mathbb{E}\big[\mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big]\big]\\ &= \mathbb{E}\big[x_k\big] - \mathbb{E}\big[x_k\big] = 0\\ \mathbb{E}\big[\tilde{y}_k\big] &= \mathbb{E}\big[y_k - \hat{y}_k\big]\\ &= \mathbb{E}\big[y_k\big] - \mathbb{E}\big[\mathbb{E}\big[y_k \mid \mathbb{Y}_{k-1}\big]\big]\\ &= \mathbb{E}\big[y_k\big] - \mathbb{E}\big[y_k\big] = 0.\end{aligned}$$

Note also that $\tilde{x}_k^-$ is uncorrelated with past measurements as they have been incorporated into $\hat{x}_k^-$ already:

$$\begin{aligned}\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_{k-1}\big] &= \mathbb{E}\big[x_k - \hat{x}_k^- \mid \mathbb{Y}_{k-1}\big]\\ &= \mathbb{E}\big[x_k - \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] \mid \mathbb{Y}_{k-1}\big]\\ &= \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] - \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big]\\ &= 0 = \mathbb{E}\big[\tilde{x}_k^-\big].\end{aligned}$$

We derive the recursive correction mechanism by examining the quantity $\mathbb{E}[\tilde{x}_k^- \mid \mathbb{Y}_k]$ in two different ways. First, we write

$$\begin{aligned}\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_k\big] &= \mathbb{E}\big[x_k - \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] \mid \mathbb{Y}_k\big]\\ &= \underbrace{\mathbb{E}\big[x_k \mid \mathbb{Y}_k\big]}_{\hat{x}_k^+} - \underbrace{\mathbb{E}\big[\hat{x}_k^- \mid \mathbb{Y}_k\big]}_{\hat{x}_k^-}.\end{aligned}$$

This is true because the operation $\mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big]$ in the first line results in the deterministic quantity $\hat{x}_k^-$. Because the expected value of a constant is that constant, the additional conditioning on $\mathbb{Y}_k$ in the second line does nothing. Second, we can also write

$$\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_k\big] = \mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_{k-1}, y_k\big] = \mathbb{E}\big[\tilde{x}_k^- \mid y_k\big]$$

because $\tilde{x}_k^-$ is uncorrelated with past measurements.

Putting these two results together, we have the sequential-probabilistic-inference solution

$$\hat{x}_k^+ = \hat{x}_k^- + \mathbb{E}\big[\tilde{x}_k^- \mid y_k\big].$$

That is, first we predict $\hat{x}_k^-$, and then we apply a correction based on the present measured value y_k, which is a predict/correct sequence of steps, as promised.

But, what is $\mathbb{E}\big[\tilde{\boldsymbol{x}}_k^- \mid \boldsymbol{y}_k\big]$? In about a page of math, we can show that when two generic random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are jointly Gaussian distributed,

$$\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{y}] = \mathbb{E}[\boldsymbol{x}] + \boldsymbol{\Sigma}_{\tilde{x}\tilde{y}}\boldsymbol{\Sigma}_{\tilde{y}}^{-1}\big(\boldsymbol{y} - \mathbb{E}[\boldsymbol{y}]\big).$$

In our case, "$\boldsymbol{x}$" is $\tilde{\boldsymbol{x}}_k^-$ and "$\boldsymbol{y}$" is $\boldsymbol{y}_k$. Noting that $\boldsymbol{y}_k = \tilde{\boldsymbol{y}}_k + \hat{\boldsymbol{y}}_k$, we have

$$\begin{aligned}
\mathbb{E}[\tilde{\boldsymbol{x}}_k^- \mid \boldsymbol{y}_k] &= \mathbb{E}[\tilde{\boldsymbol{x}}_k^-] + \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^-\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}\big(\boldsymbol{y}_k - \mathbb{E}[\boldsymbol{y}_k]\big)\\
&= \mathbb{E}[\tilde{\boldsymbol{x}}_k^-] + \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^-\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}\big(\tilde{\boldsymbol{y}}_k + \hat{\boldsymbol{y}}_k - \mathbb{E}[\tilde{\boldsymbol{y}}_k + \hat{\boldsymbol{y}}_k]\big)\\
&= 0 + \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^-\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}\big(\tilde{\boldsymbol{y}}_k + \hat{\boldsymbol{y}}_k - (0 + \hat{\boldsymbol{y}}_k)\big)\\
&= \underbrace{\boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^-\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}}_{\boldsymbol{L}_k}\tilde{\boldsymbol{y}}_k,
\end{aligned}$$

where we have defined the update gain vector $\boldsymbol{L}_k = \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^-\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}$.

Putting all of the pieces together, we get the general update equation

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k.$$

The final result of the sequential-probabilistic-inference solution is the state estimate $\hat{\boldsymbol{x}}_k^+$, which has covariance matrix $\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^+)(\tilde{\boldsymbol{x}}_k^+)^T\big]$ that can be used to determine confidence intervals on the estimate. To evaluate this matrix, we first write

$$\begin{aligned}
\tilde{\boldsymbol{x}}_k^+ &= \boldsymbol{x}_k - \big(\hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k\big)\\
&= \tilde{\boldsymbol{x}}_k^- - \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k.
\end{aligned}$$

Then,

$$\begin{aligned}
\boldsymbol{\Sigma}_{\tilde{x},k}^+ &= \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^- - \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k)(\tilde{\boldsymbol{x}}_k^- - \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k)^T\big]\\
&= \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^-)(\tilde{\boldsymbol{x}}_k^-)^T - \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k(\tilde{\boldsymbol{x}}_k^-)^T - \tilde{\boldsymbol{x}}_k^-(\tilde{\boldsymbol{y}}_k)^T\boldsymbol{L}_k^T + \boldsymbol{L}_k\tilde{\boldsymbol{y}}_k(\tilde{\boldsymbol{y}}_k)^T\boldsymbol{L}_k^T\big]\\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T,
\end{aligned}$$

where we recognize that $\mathbb{E}[\tilde{\boldsymbol{y}}_k(\tilde{\boldsymbol{x}}_k^-)^T] = \boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T$ and that $\mathbb{E}[\tilde{\boldsymbol{x}}_k^-(\tilde{\boldsymbol{y}}_k)^T] = \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}$.

3.5.1 The six-step process

To summarize, the output of the generic Gaussian sequential-probabilistic-inference recursive solution has two components:

1. *The state estimate.* At the end of every iteration, we have computed our best guess of the present state value, which is $\hat{\boldsymbol{x}}_k^+$.
2. *The covariance estimate.* The covariance matrix $\boldsymbol{\Sigma}_{\tilde{x},k}^+$ gives the uncertainty of $\hat{\boldsymbol{x}}_k^+$ and can be used to compute confidence intervals or error bounds on the estimate.

These two quantities together allow us to state that we have high confidence that the truth lies within $\hat{x}_k^+ \pm 3\,\mathrm{diag}\left(\sqrt{\Sigma_{\tilde{x},k}^+}\right)$. These two computations and all supporting definitions are summarized in Table 3.1 and in the Appendix Table on p. 161.

The generic Gaussian sequential probabilistic inference recursion computes

$$\begin{aligned}\hat{x}_k^+ &= \hat{x}_k^- + L_k(y_k - \hat{y}_k) = \hat{x}_k^- + L_k\tilde{y}_k\\ \Sigma_{\tilde{x},k}^+ &= \Sigma_{\tilde{x},k}^- - L_k\Sigma_{\tilde{y},k}L_k^T,\end{aligned}$$

where

$$\begin{aligned}\hat{x}_k^- &= \mathbb{E}[x_k \mid \mathbb{Y}_{k-1}] & \Sigma_{\tilde{x},k}^- &= \mathbb{E}\big[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T\big] = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big]\\ \hat{x}_k^+ &= \mathbb{E}[x_k \mid \mathbb{Y}_k] & \Sigma_{\tilde{x},k}^+ &= \mathbb{E}\big[(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T\big] = \mathbb{E}\big[(\tilde{x}_k^+)(\tilde{x}_k^+)^T\big]\\ \hat{y}_k &= \mathbb{E}[y_k \mid \mathbb{Y}_{k-1}] & \Sigma_{\tilde{y},k} &= \mathbb{E}\big[(y_k - \hat{y}_k)(y_k - \hat{y}_k)^T\big] = \mathbb{E}\big[(\tilde{y}_k)(\tilde{y}_k)^T\big]\\ L_k &= \mathbb{E}\big[(x_k - \hat{x}_k^-)(y_k - \hat{y}_k)^T\big]\Sigma_{\tilde{y},k}^{-1} = \Sigma_{\tilde{x}\tilde{y},k}^-\Sigma_{\tilde{y},k}^{-1}.\end{aligned}$$

Note that this is a linear recursion, even if the system is nonlinear.

Table 3.1: The generic Gaussian sequential-probabilistic-inference recursive solution.

WHEN CONSIDERING HOW TO IMPLEMENT a generic Gaussian sequential-probabilistic-inference solution, it is helpful to divide the calculations into two main steps (prediction and correction), each having three substeps. This overall six-step process forms the pattern for the linear Kalman filter, the extended Kalman filter, and the sigma-point Kalman filter to be developed later in this chapter.

General step 1a: State-prediction time update.

Each time step, we compute an updated prediction of the present value of x_k based on prior information (i.e., before the measurement is taken at time index k) and the system model of Eq. (3.6)

$$\begin{aligned}\hat{x}_k^- &= \mathbb{E}[x_k \mid \mathbb{Y}_{k-1}]\\ &= \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big].\end{aligned} \tag{3.16}$$

General step 1b: Error-covariance time update.

We determine the predicted state-estimate error covariance matrix $\Sigma_{\tilde{x},k}^-$ based on prior information and the system model. That is, we compute

$$\Sigma_{\tilde{x},k}^- = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big], \tag{3.17}$$

where $\tilde{x}_k^- = x_k - \hat{x}_k^-$.

11:56:42.

General step 1c: Predict system output $\boldsymbol{y}_k$.

We predict the system's output using prior information and the model from Eq. (3.7):

$$\begin{aligned}\hat{\boldsymbol{y}}_k &= \mathbb{E}\big[\boldsymbol{y}_k \mid \mathbb{Y}_{k-1}\big] \\ &= \mathbb{E}\big[h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \mid \mathbb{Y}_{k-1}\big].\end{aligned} \tag{3.18}$$

General step 2a: Estimator gain matrix $\boldsymbol{L}_k$.

We compute the estimator gain matrix $\boldsymbol{L}_k$ by evaluating

$$\boldsymbol{L}_k = \boldsymbol{\Sigma}^{-}_{\tilde{x}\tilde{y},k}\boldsymbol{\Sigma}^{-1}_{\tilde{y},k}. \tag{3.19}$$

General step 2b: State-estimate measurement update.

We compute the posterior (i.e., after the measurement is taken) state estimate by updating the prediction using the gain vector $\boldsymbol{L}_k$ and the innovation $\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k$

$$\hat{\boldsymbol{x}}^{+}_k = \hat{\boldsymbol{x}}^{-}_k + \boldsymbol{L}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k). \tag{3.20}$$

General step 2c: Error-covariance measurement update.

Finally, we determine the posterior state-estimate error covariance matrix. That is, we compute

$$\boldsymbol{\Sigma}^{+}_{\tilde{x},k} = \boldsymbol{\Sigma}^{-}_{\tilde{x},k} - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}^T_k. \tag{3.21}$$

THESE SIX STEPS ARE SUMMARIZED in Fig. 3.12. Once all six steps have been executed, the estimator then waits until the next sample interval, updates k, and proceeds to step 1a.

3.6 The linear Kalman filter

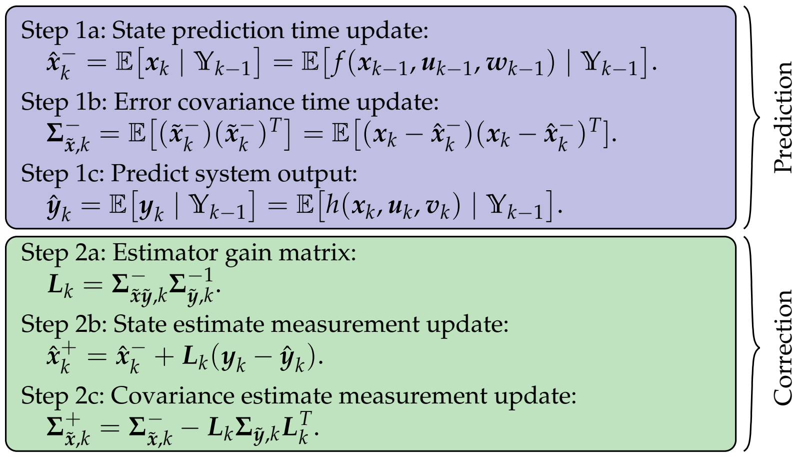
3.6.1 Deriving the linear Kalman filter

In this section, we take the general Gaussian sequential-probabilistic-inference solution and apply it to the specific case where the system dynamics are linear. Linear systems have the desirable property that all random-variable pdfs do in fact remain Gaussian if the stochastic inputs are Gaussian so the assumptions made in deriving the filter steps hold exactly.

The linear Kalman filter assumes that the system being modeled can be represented in state-space form as

$$\begin{aligned}\boldsymbol{x}_k &= \boldsymbol{A}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1} \\ \boldsymbol{y}_k &= \boldsymbol{C}_k\boldsymbol{x}_k + \boldsymbol{D}_k\boldsymbol{u}_k + \boldsymbol{v}_k.\end{aligned}$$

11:56:42.



Step 1a: State prediction time update:
$\hat{x}_k^- = \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big]$.

Step 1b: Error covariance time update:
$\Sigma_{\tilde{x},k}^- = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big] = \mathbb{E}\big[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T\big]$.

Step 1c: Predict system output:
$\hat{y}_k = \mathbb{E}\big[y_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\big]$.

Prediction

Step 2a: Estimator gain matrix:
$L_k = \Sigma_{\tilde{x}\tilde{y},k}^- \Sigma_{\tilde{y},k}^{-1}$.

Step 2b: State estimate measurement update:
$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$.

Step 2c: Covariance estimate measurement update:
$\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^- - L_k \Sigma_{\tilde{y},k} L_k^T$.

Correction

Figure 3.12: The six steps of generic Gaussian probabilistic inference.

We assume that w_k and v_k are mutually uncorrelated white Gaussian noise processes, with zero mean and covariance matrices having known value:

$$\mathbb{E}[w_n w_k^T] = \begin{cases} \Sigma_{\widetilde{w}}, & n = k; \\ 0, & n \neq k; \end{cases} \qquad \mathbb{E}[v_n v_k^T] = \begin{cases} \Sigma_{\tilde{v}}, & n = k; \\ 0, & n \neq k, \end{cases}$$

and $\mathbb{E}[w_k x_0^T] = 0$ for all $k > 0$. The assumptions on the noise processes w_k and v_k and on the linearity of system dynamics are rarely met in practice, but the consensus of the literature and of practice is that the method still works very well.

We now apply the general solution to the linear case and derive the linear Kalman filter. An attempt to aid intuition is also given as we proceed.

KF step 1a: State-prediction time update.

Starting with Eq. (3.16), we compute the predicted state using the assumed linear model.

$$\begin{aligned} \hat{x}_k^- &= \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big] \\ &= \mathbb{E}\big[A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \mid \mathbb{Y}_{k-1}\big] \\ &= \mathbb{E}\big[A_{k-1}x_{k-1} \mid \mathbb{Y}_{k-1}\big] + \mathbb{E}\big[B_{k-1}u_{k-1} \mid \mathbb{Y}_{k-1}\big] + \mathbb{E}\big[w_{k-1} \mid \mathbb{Y}_{k-1}\big] \\ &= A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}, \end{aligned}$$

by the linearity of expectation, noting that w_{k-1} is zero-mean.

intuition: When predicting the present state given only past measurements, the best we can do is to use the most recent state estimate and system model, propagating the state's mean forward in time.

KF step 1b: Error-covariance time update.

To compute the prediction-error covariance, we first note that the prediction error is defined as $\tilde{\boldsymbol{x}}_k^- = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-$. So,

$$\begin{aligned}\tilde{\boldsymbol{x}}_k^- &= \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^- \\ &= (\boldsymbol{A}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1}) - (\boldsymbol{A}_{k-1}\hat{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1}) \\ &= \boldsymbol{A}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{w}_{k-1}.\end{aligned}$$

Therefore, from Eq. (3.17), the covariance of the prediction error is

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{x}_k}^- &= \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^-)(\tilde{\boldsymbol{x}}_k^-)^T\big] \\ &= \mathbb{E}\big[(\boldsymbol{A}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{w}_{k-1})(\boldsymbol{A}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{w}_{k-1})^T\big] \\ &= \mathbb{E}\big[\boldsymbol{A}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+(\tilde{\boldsymbol{x}}_{k-1}^+)^T\boldsymbol{A}_{k-1}^T + \boldsymbol{w}_{k-1}(\tilde{\boldsymbol{x}}_{k-1}^+)^T\boldsymbol{A}_{k-1}^T \\ &\quad + \boldsymbol{A}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+\boldsymbol{w}_{k-1}^T + \boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T\big] \\ &= \boldsymbol{A}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\boldsymbol{A}_{k-1}^T + \boldsymbol{\Sigma}_{\widetilde{w}}.\end{aligned}$$

The cross-terms containing $\boldsymbol{w}_{k-1}(\tilde{\boldsymbol{x}}_{k-1}^+)^T$ and $\tilde{\boldsymbol{x}}_{k-1}^+\boldsymbol{w}_{k-1}^T$ drop out of the final result since the zero-mean white process noise $\boldsymbol{w}_{k-1}$ is not correlated with the state estimation error at time $k-1$. That is,

$$\begin{aligned}\mathbb{E}\big[\boldsymbol{w}_{k-1}(\tilde{\boldsymbol{x}}_{k-1}^+)^T\big] &= \underbrace{\mathbb{E}\big[\boldsymbol{w}_{k-1}\big]}_{0}\,\mathbb{E}\big[(\tilde{\boldsymbol{x}}_{k-1}^+)^T\big] = 0 \\ \mathbb{E}\big[\tilde{\boldsymbol{x}}_{k-1}^+\boldsymbol{w}_{k-1}^T\big] &= \mathbb{E}\big[\tilde{\boldsymbol{x}}_{k-1}^+\big]\,\underbrace{\mathbb{E}\big[\boldsymbol{w}_{k-1}^T\big]}_{0} = 0.\end{aligned}$$

INTUITION*:* When estimating the error covariance of the state prediction, the best we can do is to use the most recent covariance estimate and propagate it forward in time. For stable systems, the $\boldsymbol{A}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\boldsymbol{A}_{k-1}^T$ term is *contractive*, meaning that the uncertainty gets smaller. The state of a deterministic stable system always converges to a known trajectory regardless of the initial condition. As time goes on, this term tells us that we tend to get more and more certain of the state estimate. On the other hand, $\boldsymbol{\Sigma}_{\widetilde{w}}$ adds to the covariance. Unmeasured random input $\boldsymbol{w}_k$ adds uncertainty to our estimate because it perturbs the trajectory away from the known trajectory computed based only on $\boldsymbol{u}_k$.

KF step 1c: Predict system output $\boldsymbol{y}_k$.

We predict the system output via Eq. (3.18) as

$$\begin{aligned}\hat{\boldsymbol{y}}_k &= \mathbb{E}\big[h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \mid \mathbb{Y}_{k-1}\big] \\ &= \mathbb{E}\big[\boldsymbol{C}_k\boldsymbol{x}_k + \boldsymbol{D}_k\boldsymbol{u}_k + \boldsymbol{v}_k \mid \mathbb{Y}_{k-1}\big] \\ &= \mathbb{E}\big[\boldsymbol{C}_k\boldsymbol{x}_k \mid \mathbb{Y}_{k-1}\big] + \mathbb{E}\big[\boldsymbol{D}_k\boldsymbol{u}_k \mid \mathbb{Y}_{k-1}\big] + \mathbb{E}\big[\boldsymbol{v}_k \mid \mathbb{Y}_{k-1}\big] \\ &= \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^- + \boldsymbol{D}_k\boldsymbol{u}_k,\end{aligned}$$

because $\boldsymbol{v}_k$ is zero-mean.

INTUITION: $\hat{y}_k$ is our best guess of the system output given only past measurements. The best we can do is to predict the output using the output equation of the system model and our best guess of the system state at the present time.

KF step 2a: Estimator (Kalman) gain matrix.

To compute the Kalman gain matrix $L_k = \Sigma^-_{\tilde{x}\tilde{y},k}\Sigma^{-1}_{\tilde{y},k}$ from Eq. (3.19), we first need to compute several covariance matrices. We start with $\Sigma_{\tilde{y},k}$ by noting that

$$\begin{aligned}\tilde{y}_k &= y_k - \hat{y}_k \\ &= (C_k x_k + D_k u_k + v_k) - \big(C_k \hat{x}_k^- + D_k u_k\big) \\ &= C_k \tilde{x}_k^- + v_k.\end{aligned}$$

Therefore,

$$\begin{aligned}\Sigma_{\tilde{y},k} &= \mathbb{E}\big[(C_k\tilde{x}_k^- + v_k)(C_k\tilde{x}_k^- + v_k)^T\big] \\ &= \mathbb{E}\big[C_k\tilde{x}_k^-(\tilde{x}_k^-)^T C_k^T + v_k(\tilde{x}_k^-)^T C_k^T + C_k\tilde{x}_k^- v_k^T + v_k v_k^T\big] \\ &= C_k \Sigma^-_{\tilde{x},k} C_k^T + \Sigma_{\tilde{v}}.\end{aligned}$$

Again, the cross-terms are zero since zero-mean white sensor noise v_k is uncorrelated with $\tilde{x}_k^-$. Similarly,

$$\begin{aligned}\mathbb{E}[\tilde{x}_k^- \tilde{y}_k^T] &= \mathbb{E}\big[\tilde{x}_k^-(C_k\tilde{x}_k^- + v_k)^T\big] \\ &= \mathbb{E}\big[\tilde{x}_k^-(\tilde{x}_k^-)^T C_k^T + \tilde{x}_k^- v_k^T\big] \\ &= \Sigma^-_{\tilde{x},k} C_k^T.\end{aligned}$$

Combining these two results, we have an expression for the Kalman gain

$$L_k = \Sigma^-_{\tilde{x},k} C_k^T [C_k \Sigma^-_{\tilde{x},k} C_k^T + \Sigma_{\tilde{v}}]^{-1}.$$

INTUITION: The computation of L_k is the most critical aspect of Kalman filtering that distinguishes it from a number of other state-estimation methods.

The primary reason for calculating covariance matrices in the Kalman-filter steps is to be able to compute L_k. These covariance matrices must be updated every time step because L_k is time-varying. It adapts to give the best correction to the predicted state to form an optimal MMSE state estimate based on all measurements.

Recall that we use L_k in the equation $\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$. The first component to L_k, $\Sigma^-_{\tilde{x}\tilde{y},k}$, indicates relative need for correction to $\hat{x}_k$ and how well individual states within $\hat{x}_k$ are coupled to the measurements.

We see this clearly in $\mathbf{\Sigma}_{\tilde{x}\tilde{y},k}^{-} = \mathbf{\Sigma}_{\tilde{x},k}^{-} \mathbf{C}_k^T$. The $\mathbf{\Sigma}_{\tilde{x},k}^{-}$ component tells us about state uncertainty at the present time, which we hope to reduce as much as possible. A large entry in $\mathbf{\Sigma}_{\tilde{x},k}^{-}$ means that the corresponding state is very uncertain and therefore would benefit from a large update. A small entry in $\mathbf{\Sigma}_{\tilde{x},k}^{-}$ means that the corresponding state is very well known already and does not need as large an update. The $\mathbf{C}_k^T$ term gives the coupling between state and output. Entries that are zero indicate that a particular state has no direct influence on a particular output and therefore an output prediction error should not update that state directly. Entries that are large indicate that a particular state is highly coupled to an output so has a large contribution to any measured output prediction error; therefore, that state would benefit from a large update.

The second component to $\mathbf{L}_k$, $\mathbf{\Sigma}_{\tilde{y}}$, tells us how certain we are that the measurement is reliable. If $\mathbf{\Sigma}_{\tilde{y}}$ is large in some sense, we want small, slow updates. If $\mathbf{\Sigma}_{\tilde{y}}$ is small, we want big updates. This explains why we *divide* the Kalman gain matrix by $\mathbf{\Sigma}_{\tilde{y}}$.

The form of $\mathbf{\Sigma}_{\tilde{y}}$ can also be explained. The $\mathbf{C}_k \mathbf{\Sigma}_{\tilde{x}}^{-} \mathbf{C}_k^T$ part indicates how error in the state contributes to error in the output estimate. The $\mathbf{\Sigma}_{\tilde{v}}$ term indicates the uncertainty in the sensor reading due to sensor noise. Since sensor noise is assumed to be independent of the state, the uncertainty in $\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_k$ is computed by adding the uncertainty in $\mathbf{y}_k$ to the uncertainty in $\hat{\mathbf{y}}_k$.

KF step 2b: State-estimate measurement update.

This step computes the posterior state estimate by updating the prior state prediction using the estimator gain and the output prediction error $\mathbf{y}_k - \hat{\mathbf{y}}_k$ via Eq. (3.20),

$$\hat{\mathbf{x}}_k^{+} = \hat{\mathbf{x}}_k^{-} + \mathbf{L}_k(\mathbf{y}_k - \hat{\mathbf{y}}_k).$$

INTUITION: The variable $\hat{\mathbf{y}}_k$ is what we expect the measurement to be, based on our state prediction at the moment.

Therefore, $\mathbf{y}_k - \hat{\mathbf{y}}_k$ is what is unexpected or new in the measurement. We call this term the innovation. The innovation can result either from a bad system model, state error, or sensor noise. So, we want to use this new information to update the state but must be careful to weight it according to the value of the information it contains. $\mathbf{L}_k$ is the optimal blending factor, as we have already discussed.

KF step 2c: Error-covariance measurement update.

Finally, we update the error covariance matrix via Eq. (3.21),

$$\mathbf{\Sigma}_{\tilde{x},k}^{+} = \mathbf{\Sigma}_{\tilde{x},k}^{-} - \mathbf{L}_k \mathbf{\Sigma}_{\tilde{y},k} \mathbf{L}_k^T.$$

11:56:42.

intuition: The measurement update has decreased our uncertainty in the state estimate.

The update to the state-prediction error covariance $L_k \Sigma_{\tilde{y},k} L_k^T$ is also a positive-semi-definite form. Because we are subtracting this from the predicted-state covariance matrix, the resulting estimated-state covariance is "lower" in some sense than the premeasurement covariance.

If a measurement is missed for some reason, then we simply skip steps 2a–c for that iteration. That is, we set $L_k = 0$, $\hat{x}_k^+ = \hat{x}_k^-$, and $\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^-$. Finally, repeating from before, recall that the estimator output comprises the state estimate $\hat{x}_k^+$ and error covariance estimate $\Sigma_{\tilde{x},k}^+$. That is, we have high confidence that the truth lies within $\hat{x}_k^+ \pm 3\,\text{diag}\left(\sqrt{\Sigma_{\tilde{x},k}^+}\right)$.

3.6.2 *Visualizing the linear Kalman filter*

The Kalman-filter equations are summarized in the Appendix on p. 162 and naturally form the recursion drawn in Fig. 3.13. The filter is initialized with our best guess as to the the values of the present initial state and covariance. Then, the three prediction steps are executed, followed by the three correction steps. The filter then waits until the next sample time, increments the time index k, and repeats. The operations are quite straightforward to perform on a digital computer, as we will see.

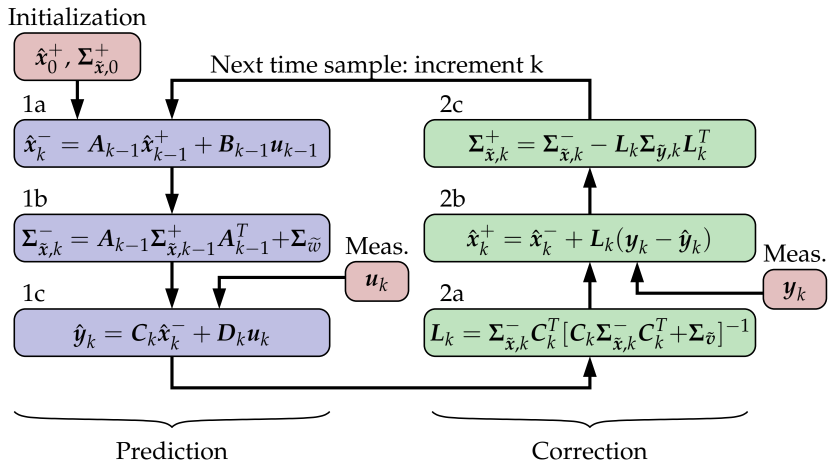

Initialization
$\hat{x}_0^+, \Sigma_{\tilde{x},0}^+$
Next time sample: increment k
1a
$\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}$
1b
$\Sigma_{\tilde{x},k}^- = A_{k-1}\Sigma_{\tilde{x},k-1}^+ A_{k-1}^T + \Sigma_{\tilde{w}}$
Meas.
u_k
1c
$\hat{y}_k = C_k\hat{x}_k^- + D_k u_k$
2c
$\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^- - L_k \Sigma_{\tilde{y},k} L_k^T$
2b
$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$
Meas.
y_k
2a
$L_k = \Sigma_{\tilde{x},k}^- C_k^T [C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}}]^{-1}$
Prediction
Correction

Figure 3.13: Visualizing the Kalman-filter recursion.

Note that while the expectation operations in the generic sequential-probabilistic-inference solution in Table 3.1 are uncomputable without complicated integrals of pdfs, all equations in Fig. 3.13 can be evaluated using data available at that time step and simple matrix/vector operations. The Kalman-filter derivation has evaluated the expectations analytically to give a practical algorithm.

We would like to implement the Kalman-filter steps for a battery-cell model to be able to estimate the internal state of the model. However, note that our cell models are nonlinear, so we cannot apply the (linear) Kalman-filter steps to them directly. Instead, we will need to develop nonlinear versions of the Gaussian sequential-probabilistic-inference steps. We will do so in Sects. 3.7 and 3.10.

For the time being, though, we wish to gain experience and intuition with Kalman-filter methods. To demonstrate the Kalman-filter steps, we will use a crude linearized battery-cell model. First, with reference to Fig. 3.14, we note that the OCV versus SOC relationship for any of the four lithium-ion chemistries illustrated can be approximated roughly by the straight-line function drawn as the black dashed line in the figure:

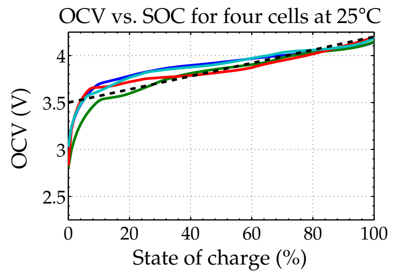


Figure 3.14: Linearizing the OCV relationship.

$$\mathrm{OCV}(z_k) \approx 3.5 + 0.7 z_k.$$

Then, omitting diffusion currents and hysteresis states from the ESC cell model, we write the approximate relationship[16]

$$\begin{aligned} z_{k+1} &= 1 \cdot z_k - \frac{1}{3{,}600 \cdot Q} i_k \\ volt_k &= 3.5 + 0.7 \times z_k - R_0 i_k. \end{aligned}$$

[16] The factor of 3,600 converts ampere-hours to coulombs for an assumed $\Delta t = 1$ s.

This model still is not linear (it is *affine*) because of the additive value 3.5 in the output equation. This is easily remedied by creating a debiased synthetic measurement $y_k = volt_k - 3.5$, and using the model

$$\begin{aligned} z_{k+1} &= 1 \cdot z_k - \frac{1}{3{,}600 \cdot Q} i_k + w_k \\ y_k &= 0.7 \times z_k - R_0 i_k + v_k, \end{aligned}$$

where we have added process-noise and sensor-noise terms w_k and v_k to the equations as well.

Comparing this set of equations to the standard linear state-space model, we see that we are defining the model state $\boldsymbol{x}_k$ to be equal to SOC z_k and model input $\boldsymbol{u}_k$ to be battery-cell current i_k. For the sake of example, we will will use $Q = 10{,}000/3{,}600$ and $R_0 = 0.01$. This yields a state-space description with $A = 1$, $B = -1 \times 10^{-4}$, $C = 0.7$, and $D = -0.01$. We also model $\Sigma_{\widetilde{w}} = 10^{-5}$, and $\Sigma_{\widetilde{v}} = 0.1$.

For the sake of example, we assume an initial cell SOC of 50 % and no initial uncertainty, so $\hat{x}_0^+ = 0.5$ and $\Sigma_{\tilde{x},0}^+ = 0$. We then manually calculate all terms for two iterations of the filter.

11:56:42.

Kalman-filter iteration 1

For the first iteration, we assume that $i_0 = 1$, $i_1 = 0.5$ and $volt_1 = 3.85$. We compute the following, where the left-hand side of each box shows the generic equation being solved, and the right-hand side shows the numeric substitutions and results:[17]

$$\boldsymbol{\hat{x}}_k^- = \boldsymbol{A}_{k-1}\boldsymbol{\hat{x}}_{k-1}^+ + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1} \qquad \hat{x}_1^- = 1 \times 0.5 - 10^{-4} \times 1 = 0.4999$$

$$\boldsymbol{\Sigma}_{\tilde{x},k}^- = \boldsymbol{A}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\boldsymbol{A}_{k-1}^T + \boldsymbol{\Sigma}_{\tilde{w}} \qquad \Sigma_{\tilde{x},1}^- = 1 \times 0 \times 1 + 10^{-5} = 10^{-5}$$

$$\boldsymbol{\hat{y}}_k = \boldsymbol{C}_k\boldsymbol{\hat{x}}_k^- + \boldsymbol{D}_k\boldsymbol{u}_k \qquad \hat{y}_1 = 0.7 \times 0.4999 - 0.01 \times 0.5 = 0.34493$$

$$\boldsymbol{L}_k = \boldsymbol{\Sigma}_{\tilde{x},k}^-\boldsymbol{C}_k^T[\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^-\boldsymbol{C}_k^T + \boldsymbol{\Sigma}_{\tilde{v}}]^{-1} \qquad \begin{aligned} L_1 &= 10^{-5} \times 0.7[0.7^2 \times 10^{-5} + 0.1]^{-1} \\ &= 6.99966 \times 10^{-5} \end{aligned}$$

$$\begin{aligned} \boldsymbol{\hat{x}}_k^+ &= \boldsymbol{\hat{x}}_k^- + \boldsymbol{L}_k(\boldsymbol{y}_k - \boldsymbol{\hat{y}}_k) \\ &\text{(where } y_k = 3.85 - 3.5) \end{aligned} \qquad \begin{aligned} \hat{x}_1^+ &= 0.4999 + 6.99966 \times 10^{-5}(0.35 - 0.34493) \\ &= 0.4999004 \end{aligned}$$

$$\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T \qquad \begin{aligned} \Sigma_{\tilde{x},1}^+ &= 10^{-5} - (6.99966 \times 10^{-5})^2(0.1000049) \\ &= 9.9995 \times 10^{-6} \end{aligned}$$

The output of this iteration gives $\hat{z}_1 = 0.4999 \pm 3\sqrt{9.9995 \times 10^{-6}} = 0.4999 \pm 0.0094866$.

Kalman-filter iteration 2

For the second iteration, we recall that $i_1 = 0.5$, and further specify that $i_2 = 0.25$ and $volt_2 = 3.84$.

$$\boldsymbol{\hat{x}}_k^- = \boldsymbol{A}_{k-1}\boldsymbol{\hat{x}}_{k-1}^+ + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1} \qquad \hat{x}_1^- = 0.4999004 - 10^{-4} \times 0.5 = 0.49985$$

$$\boldsymbol{\Sigma}_{\tilde{x},k}^- = \boldsymbol{A}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\boldsymbol{A}_{k-1}^T + \boldsymbol{\Sigma}_{\tilde{w}} \qquad \Sigma_{\tilde{x},2}^- = 9.9995 \times 10^{-6} + 10^{-5} = 1.99995 \times 10^{-5}$$

$$\boldsymbol{\hat{y}}_k = \boldsymbol{C}_k\boldsymbol{\hat{x}}_k^- + \boldsymbol{D}_k\boldsymbol{u}_k \qquad \hat{y}_2 = 0.7 \times 0.49985 - 0.01 \times 0.25 = 0.347395$$

$$\boldsymbol{L}_k = \boldsymbol{\Sigma}_{\tilde{x},k}^-\boldsymbol{C}_k^T[\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^-\boldsymbol{C}_k^T + \boldsymbol{\Sigma}_{\tilde{v}}]^{-1} \qquad \begin{aligned} L_2 &= 1.99995 \times 10^{-5} \times 0.7[1.99995 \times 10^{-5} \times \\ &\quad 0.7^2 + 0.1]^{-1} = 0.00013998 \end{aligned}$$

$$\begin{aligned} \boldsymbol{\hat{x}}_k^+ &= \boldsymbol{\hat{x}}_k^- + \boldsymbol{L}_k(\boldsymbol{y}_k - \boldsymbol{\hat{y}}_k) \\ &\text{(where } y_k = 3.84 - 3.5) \end{aligned} \qquad \begin{aligned} \hat{x}_2^+ &= 0.49985 + 0.00013998(0.34 - 0.347395) \\ &= 0.499849 \end{aligned}$$

$$\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T \qquad \begin{aligned} \Sigma_{\tilde{x},2}^+ &= 1.99995 \times 10^{-5} - 0.00013998^2 \times \\ &\quad 0.100009799755 = 1.99976 \times 10^{-5} \end{aligned}$$

The output of this iteration gives $\hat{z}_2 = 0.4998 \pm 3\sqrt{1.99976 \times 10^{-5}} = 0.4998 \pm 0.013416$.

Clearly, it is cumbersome to evaluate these steps by hand. However, it is quite straightforward to write computer code to implement them. We shall see how to do so in the next section. In the meantime, Fig. 3.15 shows a sample of the Kalman filter operating for the assumed model of this example. The results labeled "Truth" came from a simulation of the model equations, not from physical measurements from a real battery cell.

The figure shows us a few things. For example, we see that the estimated state never converges to the true state. This is because of

[17] Note that the generic equations on the left-hand side of each box use bolded characters since vectors or matrices might be involved. However, in this specific example, all quantities are scalars, so the right-hand side of each box uses normal font.

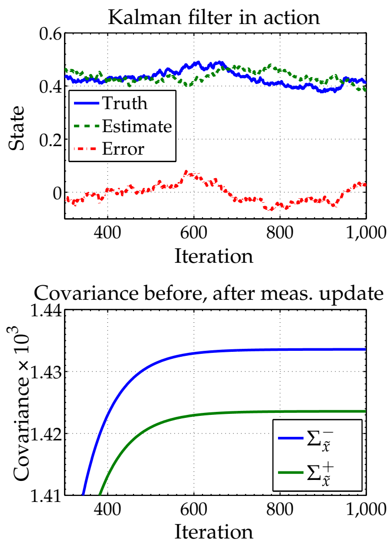


Figure 3.15: Linear Kalman filter in action.

11:56:42.

the process noise, which is always affecting the true state in ways that cannot be uniquely determined without knowing the exact values of w_k. However, the Kalman-filter state estimate does converge to the neighborhood of the true state, and the covariance of the state estimate allows us to place error bounds on the estimate.

The covariance is initialized to zero. Every time step, the prediction causes uncertainty to increase (due to the unknown changes to the state due to process noise) and the correction causes uncertainty to decrease (due to more certainty obtained from the output measurement). The graphic below shows the first few steps of this process.

$$\begin{tikzcd}
 & \Sigma^-_{\tilde{x},1} = 1\times10^{-5} \arrow[d] & \Sigma^-_{\tilde{x},2} = 1.99995\times10^{-5} \arrow[d] & \Sigma^-_{\tilde{x},3} = 2.99976\times10^{-5} \arrow[d] & {} \\
\Sigma^+_{\tilde{x},0} = 0 \arrow[ur] & \Sigma^+_{\tilde{x},1} = 9.99951\times10^{-6} \arrow[ur] & \Sigma^+_{\tilde{x},2} = 1.99976\times10^{-5} \arrow[ur] & \Sigma^+_{\tilde{x},3} = 2.99931\times10^{-5} \arrow[ur] &
\end{tikzcd}$$

Note that for this example covariance (uncertainty) converges to a steady-state solution, but it takes time. The lower frame of Fig. 3.15 shows the prediction-error and estimation-error covariances as a function of iteration. The steady-state prediction-error covariance was 1.434×10^{-3}, and the final estimation-error covariance was 1.424×10^{-3}. Estimation-error bounds are then $\pm 11.3\,\%$ for $99.7\,\%$ confidence (using 3-sigma bounds). So, in this example, the Kalman filter does not perform especially well since $\Sigma_{\tilde{v}}$ is quite large. However, we know that these are the best-possible results for the simulated conditions because we know that the Kalman filter is the optimal MMSE estimator.

3.6.3 *MATLAB code for the linear Kalman filter steps*

It is straightforward to convert the Kalman filter steps to MATLAB. However, great care must be taken to ensure that all k and $k+1$ indices (and so forth) are kept synchronized. In the example code, below, we cosimulate the system equations to produce $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ and we implement a Kalman filter on these input/output data. In a real application, we would measure $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ instead.

The first portion of the code initializes simulation variables and reserves room to store results for plotting:

```
% Initialize simulation variables
SigmaW = 1e-5;      % Process-noise covariance
SigmaV = 0.1;       % Sensor-noise covariance
A = 1;   B = -1e-4; % State-equation matrices
C = 0.7; D = -0.01; % Output-equation matrices
maxIter = 1000;     % Number of simulation time steps

xtrue  = 0.5;  % Initialize true system initial state
xhat   = 0.5;  % Initialize Kalman filter initial estimate
SigmaX = 0;    % Initialize Kalman filter covariance
```

```
u = 1;         % Initial driving input, u[0]

% Reserve storage for variables we might want to plot/evaluate
xstore = zeros(maxIter+1,length(xtrue)); xstore(1,:) = xtrue;
xhatstore = zeros(maxIter,length(xhat));
SigmaXstore = zeros(maxIter,length(xhat)^2);
```

In a BMS implementation, we do not need to store all state values. Instead, we would keep only the most recent value. In this code we store the results only to make it possible to plot the Kalman-filter performance after the simulation concludes.

In the next section of the code, we execute the six Kalman-filter steps. These are straightforward implementations of the model equations. Note that in a BMS implementation, we would measure $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ directly. In the simulation, however, we must come up with reasonable simulated values to use instead.

```
for k = 1:maxIter,
  % KF Step 1a: State-prediction time update
  xhat = A*xhat + B*u; % use prior value of "u"

  % KF Step 1b: Error-covariance time update
  SigmaX = A*SigmaX*A' + SigmaW;

  % [Implied operation of system in background, with
  % input signal u, and output signal z]
  switch k,
    case 1, u = 0.5;  % to match earlier example
    case 2, u = 0.25; % to match earlier example
    otherwise, u = randn(1); % just some interesting input
  end
  w = chol(SigmaW,'lower')*randn(length(xtrue));   % rand. process noise
  v = chol(SigmaV,'lower')*randn(length(C*xtrue)); % rand. sensor noise
  switch k,
    case 1, ytrue = 3.85 - 3.5; % to match example
    case 2, ytrue = 3.84 - 3.5; % to match example
    otherwise, ytrue = C*xtrue + D*u + v; % based on present x and u
  end
  xtrue = A*xtrue + B*u + w;  % future x is based on present u

  % KF Step 1c: Estimate system output
  yhat = C*xhat + D*u;

  % KF Step 2a: Compute Kalman gain matrix
  SigmaY = C*SigmaX*C' + SigmaV;
  L = SigmaX*C'/SigmaY;

  % KF Step 2b: State-estimate measurement update
  xhat = xhat + L*(ytrue - yhat);

  % KF Step 2c: Error-covariance measurement update
  SigmaX = SigmaX - L*SigmaY*L';

  % [Store information for evaluation/plotting purposes]
  xstore(k+1,:) = xtrue;
  xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = SigmaX(:);
end;
```

Finally, we might wish to plot some results. The following code segment plots the true state, estimated state, errors, and error bounds:

```
figure(1); clf;
plot(0:maxIter-1,xstore(1:maxIter),'k-',0:maxIter-1,xhatstore,'b--', ...
  0:maxIter-1,xhatstore+3*sqrt(SigmaXstore),'m-.',...
  0:maxIter-1,xhatstore-3*sqrt(SigmaXstore),'m-.'); grid;
legend('True','Estimate','Bounds','location','northeast');
title('Kalman filter example');
xlabel('Iteration'); ylabel('State');

figure(2); clf;
plot(0:maxIter-1,xstore(1:maxIter)-xhatstore,'b-',0:maxIter-1, ...
  3*sqrt(SigmaXstore),'m--',0:maxIter-1,-3*sqrt(SigmaXstore),'m--');
grid; legend('Error','Bounds','location','northeast');
title('Error with bounds');
xlabel('Iteration'); ylabel('Estimation error');
```

While this example is written in MATLAB, implementing a Kalman filter in another language is no more challenging except that we would need to write additional custom code to perform the matrix operations that are built in to MATLAB.

3.6.4 *Improving numeric robustness*

Within the filter, the covariance matrices $\mathbf{\Sigma}^-_{\tilde{x},k}$ and $\mathbf{\Sigma}^+_{\tilde{x},k}$ must remain symmetric and positive definite (all eigenvalues must be strictly positive). It is possible for both conditions to be violated due to roundoff errors in a computer implementation. We wish to find ways to limit or to eliminate these problems.

Covariance matrices can become asymmetric or non-positive-definite only when they are updated; that is, in either the time-update or measurement-update equations of the filter. Searching for the source of the potential problem, we consider first the time-update equation:

$$\mathbf{\Sigma}^-_{\tilde{x},k} = \mathbf{A}\mathbf{\Sigma}^+_{\tilde{x},k-1}\mathbf{A}^T + \mathbf{\Sigma}_{\widetilde{w}}.$$

Because we are adding two positive-definite quantities together, the result must be positive definite. A suitable implementation of the products of the matrices will avoid loss of symmetry in the final result. So, this equation is not the source of the problem.

Consider next the measurement-update equation:

$$\mathbf{\Sigma}^+_{\tilde{x},k} = \mathbf{\Sigma}^-_{\tilde{x},k} - \mathbf{L}_k\mathbf{\Sigma}_{\tilde{y},k}\mathbf{L}_k^T.$$

Theoretically, the result is always positive definite, but due to the subtraction operation it is possible for round-off errors in an implementation to result in a non-positive-definite solution. A better solution is to use the *Joseph-form covariance update*,

$$\mathbf{\Sigma}^+_{\tilde{x},k} = \left[\mathbf{I} - \mathbf{L}_k\mathbf{C}_k\right]\mathbf{\Sigma}^-_{\tilde{x},k}\left[\mathbf{I} - \mathbf{L}_k\mathbf{C}_k\right]^T + \mathbf{L}_k\mathbf{\Sigma}_{\tilde{v}}\mathbf{L}_k^T. \tag{3.22}$$

Eq. (3.22) may be shown to be equivalent to Eq. (3.21) mathematically via

$$\begin{aligned}
\boldsymbol{\Sigma}_{\tilde{x},k}^{+} &= [\boldsymbol{I} - \boldsymbol{L}_k\boldsymbol{C}_k]\,\boldsymbol{\Sigma}_{\tilde{x},k}^{-}\,[\boldsymbol{I} - \boldsymbol{L}_k\boldsymbol{C}_k]^T + \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{v}}\boldsymbol{L}_k^T \\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{L}_k\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T\boldsymbol{L}_k^T + \boldsymbol{L}_k\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T\boldsymbol{L}_k^T + \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{v}}\boldsymbol{L}_k^T \\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{L}_k\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T\boldsymbol{L}_k^T + \boldsymbol{L}_k\left(\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T + \boldsymbol{\Sigma}_{\tilde{v}}\right)\boldsymbol{L}_k^T \\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{L}_k\boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T\boldsymbol{L}_k^T + \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}L^T \\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T + \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}L^T \\
&= \boldsymbol{\Sigma}_{\tilde{x},k}^{-} - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T.
\end{aligned}$$

However, because the subtraction occurs in a term that is effectively squared in the Joseph-form update of Eq. (3.22), this form guarantees a positive definite result.

If we still end up with a nonpositive-definite matrix because of floating-point rounding error, we can replace the erroneous covariance matrix by the nearest symmetric positive semidefinite matrix.[18] Omitting the details, the procedure is

- Calculate singular-value decomposition: $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$.
- Compute $\boldsymbol{H} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{V}^T$.
- Replace $\boldsymbol{\Sigma}$ with $(\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^T + \boldsymbol{H} + \boldsymbol{H}^T)/4$.

[18] Higham, N.J., "Computing a Nearest Symmetric Positive Semidefinite Matrix," *Linear Algebra and its Applications*, 103, 1998, pp. 103–118.

There are still numerous improvements that may be made. We can reduce computational requirements when there are multiple outputs by using sequential updating, increase precision of numeric accuracy using square-root Kalman filtering, and more. There are also nuances in how carefully the voltage and current-sensor measurements must be synchronized. It is beyond the scope of this book to investigate all these details. A good textbook on Kalman filtering will go into these topics in depth.[19]

[19] One example is: Simon, D., *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*, Wiley Interscience, 2006, but there are many others.

3.6.5 *Measurement validation gating*

Sometimes the systems for which we would like a state estimate have sensors with intermittent faults. We would like to detect faulty measurements and discard them (the time-update steps of the Kalman filter are still implemented, but the measurement-update steps are skipped).

The Kalman filter provides an elegant theoretical means to accomplish this goal. Note that the measurement covariance matrix is computed as part of Kalman-filter step 2a as

$$\boldsymbol{\Sigma}_{\tilde{y},k} = \boldsymbol{C}_k\boldsymbol{\Sigma}_{\tilde{x},k}^{-}\boldsymbol{C}_k^T + \boldsymbol{\Sigma}_{\tilde{v}}.$$

The measurement prediction itself is also computed in step 1c as

$\hat{\boldsymbol{y}}_k = \boldsymbol{C}_k\hat{\boldsymbol{x}}_k^- + \boldsymbol{D}_k\boldsymbol{u}_k$, and the innovation is computed in step 2b as $\tilde{\boldsymbol{y}}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_k$.

Consider first the common situation where our system has a scalar output. We know that $\sigma_{\tilde{y},k} = \sqrt{\boldsymbol{\Sigma}_{\tilde{y},k}}$. So, if the absolute value of $\tilde{\boldsymbol{y}}_k$ is deemed to be significantly greater than $\sigma_{\tilde{y},k}$, then either our state estimate is very poor or we have a voltage-sensor fault. In response, if we assume a bad measurement, then we can skip the measurement-update steps. Alternately, if a number of measurements are discarded in a short time interval, it may be that the sensor has truly failed, or that the state estimate and its covariance have somehow gotten "lost." If the latter is true, we can help the Kalman filter reacquire by artificially increasing the covariance estimate by multiplying $\boldsymbol{\Sigma}_{\tilde{x},k}^+$ by a positive constant. Both of these strategies are implemented in practice to aid robustness of a real implementation.

If the model being used by the Kalman filter has more than one output (as we will see in Chap. 4), then we cannot simply compare the innovation vector directly to the covariance matrix. We can take a somewhat different sequence of steps, which we explore here quickly.

First, for a multi-output model, we define $\boldsymbol{\xi}_k = \boldsymbol{M}_k\tilde{\boldsymbol{y}}_k$, for some known time-varying matrix $\boldsymbol{M}_k$ yet to be determined. The mean of $\boldsymbol{\xi}_k$ is

$$\mathbb{E}[\boldsymbol{\xi}_k] = \mathbb{E}[\boldsymbol{M}_k\tilde{\boldsymbol{y}}_k] = \boldsymbol{M}_k\mathbb{E}[\tilde{\boldsymbol{y}}_k] = 0.$$

The covariance of $\boldsymbol{\xi}_k$ is

$$\boldsymbol{\Sigma}_{\tilde{\xi},k} = \mathbb{E}[\boldsymbol{M}_k\tilde{\boldsymbol{y}}_k\tilde{\boldsymbol{y}}_k^T\boldsymbol{M}_k^T] = \boldsymbol{M}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{M}_k^T.$$

Further, we know that $\boldsymbol{\xi}_k$ is a Gaussian random vector, since it is a linear combination of Gaussian RVs.

Second, if we choose $\boldsymbol{M}_k$ such that $\boldsymbol{M}_k^T\boldsymbol{M}_k = \boldsymbol{\Sigma}_{\tilde{y},k}^{-1}$, then $\boldsymbol{M}_k$ is the upper-triangular Cholesky factor of $\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}$. We also have $\boldsymbol{\xi}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ because

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{\xi},k} &= \boldsymbol{M}_k\left(\boldsymbol{M}_k^T\boldsymbol{M}_k\right)^{-1}\boldsymbol{M}_k^T\\ &= \boldsymbol{M}_k\boldsymbol{M}_k^{-1}\boldsymbol{M}_k^{-T}\boldsymbol{M}_k^T\\ &= \boldsymbol{I}.\end{aligned}$$

Third, if we further compute the *normalized estimation error squared*

$$e_k^2 = \boldsymbol{\xi}_k^T\boldsymbol{\xi}_k = \tilde{\boldsymbol{y}}_k^T\boldsymbol{\Sigma}_{\tilde{y},k}^{-1}\tilde{\boldsymbol{y}}_k,$$

then e_k^2 is the sum of squares of independent $\mathcal{N}(0,1)$ RVs. By definition, then, e_k^2 is a *chi-square random variable with m degrees of freedom*, where m is the dimension of $\tilde{\boldsymbol{y}}_k$.[20] Since it is a sum of squares, it is never negative; it is also asymmetric about its mean value.

[20] Notice that we can compute e_k^2 knowing only the innovation and the covariance of the measurement, which are already computed by the Kalman-filter steps. We needed to introduce $\boldsymbol{\xi}_k$ and $\boldsymbol{M}_k$ into this derivation only to show that e_k^2 is a chi-square random variable. We do not need to compute either $\boldsymbol{\xi}_k$ or $\boldsymbol{M}_k$ in practice.

If e_k^2 has an unusually high value, then the sensor measurement is very unlikely. So, we might discard the measurement if e_k^2 is above some threshold. But, what threshold to use? Using theory from the field of statistics, we should keep the measurement if it is within the high-confidence interval of the chi-squared pdf and discard it otherwise. The confidence interval is determined by integrating the pdf until a desired level of probability is achieved.

The pdf of a chi-square RV X having m degrees of freedom is

$$f_X(x) = \frac{1}{2^{m/2}\Gamma(m/2)} x^{(m/2-1)} e^{-m/2},$$

which is difficult to compute, but we never need to evaluate it in real time. Instead, we rely on using a limiting critical value precomputed from the distribution. For example, for $1-\alpha$ confidence of a valid measurement, we want $1-\alpha$ area below the critical point χ_U^2 and α area above this point. Fig. 3.16 draws the chi-squared pdf for three degrees of freedom and shades the $1-\alpha$ probability region for $\alpha = 0.05$. Thus, if we compute a value of e_k^2 that is less than 7.8147, we keep the measurement because we are 95 % confident that it is valid; if e_k^2 is above 7.8147, we discard the measurement because we do not have 95 % confidence that it is valid.

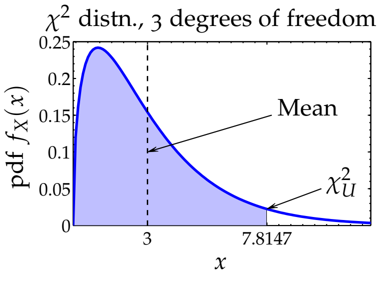


Figure 3.16: Confidence region for a valid measurement when model has three outputs

We find χ_U^2 by solving for the point where the inverse cumulative distribution function is equal to $1-\alpha$. In MATLAB, using the statistics toolbox, we can write

```
X2U = chi2inv(1-0.05,3) % Upper critical value X2U = 7.8147
```

Note that the value for χ_U^2 needs to be computed once only, offline. It is based on the number of measurements in the output vector and the desired confidence level $1-\alpha$ only, neither of which changes over time. It does not need to be recalculated as the Kalman filter executes. For hand calculations a χ^2-table is available on p. 165.

3.7 *The extended Kalman filter*

3.7.1 *Deriving the six steps of extended Kalman filter*

The Kalman filter is the optimal MMSE state estimator for *linear* systems when all noises are white and Gaussian. However, the ESC cell model is *nonlinear*, so the standard Kalman-filter recursion does not apply directly.

The generic sequential-probabilistic-inference solution is still valid for nonlinear systems described via Eqs. (3.6) and (3.7) if all random signals are Gaussian, but the expectation operations in Table 3.1 cannot be computed exactly. Instead, one of three approximation strategies is usually implemented in a state estimator for a nonlinear system:

- *The extended Kalman filter (EKF)*: This method performs *analytic linearization* of the model at each point in time. There are some problems with this approach, but it is still popular and can work well if the system nonlinearities are mild.
- *The sigma-point* (also known as the *unscented*) *Kalman filter (SPKF/UKF)*: This method performs *statistical/empirical linearization* of the model at each point in time. Its results are often much better than EKF, at same computational complexity.[21] It tends to give reasonable estimates even if the nonlinearities are significant.
- *Particle filters*: This approach is the most accurate, but often requires thousands of times more computation than either EKF or SPKF. It does not assume Gaussian distributions in general, and uses Monte Carlo integration techniques to find probabilities, expectations, and uncertainties. Particle filters are required if the system has severe nonlinearities, or when the probability distributions are multimodal (i.e., when the pdfs have multiple peaks, which might correspond to a system where very different distinct state trajectories can result in the same output measurements).

[21] For a humorous story explaining "unscented" in the UKF name, see `http://www.ieeeghn.org/wiki/index.php/First-Hand:The_Unscented_Transform`.

In this chapter, we present the EKF and SPKF. Particle filters are beyond the scope of this book.

THE EKF MAKES TWO SIMPLIFYING ASSUMPTIONS when adapting the general sequential-probabilistic-inference equations to a nonlinear system:

1. When computing estimates of the output of a nonlinear function, EKF assumes that the expected value of a nonlinear function of the unknown state is equal to the same nonlinear function evaluated at the expected value of the state. That is, it approximates

$$\mathbb{E}[\mathbf{fn}(x)] \approx \mathbf{fn}(\mathbb{E}[x]).$$

   This is not true in general. In fact, it is strictly true only when $\mathbf{fn}(x)$ is linear. This is one reason that the EKF works best for systems having only mild nonlinearities.

2. When computing covariance estimates, EKF uses a truncated Taylor-series expansion to linearize the system equations around the present operating point. Higher order terms from the expansion are discarded. This is the second reason why EKF works best for systems having only mild nonlinearities.

Here, we will show how to apply these approximations and assumptions to derive the EKF equations from the six general steps.

EKF step 1a: State-prediction time update.

Starting with Eq. (3.16) and using EKF assumption 1, the state prediction step is approximated as

$$\begin{aligned}\hat{\boldsymbol{x}}_k^- &= \mathbb{E}[\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \mid \mathbb{Y}_{k-1}] \\ &\approx \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1}),\end{aligned}$$

where $\bar{\boldsymbol{w}}_{k-1} = \mathbb{E}[\boldsymbol{w}_{k-1}]$. Often, $\bar{\boldsymbol{w}}_{k-1} = 0$. That is, we approximate the expected value of the new state by assuming that it is reasonable to propagate $\hat{\boldsymbol{x}}_{k-1}^+$ and $\bar{\boldsymbol{w}}_{k-1}$ through the state equation.

EKF step 1b: Error-covariance time update.

The covariance-prediction step is accomplished by first making an approximation for $\tilde{\boldsymbol{x}}_k^-$:

$$\begin{aligned}\tilde{\boldsymbol{x}}_k^- &= \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^- \\ &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) - \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1}).\end{aligned} \tag{3.23}$$

By EKF assumption 2, the first term is expanded as a truncated Taylor series around the prior operating condition, which is defined by the set of values $p_{k-1} = \{\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1}\}$:

$$\begin{aligned}\boldsymbol{x}_k \approx \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1}) &+ \underbrace{\left.\frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1})}{\mathrm{d}\boldsymbol{x}_{k-1}}\right|_{p_{k-1}}}_{\text{Defined as } \hat{\boldsymbol{A}}_{k-1}} (\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1}^+) \\ &+ \underbrace{\left.\frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1})}{\mathrm{d}\boldsymbol{w}_{k-1}}\right|_{p_{k-1}}}_{\text{Defined as } \hat{\boldsymbol{B}}_{k-1}} (\boldsymbol{w}_{k-1} - \bar{\boldsymbol{w}}_{k-1}).\end{aligned} \tag{3.24}$$

This gives $\tilde{\boldsymbol{x}}_k^- \approx \hat{\boldsymbol{A}}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \hat{\boldsymbol{B}}_{k-1}\widetilde{\boldsymbol{w}}_{k-1}$ when we substitute into Eq. (3.23).

We now substitute this result for $\tilde{\boldsymbol{x}}_k^-$ into Eq. (3.17) to find the prediction-error covariance

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{x},k}^- &= \mathbb{E}[(\tilde{\boldsymbol{x}}_k^-)(\tilde{\boldsymbol{x}}_k^-)^T] \\ &\approx \mathbb{E}[(\hat{\boldsymbol{A}}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \hat{\boldsymbol{B}}_{k-1}\widetilde{\boldsymbol{w}}_{k-1})(\hat{\boldsymbol{A}}_{k-1}\tilde{\boldsymbol{x}}_{k-1}^+ + \hat{\boldsymbol{B}}_{k-1}\widetilde{\boldsymbol{w}}_{k-1})^T] \\ &= \hat{\boldsymbol{A}}_{k-1}\mathbb{E}[(\tilde{\boldsymbol{x}}_{k-1}^+)(\tilde{\boldsymbol{x}}_{k-1}^+)^T]\hat{\boldsymbol{A}}_{k-1}^T + \hat{\boldsymbol{B}}_{k-1}\mathbb{E}[\widetilde{\boldsymbol{w}}_{k-1}]\mathbb{E}[(\tilde{\boldsymbol{x}}_{k-1}^+)^T]\hat{\boldsymbol{A}}_{k-1}^T \\ &\qquad + \hat{\boldsymbol{A}}_{k-1}\mathbb{E}[\tilde{\boldsymbol{x}}_{k-1}^+]\mathbb{E}[\widetilde{\boldsymbol{w}}_{k-1}^T]\hat{\boldsymbol{B}}_{k-1}^T + \hat{\boldsymbol{B}}_{k-1}\mathbb{E}[\widetilde{\boldsymbol{w}}_{k-1}\widetilde{\boldsymbol{w}}_{k-1}^T]\hat{\boldsymbol{B}}_{k-1}^T \\ &= \hat{\boldsymbol{A}}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\hat{\boldsymbol{A}}_{k-1}^T + \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\Sigma}_{\widetilde{w}}\hat{\boldsymbol{B}}_{k-1}^T,\end{aligned}$$

where we have made use of the fact that the state prediction is uncorrelated with the process noise at the same time index, and that the state-prediction error is zero mean.

To implement this expression, we will need to be able to evaluate $\hat{\boldsymbol{A}}_{k-1}$ and $\hat{\boldsymbol{B}}_{k-1}$ in Eq. (3.24). Both of these are matrix functions of time, which can be determined analytically from the model's state equation. For example, the $\hat{A}_k$ matrix has the form

$$\hat{A}_k = \begin{bmatrix} \frac{df_1(x_k,u_k,w_k)}{dx_{k,1}} & \frac{df_1(x_k,u_k,w_k)}{dx_{k,2}} & \cdots & \frac{df_1(x_k,u_k,w_k)}{dx_{k,n}} \\ \frac{df_2(x_k,u_k,w_k)}{dx_{k,1}} & \frac{df_2(x_k,u_k,w_k)}{dx_{k,2}} & \cdots & \frac{df_2(x_k,u_k,w_k)}{dx_{k,n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{df_n(x_k,u_k,w_k)}{dx_{k,1}} & \frac{df_n(x_k,u_k,w_k)}{dx_{k,2}} & & \frac{df_n(x_k,u_k,w_k)}{dx_{k,n}} \end{bmatrix}_{p_k},$$

where $x_{k,i}$ is the ith member of the n-vector state x_k, and $f_j(\cdot)$ is the jth output of the state equation.

We need to take care when considering the total differentials in Eq. (3.24). These are not necessarily the same as partial differentials, which are more familiar.[22] To evaluate these expressions, we apply the chain rule of total differentials to our model

$$\begin{aligned} df(x_{k-1},u_{k-1},w_{k-1}) = {} & \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial x_{k-1}} dx_{k-1} \\ & + \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial u_{k-1}} du_{k-1} \\ & + \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial w_{k-1}} dw_{k-1}. \end{aligned}$$

We divide both sides of this equation by dx_{k-1} to find the total differential we seek:

$$\begin{aligned} \frac{df(x_{k-1},u_{k-1},w_{k-1})}{dx_{k-1}} = {} & \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial x_{k-1}} \\ & + \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial u_{k-1}} \underbrace{\frac{du_{k-1}}{dx_{k-1}}}_{0} \\ & + \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial w_{k-1}} \underbrace{\frac{dw_{k-1}}{dx_{k-1}}}_{0} \\ = {} & \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial x_{k-1}}. \end{aligned}$$

The terms du_{k-1}/dx_{k-1} and dw_{k-1}/dx_{k-1} are zero since we can reasonably assume that neither the present deterministic input nor the present process noise are functions of the present system state. So, in this case, the total differential is equal to the partial differential. Similarly, we can find

$$\frac{df(x_{k-1},u_{k-1},w_{k-1})}{dw_{k-1}} = \frac{\partial f(x_{k-1},u_{k-1},w_{k-1})}{\partial w_{k-1}}.$$

While the distinction between the total differential and the partial differential turns out not to be noteworthy at this point, we will

[22] The partial differential $\partial f(x,u,w)/\partial x$ states how much $f(x,u,w)$ changes when there is an infinitesimal change to x, and when all other inputs are kept constant. The total differential $df(x,u,w)/dx$ states how much $f(x,u,w)$ changes when there is an infinitesimal change to x, but when the other inputs are allowed to vary due to the change in x *if they are functions of* x. If the other inputs are not functions of x, then the partial and total differentials are the same, as we will see here. However, if they are functions of x, then the two types of differential give different results.

find that it is essential in Chap. 4 when we look at model parameter estimation using EKFs.

EKF step 1c: Predict system output $\boldsymbol{y}_k$.

Starting with Eq. (3.18) and using EKF assumption 1, the system output is approximated by

$$\begin{aligned}\hat{\boldsymbol{y}}_k &= \mathbb{E}[\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \mid \mathbb{Y}_{k-1}] \\ &\approx \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \bar{\boldsymbol{v}}_k),\end{aligned}$$

where $\bar{\boldsymbol{v}}_k = \mathbb{E}[\boldsymbol{v}_k]$. Often, $\bar{\boldsymbol{v}}_k = 0$. That is, we approximate the expected value of the output by assuming that it is reasonable simply to propagate the state prediction $\hat{\boldsymbol{x}}_k^-$ and the mean sensor noise $\bar{\boldsymbol{v}}_k$ through the output equation.

EKF step 2a: Estimator gain matrix $\boldsymbol{L}_k$.

The output prediction error can be written as

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) - \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \bar{\boldsymbol{v}}_k).$$

To compute the estimator gain matrix using Eq. (3.19), we will need to be able to compute covariance matrices that are based on this prediction error. Similar to the procedure used in step 1b, we invoke EKF assumption 2 and approximate the error using a truncated Taylor-series expansion of $\boldsymbol{y}_k$ around the setpoint $q_k = \{\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \bar{\boldsymbol{v}}_k\}$:

$$\begin{aligned}\boldsymbol{y}_k &\approx \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \bar{\boldsymbol{v}}_k) + \underbrace{\left.\frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{x}_k}\right|_{q_k}}_{\text{Defined as } \hat{\boldsymbol{C}}_k}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-) \\ &\quad + \underbrace{\left.\frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{v}_k}\right|_{q_k}}_{\text{Defined as } \hat{\boldsymbol{D}}_k}(\boldsymbol{v}_k - \bar{\boldsymbol{v}}_k) \\ \tilde{\boldsymbol{y}}_k &\approx \hat{\boldsymbol{C}}_k \tilde{\boldsymbol{x}}_k^- + \hat{\boldsymbol{D}}_k \tilde{\boldsymbol{v}}_k.\end{aligned}$$

Note, much like we saw in EKF step 1b, we can show that the total derivatives required to compute $\hat{\boldsymbol{C}}_k$ and $\hat{\boldsymbol{D}}_k$ are equal to the partial derivatives:

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{x}_k} &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\partial \boldsymbol{x}_k} \\ \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{v}_k} &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)}{\partial \boldsymbol{v}_k}.\end{aligned}$$

From this, we can compute such necessary quantities as

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{y},k} &\approx \hat{\boldsymbol{C}}_k \boldsymbol{\Sigma}_{\tilde{x},k}^- \hat{\boldsymbol{C}}_k^T + \hat{\boldsymbol{D}}_k \boldsymbol{\Sigma}_{\tilde{v}} \hat{\boldsymbol{D}}_k^T, \\ \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^- &\approx \mathbb{E}[(\tilde{\boldsymbol{x}}_k^-)(\hat{\boldsymbol{C}}_k \tilde{\boldsymbol{x}}_k^- + \hat{\boldsymbol{D}}_k \tilde{\boldsymbol{v}}_k)^T] \\ &= \boldsymbol{\Sigma}_{\tilde{x},k}^- \hat{\boldsymbol{C}}_k^T.\end{aligned}$$

These terms may be combined to get the Kalman gain

$$\boldsymbol{L}_k = \boldsymbol{\Sigma}^-_{\tilde{\boldsymbol{x}},k}\hat{\boldsymbol{C}}_k^T \left[\hat{\boldsymbol{C}}_k\boldsymbol{\Sigma}^-_{\tilde{\boldsymbol{x}},k}\hat{\boldsymbol{C}}_k^T + \hat{\boldsymbol{D}}_k\boldsymbol{\Sigma}_{\tilde{\boldsymbol{v}}}\hat{\boldsymbol{D}}_k^T\right]^{-1}.$$

EKF step 2b: State-estimate measurement update.

This step computes the state estimate by updating the state prediction using the estimator gain and the innovation $\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k$. Eq. (3.20) is implemented with no changes

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k).$$

EKF step 2c: Error-covariance measurement update.

Finally, the updated covariance is computed using Eq. (3.21) as

$$\boldsymbol{\Sigma}^+_{\tilde{\boldsymbol{x}},k} = \boldsymbol{\Sigma}^-_{\tilde{\boldsymbol{x}},k} - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}},k}\boldsymbol{L}_k^T.$$

The EKF steps are summarized in the Appendix on p. 163.

3.7.2 *An EKF example, with code*

Before we implement the EKF on a battery-cell model to predict battery state, we introduce a simpler example to show an EKF implementation. We consider a nonlinear system having a scalar state and output that are described by the following dynamics:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, w_k) = \sqrt{5 + x_k} + w_k \\ y_k &= h(x_k, u_k, v_k) = x_k^3 + v_k, \end{aligned}$$

where $\Sigma_{\tilde{w}} = 1$ and $\Sigma_{\tilde{v}} = 2$.

To implement EKF, we must first determine $\hat{A}_k$, $\hat{B}_k$, $\hat{C}_k$, and $\hat{D}_k$.

$$\hat{A}_k = \left.\frac{\partial f(x_k, u_k, w_k)}{\partial x_k}\right|_{x_k = \hat{x}_k^+} = \left.\frac{\partial\left(\sqrt{5 + x_k} + w_k\right)}{\partial x_k}\right|_{x_k = \hat{x}_k^+} = \frac{1}{2\sqrt{5 + \hat{x}_k^+}}$$

$$\hat{B}_k = \left.\frac{\partial f(x_k, u_k, w_k)}{\partial w_k}\right|_{w_k = \bar{w}_k} = \left.\frac{\partial\left(\sqrt{5 + x_k} + w_k\right)}{\partial w_k}\right|_{w_k = \bar{w}_k} = 1$$

$$\hat{C}_k = \left.\frac{\partial h(x_k, u_k, v_k)}{\partial x_k}\right|_{x_k = \hat{x}_k^-} = \left.\frac{\partial\left(x_k^3 + v_k\right)}{\partial x_k}\right|_{x_k = \hat{x}_k^-} = 3(\hat{x}_k^-)^2$$

$$\hat{D}_k = \left.\frac{\partial h(x_k, u_k, v_k)}{\partial v_k}\right|_{v_k = \bar{v}_k} = \left.\frac{\partial\left(x_k^3 + v_k\right)}{\partial v_k}\right|_{v_k = \bar{v}_k} = 1.$$

The following is some sample code to implement an EKF. It is intentionally written in a very similar way to the linear Kalman-filter example of Sect. 3.6.3 to show the commonality of the two approaches. First, we initialize the true and estimated states, the initial state-estimate uncertainty, and reserve some storage:

```
% Initialize simulation variables
SigmaW = 1; % Process noise covariance
SigmaV = 2; % Sensor noise covariance
maxIter = 40;

xtrue = 2 + randn(1); % Initialize true system initial state
xhat = 2;             % Initialize Kalman filter initial estimate
SigmaX = 1;           % Initialize Kalman filter covariance
u = 0;                % Unknown initial driving input: assume zero

% Reserve storage for variables we might want to plot/evaluate
xstore = zeros(maxIter+1,length(xtrue)); xstore(1,:) = xtrue;
xhatstore = zeros(maxIter,length(xhat));
SigmaXstore = zeros(maxIter,length(xhat)^2);
```

In this example, the true state is initialized to a random value with mean 2 and variance 1. Therefore, the state estimate is initialized to 2, and the state covariance is initialized to 1.

Next, we enter the main simulation loop. To use this code for a different system, steps 1a and 1c must be modified with the new system's $\hat{A}_k$, $\hat{B}_k$, $\hat{C}_k$, and $\hat{D}_k$ matrices, and with the correct $f(\cdot)$ and $h(\cdot)$ functions.

```
for k = 1:maxIter,
  % EKF Step 1a: State-prediction time update
  % Note: You need to insert a computation of Ahat and Bhat, and
  % your system's specific f(...) equation here
  % For this example, x(k+1) = sqrt(5+x(k)) + w(k)
  Ahat = 0.5/sqrt(5+xhat); Bhat = 1;
  xhat = sqrt(5+xhat);

  % EKF Step 1b: Error-covariance time update
  SigmaX = Ahat*SigmaX*Ahat' + Bhat*SigmaW*Bhat';

  % [Implied operation of system in background, with
  % input signal u, and output signal y]
  w = chol(SigmaW)'*randn(1);
  v = chol(SigmaV)'*randn(1);
  ytrue = xtrue^3 + v;  % y is based on present x and u
  xtrue = sqrt(5+xtrue) + w;  % future x is based on present u

  % EKF Step 1c: Estimate system output y(k)
  % Note: You need to insert your system's Chat, Dhat, and h(...)
  % equation here
  % For this example, y(k) = x(k)^3
  Chat = 3*xhat^2; Dhat = 1;
  yhat = xhat^3;

  % EKF Step 2a: Compute Kalman gain matrix L(k)
  SigmaY = Chat*SigmaX*Chat' + Dhat*SigmaV*Dhat';
  L = SigmaX*Chat'/SigmaY;

  % EKF Step 2b: State-estimate measurement update
  xhat = xhat + L*(ytrue - yhat);
  xhat = max(-5,xhat); % don't get square root of negative xhat!

  % EKF Step 2c: Error-covariance measurement update
  SigmaX = SigmaX - L*SigmaY*L';
  [~,S,V] = svd(SigmaX);
  HH = V*S*V';
  SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help to keep robust

  % [Store information for evaluation/plotting purposes]
  xstore(k+1,:) = xtrue; xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = (SigmaX(:))';
end
```

Notice that extra precautions are taken in steps 2b and 2c to keep the operation of the filter robust. In step 2c, we use the method from Sect. 3.6.4 to ensure that the covariance matrices remain symmetric and positive semidefinite. In step 2b, we recognize that x_k cannot be less than -5 or else the square-root operation executed by step 1a in the next filter iteration will give an imaginary result. So, we use this side knowledge to enforce the condition that the state estimate be larger than -5 at all times.[23]

[23] Real-world implementations of Kalman filters almost always have some modifications to the generic six steps to make them work better in an environment where the assumptions made when deriving the filter equations are violated. This is why the six steps are derived in this book instead of simply listed. The algorithm designer must know where the equations come from and what they mean in order to modify them or augment them to work in a practical application.

Results are plotted using the same code as in Sect. 3.6.3. The plots will be different every time the EKF is executed due to the random initial state and noise inputs. One example is shown in Fig. 3.17. Normally, we do not know the true system state, but because the true system is being cosimulated inside of this code, we are able to plot that information in this figure. We see that the state estimate tracks the true state quite well. However, the error bounds are unrealistically tight, as is shown in the bottom figure. The error should remain within the 3σ bounds 95 % of the time; here, we see that it is very often outside of the bounds. This is a common outcome when using an EKF for state estimation for a system whose nonlinearities are at least moderately high. This is a problem, as the error bounds cannot be trusted. We will see in Sect. 3.10 that the SPKF often does a better job of computing error bounds.

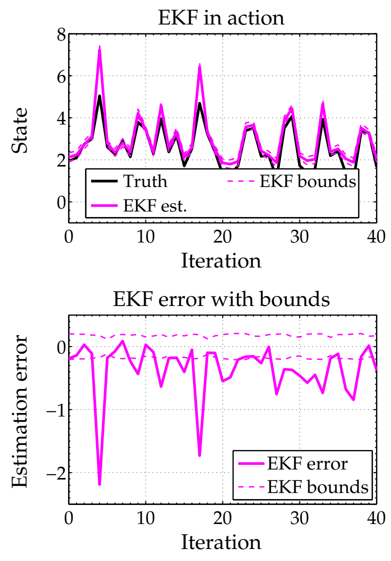


Figure 3.17: Sample EKF estimation results for simple problem.

3.8 *Implementing an EKF using the ESC cell model*

3.8.1 *Computing the EKF matrices*

We are now ready to apply the generic EKF state-estimation procedure to the particular problem of battery-cell state estimation using the ESC cell model. To do so, we must be able to compute the $\hat{A}_k$, $\hat{B}_k$, $\hat{C}_k$, and $\hat{D}_k$ matrices needed by the EKF. We will proceed by first examining the components of the state equation to find $\hat{A}_k$ and $\hat{B}_k$.

Suppose that the process noise models current-sensor measurement error. That is, suppose that the true cell current is $i_k + w_k$, but that we measure i_k only. Also assume that we can simplify the model with coulombic efficiency $\eta_k = 1$, and allow the adaptivity of the EKF to handle the small error introduced by this assumption.

Then, the SOC equation can be written as

$$z_{k+1} = z_k - \frac{\Delta t}{Q}\left(i_k + w_k\right).$$

The two derivatives that we need for this term are

$$\left.\frac{\partial z_{k+1}}{\partial z_k}\right|_{z_k=\hat{z}_k^+} = 1, \qquad \text{and} \qquad \left.\frac{\partial z_{k+1}}{\partial w_k}\right|_{w_k=\bar{w}} = -\frac{\Delta t}{Q},$$

remembering that Q is measured in ampere-seconds.

If $\tau_j = \exp(-\Delta t/(R_j C_j))$, then the resistor-currents state equation can be written as

$$\boldsymbol{i}_{R,k+1} = \underbrace{\begin{bmatrix} \tau_1 & 0 & \cdots \\ 0 & \tau_2 & \\ \vdots & & \ddots \end{bmatrix}}_{\boldsymbol{A}_{RC}} \boldsymbol{i}_{R,k} + \underbrace{\begin{bmatrix} 1-\tau_1 \\ 1-\tau_2 \\ \vdots \end{bmatrix}}_{\boldsymbol{B}_{RC}} \left(i_k + w_k\right).$$

The two derivatives can be found to be

$$\left.\frac{\partial \boldsymbol{i}_{R,k+1}}{\partial \boldsymbol{i}_{R,k}}\right|_{\boldsymbol{i}_{R,k}=\hat{\boldsymbol{i}}_{R,k}^+} = \boldsymbol{A}_{RC}, \qquad \text{and} \qquad \left.\frac{\partial \boldsymbol{i}_{R,k+1}}{\partial w_k}\right|_{w_k=\bar{w}} = \boldsymbol{B}_{RC}.$$

If we define $A_{H,k} = \exp\left(-\left|\frac{(i_k+w_k)\gamma\Delta t}{Q}\right|\right)$, then hysteresis state equation can be written as

$$h_{k+1} = A_{H,k} h_k + \left(A_{H,k} - 1\right)\operatorname{sgn}\left(i_k + w_k\right).$$

Taking the partial derivative with respect to the state and evaluating it at the setpoint p_k (noting that $w_k = \bar{w}$ is a member of the setpoint),

$$\left.\frac{\partial h_{k+1}}{\partial h_k}\right|_{\substack{h_k=\hat{h}_k^+ \\ w_k=\bar{w}}} = \exp\left(-\left|\frac{\left(i_k + \bar{w}_k\right)\gamma\Delta t}{Q}\right|\right) = \bar{A}_{H,k}.$$

Next, we must find $\partial h_{k+1}/\partial w_k$. However, the absolute-value and sign functions are not differentiable at $i_k + w_k = 0$. Ignoring this detail for now:

- If we assume that $i_k + w_k > 0$, then

$$\frac{\partial h_{k+1}}{\partial w_k} = -\left|\frac{\gamma\Delta t}{Q}\right| \exp\left(-\left|\frac{\gamma\Delta t}{Q}\right| \left|\left(i_k + w_k\right)\right|\right)\left(1 + h_k\right).$$

- If we assume that $i_k + w_k < 0$, then

$$\frac{\partial h_{k+1}}{\partial w_k} = -\left|\frac{\gamma\Delta t}{Q}\right| \exp\left(-\left|\frac{\gamma\Delta t}{Q}\right| \left|\left(i_k + w_k\right)\right|\right)\left(1 - h_k\right).$$

Overall, evaluating at the Taylor-series linearization setpoint we will assume that the following generalization is reasonable for all $i_k + w_k$

$$\left.\frac{\partial h_{k+1}}{\partial w_k}\right|_{\substack{h_k=\hat{h}_k^+ \\ w_k=\bar{w}}} = -\left|\frac{\gamma\Delta t}{Q}\right| \bar{A}_{H,k}\left(1 + \operatorname{sgn}(i_k + \bar{w})\hat{h}_k^+\right).$$

12:07:34.

The zero-state hysteresis equation is defined as

$$s_{k+1} = \begin{cases} \operatorname{sgn}(i_k + w_k), & |i_k + w_k| > 0, \\ s_k, & \text{else.} \end{cases}$$

If we consider $i_k + w_k = 0$ to be a zero-probability event, then

$$\frac{\partial s_{k+1}}{\partial s_k} = 0, \qquad \text{and} \qquad \frac{\partial s_{k+1}}{\partial w_k} = 0.$$

We now look at the components that determine $\hat{\boldsymbol{C}}_k$ and $\hat{\boldsymbol{D}}_k$. The ESC-model output equation is

$$y_k = \mathrm{OCV}(z_k) + M h_k + M_0 s_k - \sum_j R_j i_{R_j,k} - R_0 i_k + v_k.$$

We no longer consider i_k to have w_k noise added to it (this would add correlation between process noise and the overall noise present in the measurement, which violates one assumption made when deriving the Kalman filter). Thus, we have

$$\left.\frac{\partial y_k}{\partial s_k}\right| = M_0, \quad \left.\frac{\partial y_k}{\partial h_k}\right| = M, \quad \left.\frac{\partial y_k}{\partial i_{R_j,k}}\right| = -R_j, \quad \text{and} \quad \left.\frac{\partial y_k}{\partial v_k}\right| = 1.$$

We also require

$$\left.\frac{\partial y_k}{\partial z_k}\right|_{z_k = \hat{z}_k^-} = \left.\frac{\partial \mathrm{OCV}(z_k)}{\partial z_k}\right|_{z_k = \hat{z}_k^-},$$

which can be approximated from OCV data as follows. If `SOC` is a vector of evenly-spaced SOC points with corresponding open-circuit voltage vector `OCV`, the following MATLAB code can approximate this partial derivative.

```
% Find dOCV/dz at SOC = z from {SOC,OCV} data
function dOCVz = dOCVfromSOC(SOC,OCV,z)
  dZ = SOC(2) - SOC(1); % Find spacing of SOC vector
  dUdZ = diff(OCV)/dZ;  % Scaled forward finite difference
  dOCV = ([dUdZ(1) dUdZ] + [dUdZ dUdZ(end)])/2; % Avg of fwd/bkwd diffs
  dOCVz = interp1(SOC,dOCV,z); % Could make more efficient than this...
```

Some sample evaluations of this function for six different lithium-ion OCV relationships are shown in Fig. 3.18. There is some noise in this empirical estimate, which could be filtered, but it's not really necessary to do so.[24]

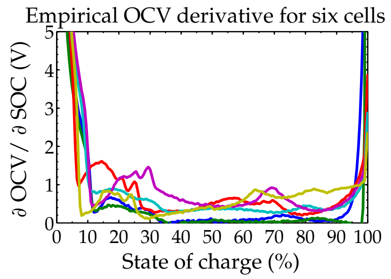


Figure 3.18: Estimates of the variation in OCV as a function of a change in SOC.

[24] Note that a zero-phase filter is required if the empirical derivative relationship must be smoothed. Otherwise, the curve will be shifted along the state of charge axis. Zero-phase filter design and implementation is beyond the scope of this book.

3.8.2 *A refactored implementation of the EKF*

We could choose to implement the EKF for cell state estimation the same way as we did in the example in Sect. 3.17. Instead, we refactor the code to be more representative of a real BMS implementation. The new organization is consistent with the process diagramed in Fig. 3.1.

12:07:34.

There are three main parts to the refactored code. There is an initialization routine (`initEKF.m`), called once at startup. There is an update routine (`iterEKF.m`), which is called every sample interval when new measurements of cell current and cell voltage arrive. Finally, there is the "wrapper" code, which coordinates the entire simulation process. This wrapper code is the main BMS function, and implements the BMS-application program loop.[25]

[25] This code may be downloaded from `http://mocha-java.uccs.edu/BMS2/CH3/ESCEKF.zip`.

WRAPPER CODE: We start by examining the main BMS program loop. The first part of this code loads the ESC cell model file and the dynamic cell-test datafile. In an implementation, we would measure voltage and current as the BMS operated; however, in the simulation we load data that have already been collected from a cell in a laboratory. Vectors of `time`, `current`, and `voltage` are retrieved from the dynamic datafile, and the time variable is modified to ensure that it starts at zero.

```
% Load cell model file into structure "model"
load cellModel

% Load cell-test data. Contains variable "DYNData" of which the field
% "script1" is of interest. It has sub-fields time, current, voltage, soc.
load Cell_DYN_P5 % load dynamic data
T = 5;           % test temperature = 5 degC

time    = DYNData.script1.time(:);   deltat = time(2)-time(1);
time    = time-time(1); % start time at 0
current = DYNData.script1.current(:); % discharge > 0; charge < 0.
voltage = DYNData.script1.voltage(:);
```

A precomputed SOC "truth" vector `soc`, based on coulomb counting with very accurate initialization and using lab-grade current sensing is also loaded. This truth vector would not be available in an implementation; here, it is used to compare against the EKF results when making plots. Initialization continues by reserving storage for some computed results for later plotting. This would also not be necessary in an implementation, because there is no need to store past estimates.

```
soc     = DYNData.script1.soc(:);

% Reserve storage for computed results, for plotting
sochat = zeros(size(soc)); socbound = zeros(size(soc));
```

Next, we proceed with some "key on" initialization functions. We specify values for the initial-state, process-noise and sensor-noise covariances. Then, we invoke a function `initEKF.m`, which initializes the data structure `ekfData` that is used by the EKF to store algorithm data from iteration to iteration. We will discuss `initEKF.m` shortly.

```
% Covariance values
SigmaX0 = diag([1e-6 1e-8 2e-4]); % uncertainty of initial state
```

```
SigmaW = 2e-1; % uncertainty of current sensor, state equation
SigmaV = 2e-1; % uncertainty of voltage sensor, output equation

% Create ekfData structure and initialize variables using first
% voltage measurement and first temperature measurement
ekfData = initEKF(voltage(1),T,SigmaX0,SigmaV,SigmaW,model);
```

With initialization complete, we now enter the main program loop. Because the computations take some time, we use a MATLAB feature to open a "waitbar" window to display progress. Then, at every time interval for which we have data, we extract the present voltage, current, and temperature measurement from the dataset. We then invoke the `iterEKF.m` function to update the state estimate using the EKF (we will discuss this function shortly). The waitbar is periodically updated and is closed when all the data have been processed.

```
% Now, enter loop for remainder of time, where we update the EKF
% once per sample interval
hwait = waitbar(0,'Computing...');
for k = 1:length(voltage),
  vk = voltage(k); % "measure" voltage
  ik = current(k); % "measure" current
  Tk = T;          % "measure" temperature

  % Update SOC (and other model states)
  [sochat(k),socbound(k),ekfData] = iterEKF(vk,ik,Tk,deltat,ekfData);
  % update waitbar periodically, but not too often (slow procedure)
  if mod(k,1000)==0, waitbar(k/length(current),hwait); end;
end
close(hwait);
```

Finally, we plot some results.[26] The first figure shows true SOC, its estimate, and bounds on the estimate. The second figure shows SOC estimation error with bounds. Root-mean-squared SOC estimation error is also computed and printed, as is the percentage of time that the error is outside of bounds. Ideally, both of these values would be zero.

[26] This code would not be in an implementation: it is here only to enable evaluation of the EKF performance on the desktop application.

```
figure(1); clf; plot(time/60,100*sochat,time/60,100*soc); hold on
h = plot([time/60; NaN; time/60],...
         [100*(sochat+socbound); NaN; 100*(sochat-socbound)]);
title('SOC estimation using EKF');
xlabel('Time (min)'); ylabel('SOC (%)');
legend('Estimate','Truth','Bounds'); grid on

fprintf('RMS SOC estimation error = %g%%\n',...
        sqrt(mean((100*(soc-sochat)).^2)));

figure(2); clf; plot(time/60,100*(soc-sochat)); hold on
h = plot([time/60; NaN; time/60],[100*socbound; NaN; -100*socbound]);
title('SOC estimation errors using EKF');
xlabel('Time (min)'); ylabel('SOC error (%)'); ylim([-4 4]);
legend('Estimation error','Bounds'); grid on

ind = find(abs(soc-sochat)>socbound);
fprintf('Percent of time error outside bounds = %g%%\n',...
```

12:07:34.

```
        length(ind)/length(soc)*100);
```

INITIALIZATION CODE: We now examine `initEKF.m`. The purpose of this function is to create the data structure `ekfData`, which is used by the EKF to store relevant constants and variables. At the top of the function, the state vector `xhat` is initialized. We assume that diffusion currents are zero, hysteresis voltage is zero, and we look up initial state of charge based on the OCV relationship and initial voltage `v0`, assuming that the cell is in equilibrium at startup. Since the state vector can be ordered in arbitrary ways, we also define index variables `irInd`, `hkInd`, and `zkInd` as indices into the state vector for diffusion current, hysteresis, and SOC, respectively. All of these values are stored as fields of the `ekfData` output structure.

```
function ekfData = initEKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)

  % Initial state description
  ir0   = 0;                           ekfData.irInd = 1;
  hk0   = 0;                           ekfData.hkInd = 2;
  SOC0  = SOCfromOCVtemp(v0,T0,model); ekfData.zkInd = 3;
  ekfData.xhat  = [ir0 hk0 SOC0]';     % initial state

  % Covariance values
  ekfData.SigmaW = SigmaW;             ekfData.SigmaV = SigmaV;
  ekfData.SigmaX = SigmaX0;            ekfData.SXbump = 5;

  % previous value of current
  ekfData.priorI = 0;
  ekfData.signIk = 0;

  % store model data structure too
  ekfData.model = model;
end
```

Next, we store the known values for initial-state, process-noise, and sensor noise covariance. The `SXbump` field is used with measurement-validation gating as a factor to increase the estimated state's covariance if the filter believes that its estimate is "lost." The (unknown) previous value of cell input current is set to zero, and the sign of that current is also set to zero. Finally, the ESC cell model structure is saved to `ekfData` and the function returns.

ITERATION CODE: Every time cell current and voltage are sensed, the main program loop invokes `iterEKF.m`, which we describe here. First, the cell model is unpacked from the `ekfData` structure, and the cell-model parameter values are computed based on the present value of cell temperature. This must be done every iteration because temperature might be changing. The standard `getParamESC.m` model data accessor function from the ESC toolbox is used.

```
function [zk,zkbnd,ekfData] = iterEKF(vk,ik,Tk,deltat,ekfData)
```

12:07:34.

```
model = ekfData.model;
% Load the cell model parameters for the present operating temp.
Q  = getParamESC('QParam',Tk,model);
G  = getParamESC('GParam',Tk,model);
M  = getParamESC('MParam',Tk,model);
M0 = getParamESC('M0Param',Tk,model);
RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
R  = getParamESC('RParam',Tk,model)';
R0 = getParamESC('R0Param',Tk,model);
eta = getParamESC('etaParam',Tk,model);
if ik<0, ik=ik*eta; end; % adjust current if charging cell
```

Next, EKF constants and variables are extracted from the `ekfData` structure. If the present cell current is larger in magnitude than a C/100 rate, the new sign of current is also computed.

```
% Get data stored in ekfData structure
SigmaX = ekfData.SigmaX;     SigmaW = ekfData.SigmaW;
SigmaV = ekfData.SigmaV;
irInd  = ekfData.irInd;      hkInd  = ekfData.hkInd;
zkInd  = ekfData.zkInd;
xhat   = ekfData.xhat;       nx     = length(xhat);
I = ekfData.priorI;
if abs(ik)>Q/100, ekfData.signIk = sign(ik); end;
signIk = ekfData.signIk;
```

We now begin evaluating the six EKF steps. Step 1a must compute the $\hat{A}_{k-1}$ and $\hat{B}_{k-1}$ matrices first. It also computes a B matrix, which is used when computing the predicted state vector. Note that prior to executing steps 1a through 1c, `xhat` and `SigmaX` refer to $\hat{x}_{k-1}^{+}$ and $\Sigma_{\tilde{x},k-1}^{+}$, respectively. After these steps, `xhat` and `SigmaX` refer to $\hat{x}_{k}^{-}$ and $\Sigma_{\tilde{x},k}^{-}$, respectively.

```
% Step 1a: State-prediction time update
% First, compute Ahat[k-1], Bhat[k-1]
Ah = exp(-abs(I*G*deltat/(3600*Q)));  % hysteresis factor
Bh = -abs(G*deltat/(3600*Q))*Ah*(1+sign(I)*xhat(hkInd));
Ahat = zeros(nx,nx);           Bhat        = zeros(nx,1);
Ahat(zkInd,zkInd) = 1;         Bhat(zkInd) = -deltat/(3600*Q);
Ahat(irInd,irInd) = diag(RC);  Bhat(irInd) = 1-RC(:);
B = [Bhat, 0*Bhat];
Ahat(hkInd,hkInd) = Ah;        Bhat(hkInd) = Bh;
B(hkInd,2) = Ah-1;
% Next, update xhat
xhat = Ahat*xhat + B*[I; sign(I)];

% Step 1b: Error-covariance time update
%          sigmaminus(k) = Ahat(k-1)*sigmaplus(k-1)*Ahat(k-1)' + ...
%                          Bhat(k-1)*sigmawtilde*Bhat(k-1)'
SigmaX = Ahat*SigmaX*Ahat' + Bhat*SigmaW*Bhat';

% Step 1c: Output estimate
yhat = OCVfromSOCtemp(xhat(zkInd),Tk,model) + M0*signIk + ...
       M*xhat(hkInd) - R*xhat(irInd) - R0*ik;
```

Steps 1b and 1c are implemented in a straightforward way, making use of the ESC toolbox function `OCVfromSOCtemp.m` to compute the OCV as a function of SOC and temperature.

In step 2a, we must compute the $\hat{C}_k$ and $\hat{D}_k$ matrices first, then $\Sigma_{\tilde{y},k}$, and finally the Kalman gain L_k. In step 2b, we use measurement validation gating to discard a measurement whose innovation is larger than $10\sigma_{\tilde{y}}$ in magnitude. Limits are put on the hysteresis state to keep it within its required bounds of $h_k \in [-1, 1]$, and limits are also placed on the SOC state to keep it within $z_k \in [-0.05, 1.05]$. Slight undercharge and overcharge are permitted in the estimate. In step 2c, the state of charge covariance is multiplied by the `SXbump` factor if the innovation is larger than $2\sigma_{\tilde{y}}$ in magnitude. Finally, data are stored for the next iteration, and the function returns control to the main program loop. Note that prior to executing step 2, `xhat` and `SigmaX` refer to $\hat{x}_k^-$ and $\Sigma_{\tilde{x},k}^-$; subsequent to step 2, they refer to $\hat{x}_k^+$ and $\Sigma_{\tilde{x},k}^+$, respectively.

```
  % Step 2a: Estimator gain matrix
  Chat = zeros(1,nx);
  Chat(zkInd) = dOCVfromSOCtemp(xhat(zkInd),Tk,model);
  Chat(hkInd) = M;
  Chat(irInd) = -R;
  Dhat = 1;
  SigmaY = Chat*SigmaX*Chat' + Dhat*SigmaV*Dhat';
  L = SigmaX*Chat'/SigmaY;

  % Step 2b: State-estimate measurement update
  r = vk - yhat; % residual.  Use to check for sensor errors...
  if r^2 > 100*SigmaY, L(:)=0.0; end
  xhat = xhat + L*r;
  xhat(hkInd) = min(1,max(-1,xhat(hkInd))); % Help maintain robustness
  xhat(zkInd) = min(1.05,max(-0.05,xhat(zkInd)));

  % Step 2c: Error-covariance measurement update
  SigmaX = SigmaX - L*SigmaY*L';
  % SigmaX-bump code
  if r^2 > 4*SigmaY, % bad voltage estimate by 2 std devs, bump SigmaX
    fprintf('Bumping SigmaX\n');
    SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*ekfData.SXbump;
  end
  [~,S,V] = svd(SigmaX);
  HH = V*S*V';
  SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help maintain robustness

  % Save data in ekfData structure for next time...
  ekfData.priorI = ik;
  ekfData.SigmaX = SigmaX;
  ekfData.xhat = xhat;
  zk = xhat(zkInd);
  zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));
end
```

The function returns the present SOC estimate `zk`, the 3σ bounds on that estimate `zkbnd`, and the updated `ekfData` structure.

12:07:34.

3.8.3 *Example of EKF on ESC model*

Fig. 3.19 shows sample results when executing this code. The dataset that was used as input comprised 16 repetitions of a UDDS drive-cycle profile separated by rest intervals. The cell began in a fully-charged state and completed the test at around 3.3 % SOC.

The top frame shows that SOC tracking by the EKF in an absolute sense is generally quite good. The bottom frame shows state of charge estimation error, and is easier to interpret. For this example, *root-mean-squared (RMS)* SOC estimation error was 1.53 %. The error *should* always be within the bounds; however, in this example, it was outside of bounds 35.9 % of the time. Note that this test was performed at 5 °C where nonlinear hysteresis is a significant factor in the cell voltage. Results for tests using data collected at warmer temperatures tend to yield better estimates, and estimates tend to stay within error bounds a greater fraction of the time. This example is not a worst case, but it is a worse-than-typical case.

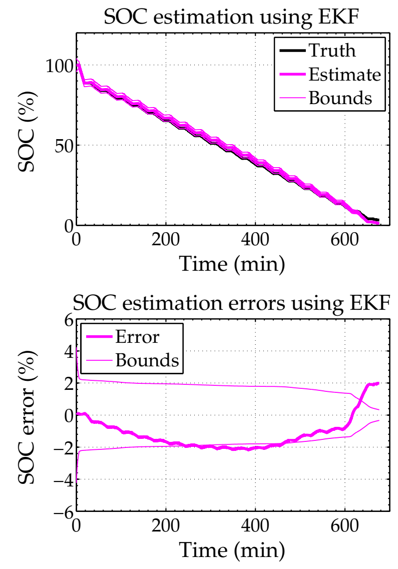


Figure 3.19: Example EKF SOC estimation results.

WITH SOME EFFORT EXPENDED in adjusting the values of $\boldsymbol{\Sigma}^+_{\tilde{x},0}$, $\Sigma_{\widetilde{w}}$, and $\Sigma_{\tilde{v}}$, these results might be improved. If all assumptions of the EKF were met exactly, we could arrive at an expression for $\boldsymbol{\Sigma}^+_{\tilde{x},0}$ based on the initial voltage-reading uncertainty and the uncertainty of the SOC versus OCV curve. We could derive a value for $\Sigma_{\widetilde{w}}$ based on the variance of the current-sensor reading, and for $\Sigma_{\tilde{v}}$ based on the variance of the voltage-sensor reading. However, the EKF assumptions are *not* met exactly. Therefore, such derived quantities are only approximately helpful. They can be used as an initial guess.

Then, how do we compute $\boldsymbol{\Sigma}^+_{\tilde{x},0}$, $\boldsymbol{\Sigma}_{\widetilde{w}}$, and $\boldsymbol{\Sigma}_{\tilde{v}}$ for an application? There is some literature on adaptive versions of EKF, which adjust $\Sigma_{\widetilde{w}}$ and $\Sigma_{\tilde{v}}$ while the filter runs. Our experience with these methods has been mixed. They seem to work well for some applications but not very well for others. Instead, we have usually resorted to trial and error. Many datasets are collected from a cell over the entire anticipated operating range of the application. These include cold temperatures and hot temperatures, data with calibrated current and voltage sensors and data with biased current values and noisy voltages, and so forth. A single set of $\boldsymbol{\Sigma}^+_{\tilde{x},0}$, $\boldsymbol{\Sigma}_{\widetilde{w}}$, and $\boldsymbol{\Sigma}_{\tilde{v}}$ are adapted so that the overall estimation results for all datasets are acceptable. This might be done by hand or in an optimization program loop. Either way, it is a time-consuming task.

3.9 *Problems with EKF, improved with sigma-point methods*

The EKF is the best known and probably the most widely used nonlinear Kalman filter. Although it has some serious flaws, these can be remedied fairly easily by using a sigma-point approach, as we will discuss.

The primary issue is how the EKF propagates the mean and covariance of a random vector through a static nonlinear function to estimate the mean and covariance of the output random vector. The issue is not with the general predict/correct mechanism of sequential probabilistic inference, nor is it fundamentally with propagating a random vector from one time step to another. So, in this section, we focus only on the propagation of these two statistics through a nonlinear function.

Recall that the extended Kalman filter, when computing mean estimates in Steps 1a and 1c, makes the simplification $\mathbb{E}[\mathbf{fn}(x)] \approx \mathbf{fn}(\mathbb{E}[x])$. This is not true in general, and may not be even close to true depending on "how nonlinear" the function $\mathbf{fn}(\cdot)$ is. Also, in EKF Steps 1b and 2a, a Taylor-series expansion is performed as part of the calculation of output-variable covariance. Nonlinear terms are dropped, resulting in a loss of accuracy.

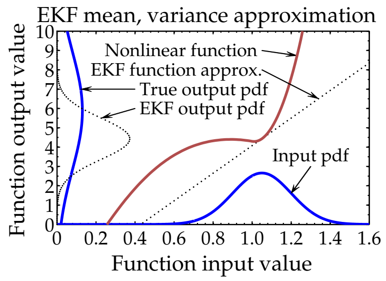


Figure 3.20: EKF approach to computing output mean and covariance.

A simple one-dimensional example illustrates these two effects. Consider the illustration in Fig. 3.20. The horizontal axis represents an input value to a nonlinear function, and the vertical axis represents the output value from the function. An example nonlinear function is drawn as a red line.

If the input to this function were deterministic, we would simply calculate the dependent output variable from this independent input variable using the function formulation. Graphically, we would locate the independent variable on the horizontal axis, draw a line straight up until it intersected with the nonlinear function, and then draw a line straight left until it intersected with the output axis. The point of intersection with the output axis would give us the output dependent-variable value.

In our case, however, the input is a random variable. We don't know its exact value; instead, we know its pdf, which is drawn on the horizontal axis of the figure.[27] EKF estimates the mean of the output random variable by computing the output of the nonlinear function when the input is equal to the mean of the input pdf. In this example, the input random variable has mean of 1.05. This produces an output mean estimate of around 4.4.

[27] With EKF, as with SPKF later, we assume that both the input and output RVs have Gaussian distribution, even though we know this is not exact when propagating distributions through nonlinear functions. In any case, we are attempting to estimate the mean and covariance of the output given only the mean and covariance of the input.

To compute the output variance, the EKF method linearizes the nonlinear function in the neighborhood of the input-pdf mean. This linearization is drawn as a dotted straight line in the figure. The

12:07:34.

input-pdf standard deviation is multiplied by the slope of this line estimate to compute the output-pdf standard deviation. In this example, the input pdf has standard deviation of 0.15 and the linearization has slope of 7.1, giving an output-pdf standard-deviation estimate of 1.06. The EKF estimate of the output pdf—assuming that it is Gaussian with the computed mean of 4.4 and standard deviation of 1.06—is drawn on the vertical axis using a dotted line.

To compute near-exact estimates of the output mean and variance, 100,000 samples are randomly generated from the input pdf, and propagated through the nonlinear function to produce output samples. The mean and standard deviation of those output samples were 5.9 and 3.06, respectively. A Gaussian pdf having those statistics is drawn on the vertical axis as a solid blue line. We notice significant differences between the means and variances: the EKF approach is not producing an accurate estimate of either.

We can improve on mean and covariance propagation through the nonlinear state and output equations using a sigma-point method.

3.9.1 Approximating statistics with sigma points

We now look at a different approach to characterizing the mean and covariance of the output of a nonlinear function. It avoids the two assumptions made by the EKF method and can produce much better estimates.

In the example we have just seen, we noticed that if we randomly generate a very large number of samples from the input pdf, propagate those values through the nonlinear function, and then compute statistics based on the results, we can arrive at good estimates of the mean and variance of the output pdf. The sigma-point approach uses this general idea but is able to reduce the number of samples taken from the input pdf down to a bare minimum. Effectively, the analytic linearization of the EKF is replaced by an efficient empirical or statistical linearization using a small number of function evaluations.

This has several advantages:

1. Derivatives do not need to be computed (which is one of the most error-prone steps when implementing EKF), also implying
2. The original functions do not need to be differentiable, and
3. Better covariance approximations are usually achieved, relative to EKF, allowing for better state estimation and error bounds,
4. All with comparable computational complexity to EKF.

The key to the sigma-point approach is the method by which the samples from the input pdf are selected. If we are content to use many samples, then this can be done randomly. However, for an efficient algorithm we must very carefully choose which samples are used.

We denote the set of samples from the input pdf as $\boldsymbol{\mathcal{X}}$, and call them the *input sigma points* to the nonlinear function. These sigma points will be chosen such that the (possibly weighted) mean and covariance of the points exactly matches the mean $\bar{x}$ and covariance $\boldsymbol{\Sigma}_{\tilde{x}}$ of the input random variable to the nonlinear function.

These points are then individually passed through the nonlinear function, resulting in a transformed set of *output sigma points* $\boldsymbol{\mathcal{Y}}$. The mean $\bar{y}$ and covariance $\boldsymbol{\Sigma}_{\tilde{y}}$ of the output random variable are then approximated by the (possibly weighted) mean and covariance of these transformed sigma points $\boldsymbol{\mathcal{Y}}$.

Note that the sigma points comprise a fixed small number of vectors that are calculated deterministically—not like the randomly generated points of the previous example or like particles used in particle-filter methods. Specifically, if the input random vector x has dimension L, mean $\bar{x}$, and covariance $\boldsymbol{\Sigma}_{\tilde{x}}$, then $p+1 = 2L+1$ sigma points are generated as a set that we write as

$$\boldsymbol{\mathcal{X}} = \left\{\bar{x}, \bar{x} + \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}, \bar{x} - \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}\right\}. \tag{3.25}$$

The notation in Eq. (3.25) is mathematical shorthand and requires some explanation. First, braces $\{\cdot\}$ are used to underscore the fact that $\boldsymbol{\mathcal{X}}$ is a set of vectors. We will find it convenient to store this set in a compact form as a matrix, where every column of the matrix is one of the members of the set; nonetheless, $\boldsymbol{\mathcal{X}}$ is technically a set.

The members of $\boldsymbol{\mathcal{X}}$ are indexed from 0 to p. The zeroth element of $\boldsymbol{\mathcal{X}}$ is the mean $\bar{x}$ of the pdf being modeled. The next L elements of the set are written compactly as $\bar{x} + \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$. In this notation, the matrix square root $\boldsymbol{R} = \sqrt{\boldsymbol{\Sigma}}$ computes a result such that $\boldsymbol{\Sigma} = \boldsymbol{R}\boldsymbol{R}^T$. Usually, the efficient *Cholesky decomposition* is used, resulting in a *lower-triangular* square matrix $\boldsymbol{R}$ of same dimension as $\boldsymbol{\Sigma}_{\tilde{x}}$.[28] In the equation, γ is a weighting constant that can be adjusted to tune the performance of the sigma-point method.

[28] Take care: MATLAB, by default, returns an upper-triangular matrix. The `'lower'` optional argument must be used to arrive at the correct result.

So, $\bar{x}$ is a vector and $\gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$ is a matrix. They are of incompatible dimensions to be added, so the notation "$\bar{x} + \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$" makes no sense per standard linear algebra. Instead, what the notation *means* is that the vector $\bar{x}$ is added to every column of $\gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$ to produce a resulting matrix of the same size as $\boldsymbol{\Sigma}_{\tilde{x}}$. The L columns of this output matrix comprise sigma points 1 through L in $\boldsymbol{\mathcal{X}}$.

Similarly, the final L sigma points of $\boldsymbol{\mathcal{X}}$ are denoted as "$\bar{x} - \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$" which means "subtract the columns of $\gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$ from $\bar{x}$ to make L sigma points." These are the elements in $\boldsymbol{\mathcal{X}}$ indexed from $L+1$ to $2L$.

The weighted mean and covariance of the elements of $\boldsymbol{\mathcal{X}}$ are equal to the original mean and covariance of x for some $\{\gamma, \alpha_i^{(\mathrm{m})}, \alpha_i^{(\mathrm{c})}\}$ if we

12:07:34.

compute

$$\bar{x} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \boldsymbol{\mathcal{X}}_i \qquad \text{and} \qquad \boldsymbol{\Sigma}_{\tilde{x}} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\boldsymbol{\mathcal{X}}_i - \bar{x})(\boldsymbol{\mathcal{X}}_i - \bar{x})^T,$$

where $\boldsymbol{\mathcal{X}}_i$ is the ith vector member of the set $\boldsymbol{\mathcal{X}}$, and both $\alpha_i^{(\mathrm{m})}$ and $\alpha_i^{(\mathrm{c})}$ are real scalars where $\alpha_i^{(\mathrm{m})}$ and $\alpha_i^{(\mathrm{c})}$ must both sum to one. The $\alpha_i^{(\mathrm{m})}$ are weighting constants used when computing the mean and the $\alpha_i^{(\mathrm{c})}$ are weighting constants used when computing the covariance. They are tuning parameters of the sigma-point methods.

The various sigma-point methods differ only in the choices taken for these weighting constants. Table 3.2 lists values used by the two most popular methods, the *unscented Kalman filter (UKF)* and the *central-difference Kalman filter (CDKF)*. The original derivations of these two methods were quite different but the final methods are essentially identical. CDKF has only one tuning parameter h, so implementation is simpler. It also has marginally higher theoretic accuracy when the distributions are indeed Gaussian. However, UKF has more tuning parameters, so can be made to work better in practice when the distributions are not Gaussian.

Method	γ	$\alpha_0^{(\mathrm{m})}$	$\alpha_k^{(\mathrm{m})}$	$\alpha_0^{(\mathrm{c})}$	$\alpha_k^{(\mathrm{c})}$
UKF	$\sqrt{L+\lambda}$	$\frac{\lambda}{L+\lambda}$	$\frac{1}{2(L+\lambda)}$	$\frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta)$	$\frac{1}{2(L+\lambda)}$
CDKF	h	$\frac{h^2-L}{h^2}$	$\frac{1}{2h^2}$	$\frac{h^2-L}{h^2}$	$\frac{1}{2h^2}$

Table 3.2: Constants for the sigma-point methods. $\lambda = \alpha^2(L+\kappa) - L$ is a scaling parameter. Note that this $(10^{-2} \le \alpha \le 1)$ is different from $\alpha_k^{(\mathrm{m})}$ and $\alpha_k^{(\mathrm{c})}$. κ is either 0 or $3 - L$. β incorporates prior information. h may take any positive value. For Gaussian RVs, $\beta = 2$ or $h = \sqrt{3}$.

Each one of the input random-variable sigma points $\boldsymbol{\mathcal{X}}_i$ in the set $\boldsymbol{\mathcal{X}}$ is passed through the nonlinear function $f(\cdot)$ to produce a corresponding output sigma point, $\boldsymbol{\mathcal{Y}}_i = f(\boldsymbol{\mathcal{X}}_i)$. Then, the output mean and covariance are computed as well:

$$\bar{y} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \boldsymbol{\mathcal{Y}}_i \qquad \text{and} \qquad \boldsymbol{\Sigma}_{\tilde{y}} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\boldsymbol{\mathcal{Y}}_i - \bar{y})(\boldsymbol{\mathcal{Y}}_i - \bar{y})^T. \quad (3.26)$$

Fig. 3.21 illustrates the overall process. On the top right, we start with the mean vector and covariance matrix of the input random variable. In the example, $\bar{x}$ is a 4-vector and $\boldsymbol{\Sigma}_{\tilde{x}}$ is a 4×4 matrix. From these inputs, we create $2L + 1 = 9$ sigma points, which are stored compactly as the columns of a 4×9 matrix. The zeroth sigma point is equal to $\bar{x}$ so is drawn with the same shading. The next L sigma points are equal to the columns of $\gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$ added to $\bar{x}$. Since $\sqrt{\boldsymbol{\Sigma}_{\tilde{x}}}$ is lower triangular, all values above the diagonal are zero, and so the result when adding it to $\bar{x}$ differs from $\bar{x}$ only in the lower-triangular region. This is why the elements above the diagonal in

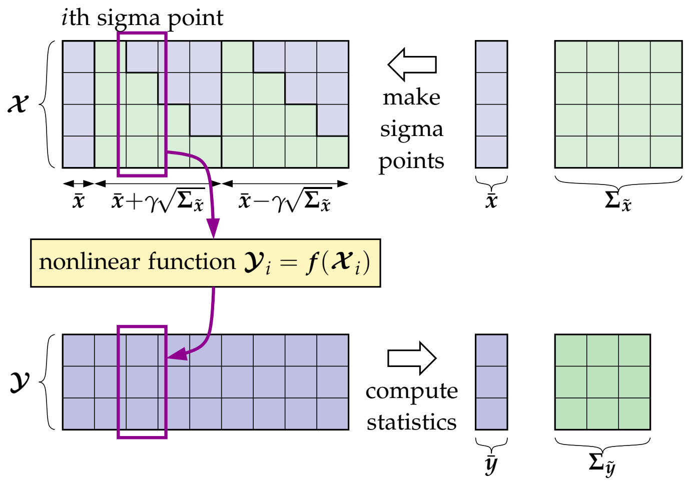


Figure 3.21: Visualizing the sigma-point approach.

the figure are drawn using the same shading as $\bar{x}$, but the lower-triangular elements are drawn with the same shading as $\Sigma_{\tilde{x}}$. The final L columns are computed as $\bar{x} - \gamma\sqrt{\Sigma_{\tilde{x}}}$ and stored in a similar way.

Next, each of the sigma points $\mathcal{X}_i$ in the set $\mathcal{X}$ is individually passed through the nonlinear function to produce a corresponding output sigma point $\mathcal{Y}_i$. These output sigma points form the set $\mathcal{Y}$ and are collected together in a matrix for convenient storage. In the figure, we have emphasized that the function output need not have the same dimension as the function input. In this case, input 4-vectors produce output 3-vectors. Despite the different dimension of the output, the weighting constants $\alpha_i^{(m)}$, $\alpha_i^{(c)}$, and γ, as well as number of sigma points $p + 1 = 2L + 1$ are inherited from the dimension of the input x.

Finally, the output statistics $\bar{y}$ and $\Sigma_{\tilde{y}}$ are computed from the sigma points in $\mathcal{Y}$ using Eq. (3.26). In this case, the mean is a 3-vector and the covariance is a 3×3 matrix. Note also that the elements corresponding to $\mathcal{Y}$ are drawn using a similar color scheme to those of $\mathcal{X}$, but with a different shade to emphasize that they are different quantities.

Before introducing the SPKF algorithm, we revisit the one-dimensional example in Fig. 3.20, now considering a sigma-point approach. In this example, the dimension of x is 1, so we require $2L + 1 = 3$ sigma points to represent the input random variable. These are shown in Fig. 3.22 as black squares on the horizontal axis. As expected, one sigma point is equal to the input pdf mean, and the other two are equally spaced above and below the mean.

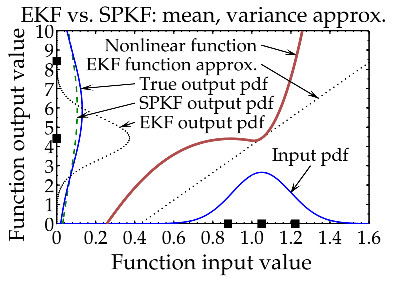


Figure 3.22: One-dimensional example revisited, using a sigma-point approach.

These three sigma points are passed through the nonlinear function, to compute three output sigma points, which are shown on the

12:07:34.

vertical axis (two of the output sigma points are very close to equal, so look to be coincident in the figure). The output mean is estimated using Eq. (3.26) to be 5.9, and the output standard deviation is estimated to be 3.80. A Gaussian pdf having these statistics is plotted on the vertical axis as a green dashed line. We see that this pdf is much closer to the truth than the prior estimate from the EKF approach.

Will the sigma-point method always be so much better than the analytic linearization used by EKF? The answer depends on the degree of nonlinearity of the state and output equations—the more nonlinear the equations, the better SPKF should be with respect to EKF.

3.10 The SPKF

3.10.1 Deriving the six steps of SPKF

We now return to the state-estimation problem and apply the sigma-point approach to propagating statistics through a nonlinear function. We follow the six steps of sequential probabilistic inference, as before.

SPKF step 1a: State-prediction time update.

For step 1a, we wish to approximate

$$\hat{x}_k^- = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big]$$

using a sigma-point approach. The general procedure is to represent the input randomness by sigma points, propagate those sigma points through $f(\cdot)$, and then compute $\hat{x}_k^-$ as the average of the output sigma points.

One complication that was not present when considering the static nonlinearity of Sect. 3.9.1 is that there are multiple sources of randomness in the dynamic problem: the state, process noise, and sensor noise are all random vectors. To use a sigma-point method, we must combine all the randomness in a single random vector.

So, we define an augmented random state vector x_k^a that combines these random factors at time index k

$$x_k^a = \begin{bmatrix} x_k \\ w_k \\ v_{k+1} \end{bmatrix}, \qquad \text{and} \qquad \Sigma_{\tilde{x}_k}^a = \begin{bmatrix} \Sigma_{\tilde{x},k} & 0 & 0 \\ 0 & \Sigma_{\tilde{w}} & 0 \\ 0 & 0 & \Sigma_{\tilde{v}} \end{bmatrix}.$$

This augmented vector is used in the estimation process as described below.

For step 1a, we first form the augmented posterior state estimate and augmented posterior state-estimation-error covariance matrix for

12:07:34.

the previous time interval

$$\hat{x}^{a,+}_{k-1} = \begin{bmatrix} \hat{x}^{+}_{k-1} \\ \bar{w} \\ \bar{v} \end{bmatrix}, \qquad \text{and} \qquad \Sigma^{a,+}_{\tilde{x},k-1} = \begin{bmatrix} \Sigma^{+}_{\tilde{x},k-1} & 0 & 0 \\ 0 & \Sigma_{\widetilde{w}} & 0 \\ 0 & 0 & \Sigma_{\tilde{v}} \end{bmatrix}.$$

These factors are used to generate the $p+1$ augmented sigma points

$$\mathcal{X}^{a,+}_{k-1} = \left\{ \hat{x}^{a,+}_{k-1},\ \hat{x}^{a,+}_{k-1} + \gamma\sqrt{\Sigma^{a,+}_{\tilde{x},k-1}},\ \hat{x}^{a,+}_{k-1} - \gamma\sqrt{\Sigma^{a,+}_{\tilde{x},k-1}} \right\}.$$

As before, these can be organized into a convenient matrix form, as depicted in Fig. 3.23. The figure shows the augmented state vector as three colored regions (the figure is drawn for the case where the process noise is a 2-vector and the sensor noise is a 1-vector). The augmented covariance matrix shows the block-diagonal structure, with white regions representing zero. The structure of the covariance matrix leads to a structure in the augmented sigma points, as illustrated in the depiction of $\mathcal{X}^{a,+}_{k-1}$.

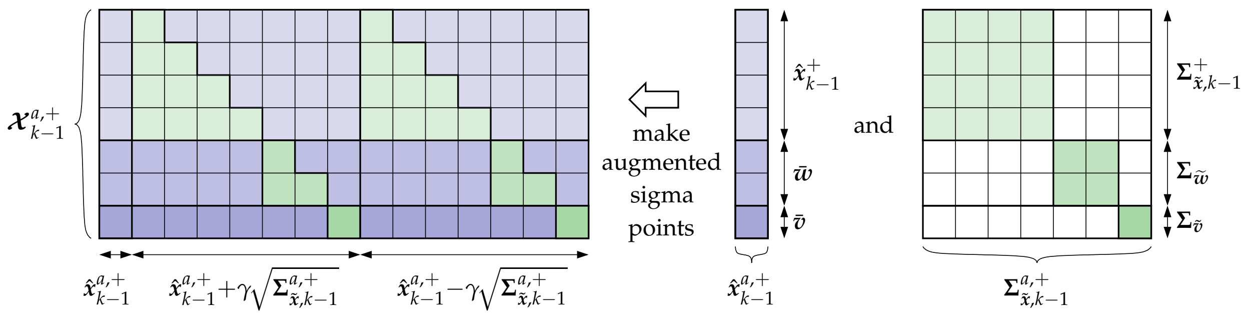
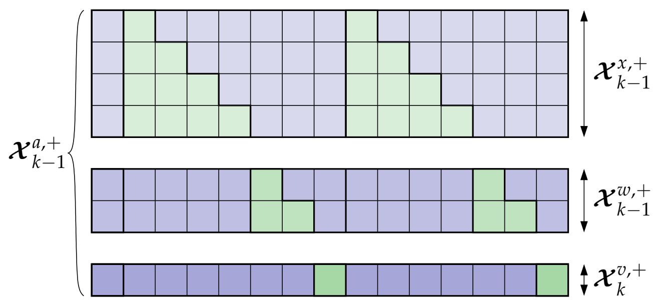

$\mathcal{X}^{a,+}_{k-1}$
make augmented sigma points
$\hat{x}^{+}_{k-1}$
$\bar{w}$
$\bar{v}$
and
$\Sigma^{+}_{\tilde{x},k-1}$
$\Sigma_{\widetilde{w}}$
$\Sigma_{\tilde{v}}$
$\hat{x}^{a,+}_{k-1}$
$\hat{x}^{a,+}_{k-1}+\gamma\sqrt{\Sigma^{a,+}_{\tilde{x},k-1}}$
$\hat{x}^{a,+}_{k-1}-\gamma\sqrt{\Sigma^{a,+}_{\tilde{x},k-1}}$
$\hat{x}^{a,+}_{k-1}$
$\Sigma^{a,+}_{\tilde{x},k-1}$

Figure 3.23: SPKF step 1a: Creating augmented sigma points.

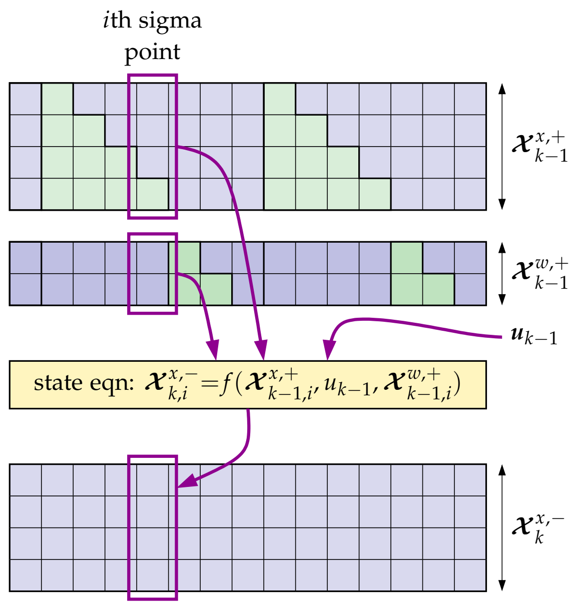
Once we have formed the sigma points, we split them into three parts, as depicted in Fig. 3.24. The top rows of $\mathcal{X}^{a,+}_{k-1}$ describe the randomness of the state estimate and are collectively denoted as the set $\mathcal{X}^{x,+}_{k-1}$; the middle rows describe the randomness of the process noise and are denoted as the set $\mathcal{X}^{w,+}_{k-1}$; the bottom rows represent the randomness of the process noise and are denoted as the set $\mathcal{X}^{v,+}_{k-1}$. We can further think of the ith element of each of these sets as $\mathcal{X}^{x,+}_{k-1,i}$, $\mathcal{X}^{w,+}_{k-1,i}$, and $\mathcal{X}^{v,+}_{k-1,i}$, respectively.

Now, we use the sigma-point approach to predict the present value of the state by propagating sigma points that represent the prior randomness through the state function $f(\cdot)$. This is illustrated in Fig. 3.25. We evaluate the state equation using all pairs of $\mathcal{X}^{x,+}_{k-1,i}$ and $\mathcal{X}^{w,+}_{k-1,i}$, yielding the state-prediction sigma points $\mathcal{X}^{x,-}_{k,i}$. That is, we compute $\mathcal{X}^{x,-}_{k,i} = f(\mathcal{X}^{x,+}_{k-1,i}, u_{k-1}, \mathcal{X}^{w,+}_{k-1,i})$.



$\mathcal{X}^{a,+}_{k-1}$
$\mathcal{X}^{x,+}_{k-1}$
$\mathcal{X}^{w,+}_{k-1}$
$\mathcal{X}^{v,+}_{k}$

Figure 3.24: Splitting the augmented sigma points into their three constituent parts.



ith sigma point
$\mathcal{X}^{x,+}_{k-1}$
$\mathcal{X}^{w,+}_{k-1}$
u_{k-1}
state eqn: $\mathcal{X}^{x,-}_{k,i} = f(\mathcal{X}^{x,+}_{k-1,i}, u_{k-1}, \mathcal{X}^{w,+}_{k-1,i})$
$\mathcal{X}^{x,-}_{k}$

Figure 3.25: Predicting the present state using sigma points representing prior randomness.

Finally, the state prediction is computed as the weighted average of these sigma points:

$$\hat{x}_k^- = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big] \approx \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} f(\mathcal{X}^{x,+}_{k-1,i}, u_{k-1}, \mathcal{X}^{w,+}_{k-1,i})$$
$$= \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{X}^{x,-}_{k,i}.$$

This operation can be computed efficiently as a matrix multiply, as illustrated in Fig. 3.26. If $\boldsymbol{\alpha}^{(\mathrm{m})}$ is a vector comprising all the $\alpha_i^{(\mathrm{m})}$ values and $\mathcal{X}^{x,-}_k$ is stored in matrix form, then $\hat{x}_k^- = \mathcal{X}^{x,-}_k \boldsymbol{\alpha}^{(\mathrm{m})}$.

SPKF step 1b: Error-covariance time update.

We wish to approximate $\Sigma^-_{\tilde{x},k} = \mathbb{E}[(\tilde{x}_k^-)(\tilde{x}_k^-)^T]$ using a sigma-point method. Most of the hard work has already been done, since we have already found sigma points that describe the state prediction. First,

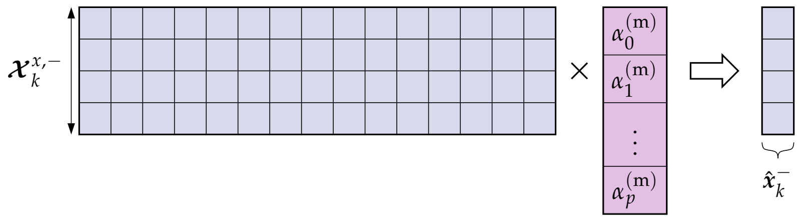


Figure 3.26: State prediction as a matrix multiply.

we compute

$$\widetilde{\boldsymbol{\mathcal{X}}}_{k,i}^{x,-} = \boldsymbol{\mathcal{X}}_{k,i}^{x,-} - \hat{x}_k^-$$

for every sigma point i. We then compute the covariance matrix as the weighted sum

$$\Sigma_{\tilde{x},k}^- = \sum_{i=0}^{p} \alpha_i^{(c)} \big(\widetilde{\boldsymbol{\mathcal{X}}}_{k,i}^{x,-}\big)\big(\widetilde{\boldsymbol{\mathcal{X}}}_{k,i}^{x,-}\big)^T.$$

If $\boldsymbol{\alpha}^{(c)}$ is a vector comprising all the $\alpha_i^{(c)}$ values and $\widetilde{\boldsymbol{\mathcal{X}}}_k^{x,-}$ is stored in matrix form, then this summation can be computed using a matrix product as

$$\Sigma_{\tilde{x},k}^- = \left(\widetilde{\boldsymbol{\mathcal{X}}}_k^{x,-}\right) \operatorname{diag}\left(\boldsymbol{\alpha}^{(c)}\right) \left(\widetilde{\boldsymbol{\mathcal{X}}}_k^{x,-}\right)^T,$$

as is illustrated in Fig. 3.27. This compact notation makes programming this step in MATLAB very straightforward, although there are a lot of multiply-by-zero operations that can be avoided if hand-coding for efficiency in a language like C.

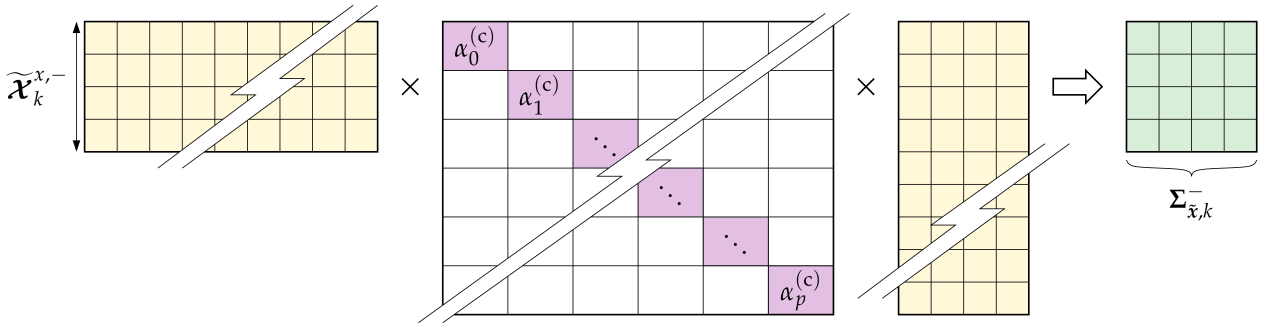


Figure 3.27: State-prediction error covariance calculation as a matrix multiply.

SPKF step 1c: Predict system output y_k.

In this step, we wish to approximate $\hat{y}_k = \mathbb{E}\big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\big]$. To do so, we will take the sigma points that represent the randomness in x_k and v_k and propagate them through the output equation $h(\cdot)$. The weighted mean of the output sigma points will comprise the output prediction.

12:07:34.

First, we compute the points $\boldsymbol{\mathcal{Y}}_{k,i} = \boldsymbol{h}(\boldsymbol{\mathcal{X}}_{k,i}^{x,-}, \boldsymbol{u}_k, \boldsymbol{\mathcal{X}}_{k,i}^{v,+})$, as illustrated in Fig. 3.28. The output estimate is then

$$\begin{aligned}\hat{\boldsymbol{y}}_k &= \mathbb{E}\big[\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \mid \mathbb{Y}_{k-1}\big] \approx \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \boldsymbol{h}(\boldsymbol{\mathcal{X}}_{k,i}^{x,-}, \boldsymbol{u}_k, \boldsymbol{\mathcal{X}}_{k,i}^{v,+}) \\ &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \boldsymbol{\mathcal{Y}}_{k,i}.\end{aligned}$$

This can be computed with a simple matrix multiplication, as we did when calculating $\hat{\boldsymbol{x}}_k^-$ at the end of step 1a.

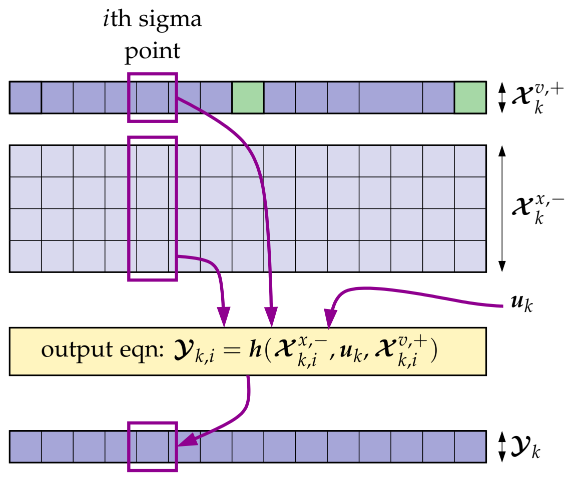

ith sigma point
$\boldsymbol{\mathcal{X}}_k^{v,+}$
$\boldsymbol{\mathcal{X}}_k^{x,-}$
$\boldsymbol{u}_k$
output eqn: $\boldsymbol{\mathcal{Y}}_{k,i} = \boldsymbol{h}(\boldsymbol{\mathcal{X}}_{k,i}^{x,-}, \boldsymbol{u}_k, \boldsymbol{\mathcal{X}}_{k,i}^{v,+})$
$\boldsymbol{\mathcal{Y}}_k$

Figure 3.28: Computing the output prediction sigma points.

SPKF step 2a: Estimator gain matrix $\boldsymbol{L}_k$.

To compute the estimator gain matrix, we must first compute the required covariance matrices:

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{y},k} &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)\big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)^T \\ \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^{-} &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{X}}_{k,i}^{x,-} - \hat{\boldsymbol{x}}_k^-\big)\big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)^T.\end{aligned}$$

These depend on the sigma-point matrices $\boldsymbol{\mathcal{X}}_k^{x,-}$ and $\boldsymbol{\mathcal{Y}}_k$, already computed in steps 1b and 1c, as well as $\hat{\boldsymbol{x}}_k^-$ and $\hat{\boldsymbol{y}}_k$, already computed in steps 1a and 1c. The summations can be performed using matrix multiplies, as we did in step 1b.

Then, once these covariance matrices are available, we simply compute $\boldsymbol{L}_k = \boldsymbol{\Sigma}_{\tilde{x}\tilde{y},k}^{-} \boldsymbol{\Sigma}_{\tilde{y},k}^{-1}$.

SPKF step 2b: State-estimate measurement update.

The state estimate is computed as

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k).$$

All necessary quantities have already been computed.

SPKF step 2c: Error-covariance measurement update.

The final step is calculated directly from the optimal formulation

$$\boldsymbol{\Sigma}^+_{\tilde{x},k} = \boldsymbol{\Sigma}^-_{\tilde{x},k} - \boldsymbol{L}_k \boldsymbol{\Sigma}_{\tilde{y},k} \boldsymbol{L}_k^T.$$

All necessary quantities have already been computed. The SPKF steps are summarized in the Appendix on p. 164.

3.10.2 *An SPKF example, with code*

As our first example of the SPKF method, consider again the scenario used to illustrate EKF in Sect. 3.7.2. We implement the SPKF in MATLAB in much the same structure as we implemented the EKF. First, we define some constants and reserve storage:

```
% Define size of variables in model
Nx = 1;     % state  = 1x1 scalar
Nxa = 3;    % augmented state has also w(k) and v(k) contributions
Ny = 1;     % output = 1x1 scalar

% Some constants for the SPKF algorithm. Use standard values for
% cases with Gaussian noises.  (These are the weighting matrices
% comprising the values of alpha(c) and alpha(m) organized in a
% way to make later computation efficient).
h = sqrt(3);
alpha1 = (h*h-Nxa)/(h*h); alpha2 = 1/(2*h*h);
alpham = [alpha1; repmat(alpha2,[2*Nxa 1])]; % mean weights
alphac = alpham; % covariance weights

% Initialize simulation variables
SigmaW = 1; % Process noise covariance
SigmaV = 2; % Sensor noise covariance
maxIter = 40;
xtrue = 2 + randn(1);  % Initialize true system initial state
xhat = 2;             % Initialize Kalman filter initial estimate
SigmaX = 1;         % Initialize Kalman filter covariance

% Reserve storage for variables we might want to plot/evaluate
xstore = zeros(maxIter+1,length(xtrue)); xstore(1,:) = xtrue;
xhatstore = zeros(maxIter,length(xhat));
SigmaXstore = zeros(maxIter,length(xhat)^2);
```

In this code `Nx` is the number of states, `Nxa` is the number of elements in the augmented state vector, and `Ny` is the number of outputs. The example uses the CDKF tuning parameters from Table 3.2 with tuning constant $h = \sqrt{3}$. The `alpham` and `alphac` vectors stores the $\boldsymbol{\alpha}^{(\mathrm{m})}$ and $\boldsymbol{\alpha}^{(\mathrm{c})}$ values, respectively, which happen to be the same in this simulation.

We now enter the main program loop:

```
for k = 1:maxIter,
  % SPKF Step 1a: State-estimate time update
  % 1a-i: Calculate augmented state estimate, including ...
  xhata = [xhat; 0; 0]; % process and sensor noise mean
  % 1a-ii: Get desired Cholesky factor
  Sigmaxa = blkdiag(SigmaX,SigmaW,SigmaV);
  sSigmaxa = chol(Sigmaxa,'lower');
  % 1a-iii: Calculate sigma points (strange indexing of xhat to avoid
  % "repmat" call, which is very inefficient in Matlab)
  % Note: xhata(:,ones([1 N])) creates an Nxa * N matrix where every
  % column is a copy of xhata.
  X = xhata(:,ones([1 2*Nxa+1])) + h*[zeros([Nxa 1]), ...
                                    sSigmaxa, -sSigmaxa];
  % 1a-iv: Calculate state equation for every element
  % Hard-code equation here for efficiency
  Xx = sqrt(5+X(1,:)) + X(2,:);
  xhat = Xx*alpham;

  % SPKF Step 1b: Covariance of prediction
  Xs = Xx - xhat(:,ones([1 2*Nxa+1]));
  SigmaX = Xs*diag(alphac)*Xs';

  % [Implied operation of system in background, with
  % input signal u, and output signal y]
  w = chol(SigmaW)'*randn(1);
  v = chol(SigmaV)'*randn(1);
  ytrue = xtrue^3 + v;  % y is based on present x and u
  xtrue = sqrt(5+xtrue) + w;  % future x is based on present u

  % SPKF Step 1c: Create output estimate
  % Hard-code equation here for efficiency
  Y = Xx.^3 + X(3,:);
  yhat = Y*alpham;

  % SPKF Step 2a: Estimator gain matrix
  Ys = Y - yhat*ones([1 2*Nxa+1]);
  SigmaXY = Xs*diag(alphac)*Ys';
  SigmaY  = Ys*diag(alphac)*Ys';
  Lx= SigmaXY/SigmaY;

  % SPKF Step 2b: State-estimate measurement update
  xhat = xhat + Lx*(ytrue-yhat); % update prediction to estimate

  % SPKF Step 2c: Error-covariance meas. update
  SigmaX = SigmaX - Lx*SigmaY*Lx';

  % [Store information for evaluation/plotting purposes]
  xstore(k+1,:) = xtrue;
  xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = (SigmaX(:))';
end
```

For the most part, this code implements the six SPKF steps in a straightforward way. However, there are some strange-looking MATLAB lines when tiling a matrix by repeating a vector multiple times horizontally, as documented in the first instance in step 1a-iii.[29]

For a simple example, if we write

```
x1 = [1;2;3];
x2 = x1(:,[1 1 1]);
```

[29] This would be far more clear using MATLAB's built-in `repmat` function. However, `repmat` is not very efficient and the strange code presented here is many times faster.

12:07:34.

this results in

$$x_2 = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}.$$

The line `x2 = x1(:,[1 1 1])` means, "compute `x2` by taking `x1`, all rows and column 1, then append `x1`, all rows and column 1, then append `x1`, all rows and column 1." Similarly, `xhata(:,ones([1 2*Nxa+1]))` replicates the `xhata` vector two times `Nxa` plus one times horizontally.

Fig. 3.29 displays results from running the SPKF. These results are for the same model, covariance-matrix tuning, and input/output data used by the EKF in Sect. 3.7.2. The top frame of Fig. 3.29 compares the SPKF estimates to the truth, and the lower frame shows SPKF and EKF errors along with error bounds. The SPKF estimates are, on the whole, better. Also, the SPKF estimation-error bounds are more reliable than the EKF bounds, which is a great improvement.

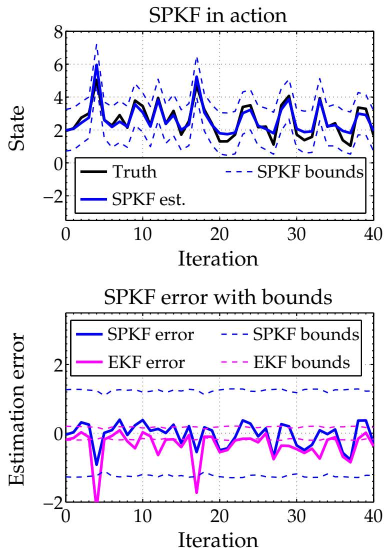


Figure 3.29: Simple SPKF example results.

3.11 *Implementing SPKF using the ESC cell model*

When implementing the SPKF on the ESC cell model, we refactor the code—much like we did in Sect. 3.8.2—to be more representative of the organization required in a battery-management-system main program loop. Because SPKF does not require derivatives, we don't need to spend time discussing how to adapt the SPKF approach to the ESC cell model. Rather, we proceed with a straightforward implementation. As in Sect. 3.8.2, the code is divided into a wrapper function that simulates the main program loop, an initialization function `initSPKF.m`, and an iteration function `iterSPKF.m`. The wrapper function for SPKF is the same as for EKF (except that it calls the SPKF initialization and iteration functions instead of the EKF versions).

The SPKF initialization code is presented below. When comparing to the corresponding EKF code, there should be no surprises. The main differences have to do with needing to initialize the SPKF tuning parameters γ, $\boldsymbol{\alpha}^{(\mathrm{m})}$, and $\boldsymbol{\alpha}^{(\mathrm{c})}$ and constants defining vector lengths for the augmented state vector. Also, since the Cholesky decomposition is a computationally intensive operation relative to the other SPKF computations, we precompute the Cholesky factor of the process-noise and sensor-noise covariances so that we don't need to do so every iteration:

```
function spkfData = initSPKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)

  % Initial state description
  ir0   = 0;                           spkfData.irInd = 1;
  hk0   = 0;                           spkfData.hkInd = 2;
  SOC0  = SOCfromOCVtemp(v0,T0,model); spkfData.zkInd = 3;
  spkfData.xhat  = [ir0 hk0 SOC0]';    % initial state

  % Covariance values
  spkfData.SigmaX = SigmaX0;
  spkfData.SigmaV = SigmaV;
  spkfData.SigmaW = SigmaW;
  spkfData.Snoise = chol(blkdiag(SigmaW, SigmaV),'lower');
  spkfData.SXbump = 5;

  % SPKF specific parameters
  Nx = length(spkfData.xhat); spkfData.Nx = Nx; % state-vector length
  Ny = 1; spkfData.Ny = Ny; % measurement-vector length
  Nu = 1; spkfData.Nu = Nu; % input-vector length
  Nw = size(SigmaW,1); spkfData.Nw = Nw; % process-noise-vector length
  Nv = size(SigmaV,1); spkfData.Nv = Nv; % sensor-noise-vector length
  Na = Nx+Nw+Nv; spkfData.Na = Na;     % augmented-state-vector length

  h = sqrt(3); spkfData.h = h; % SPKF/CDKF tuning factor
  alpha1 = (h*h-Na)/(h*h); % weighting factors when computing mean
  alpha2 = 1/(2*h*h);      % and covariance
  spkfData.alpham = [alpha1; alpha2*ones(2*Na,1)]; % mean
  spkfData.alphac = spkfData.alpham;               % covar

  % previous value of current
  spkfData.priorI = 0;
  spkfData.signIk = 0;

  % store model data structure too
  spkfData.model = model;
end
```

The SPKF iteration code also follows the same pattern as the EKF version, so we do not spend too much time discussing it. First, constants and variables are unpacked from the `spkfData` structure:

```
function [zk,zkbnd,spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData)
  model = spkfData.model;

  % Load the cell model parameters
  Q  = getParamESC('QParam',Tk,model);
  G  = getParamESC('GParam',Tk,model);
  M  = getParamESC('MParam',Tk,model);
  M0 = getParamESC('M0Param',Tk,model);
  RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
  R  = getParamESC('RParam',Tk,model)';
  R0 = getParamESC('R0Param',Tk,model);
  eta = getParamESC('etaParam',Tk,model);
  if ik<0, ik=ik*eta; end;

  % Get data stored in spkfData structure
  irInd  = spkfData.irInd;
  hkInd  = spkfData.hkInd;
  zkInd  = spkfData.zkInd;
  xhat   = spkfData.xhat;
  SigmaX = spkfData.SigmaX;

  % Get SPKF specific parameters
  Snoise = spkfData.Snoise;
  Nx = spkfData.Nx;
  Nw = spkfData.Nw;
  Nv = spkfData.Nv;
```

12:07:34.

```
Na = spkfData.Na;
alpham = spkfData.alpham;
alphac = spkfData.alphac;

% Dynamic variables relating to input current
I = spkfData.priorI;
if abs(ik)>Q/100, spkfData.signIk = sign(ik); end;
signIk = spkfData.signIk;
```

SPKF step 1a involves creating the augmented sigma points representing the prior estimate and associated noises. First, the augmented state-estimate vector `xhata` is created from the prior estimate `xhat`. Next, we create the Cholesky factor of the prior estimation-error covariance `SigmaX`. If `SigmaX` is not positive definite—it *should* be, but numeric imprecision can sometimes make `SigmaX` negative definite—then the optional return argument `p` from the `chol.m` function will indicate an error. If we encounter such an error, we must make some reasonable attempt to recover. In this case, we replace `SigmaX` with a diagonal matrix having elements equal to the absolute values of the diagonal of the original `SigmaX`. This forces positive semi-definiteness. Then, we compute the Cholesky factor as the square root of this matrix, enforcing that each diagonal element be at least as large as `SigmaW`:

```
% Step 1a: State-estimate time update
%          - Create xhatminus augmented SigmaX points
%          - Extract xhatminus state SigmaX points
%          - Compute weighted average xhatminus(k)

% Step 1a-1: Create augmented xhat and SigmaX
xhata = [xhat; zeros([Nw+Nv 1])];
[sigmaXa,p] = chol(SigmaX,'lower');
if p>0,
  fprintf('Cholesky error.  Recovering...\n');
  theAbsDiag = abs(diag(SigmaX));
  sigmaXa = diag(max(SQRT(theAbsDiag),SQRT(spkfData.SigmaW)));
end
sigmaXa= blkdiag(real(sigmaXa),Snoise);
% NOTE: sigmaXa is lower-triangular

% Step 1a-2: Calculate SigmaX points (strange indexing of xhata to
% avoid "repmat" call, which is very inefficient in MATLAB)
Xa = xhata(:,ones([1 2*Na+1])) + ...
     spkfData.h*[zeros([Na 1]), sigmaXa, -sigmaXa];
```

The remaining steps of the iteration should look very familiar, compared to our prior SPKF example. The biggest difference is that the model state and output equations are not embedded inline but are separated out as independent functions `stateEqn` and `outputEqn`. These functions are nested inside of `iterSPKF.m`, so they share the same variable space, and will be discussed shortly. Note the code added in step 2b after the state-estimate update to ensure that the hysteresis and SOC states remain within prescribed limits:

```
% Step 1a-3: Time update from last iteration until now
```

12:07:34.

```
%     stateEqn(xold,current,xnoise)
Xx   = stateEqn(Xa(1:Nx,:),I,Xa(Nx+1:Nx+Nw,:));
xhat = Xx*alpham;

% Step 1b: Error-covariance time update
%          - Compute weighted covariance sigmaminus(k)
%            (strange indexing of xhat to avoid "repmat" call)
Xs = Xx - xhat(:,ones([1 2*Na+1]));
SigmaX = Xs*diag(alphac)*Xs';

% Step 1c: Output estimate
%          - Compute weighted output estimate yhat(k)
I = ik; yk = vk;
Y = outputEqn(Xx,I,Xa(Nx+Nw+1:end,:),Tk,model);
yhat = Y*alpham;

% Step 2a: Estimator gain matrix
Ys = Y - yhat(:,ones([1 2*Na+1]));
SigmaXY = Xs*diag(alphac)*Ys';
SigmaY = Ys*diag(alphac)*Ys';
L = SigmaXY/SigmaY;

% Step 2b: State-estimate measurement update
r = yk - yhat; % residual.  Use to check for sensor errors...
if r^2 > 100*SigmaY, L(:,1)=0.0; end
xhat = xhat + L*r;
xhat(hkInd) = min(1,max(-1,xhat(hkInd)));
xhat(zkInd) = min(1.05,max(-0.05,xhat(zkInd)));

% Step 2c: Error-covariance measurement update
SigmaX = SigmaX - L*SigmaY*L';
[~,S,V] = svd(SigmaX);
HH = V*S*V';
SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help maintain robustness

% Q-bump code
if r^2>4*SigmaY, % bad voltage estimate by 2-SigmaX, bump Q
  fprintf('Bumping sigmax\n');
  SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*spkfData.SXbump;
end

% Save data in spkfData structure for next time...
spkfData.priorI = ik;
spkfData.SigmaX = SigmaX;
spkfData.xhat = xhat;
zk = xhat(zkInd);
zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));
```

The state equation is implemented in its own nested function. The inputs to this function are xold, current, and xnoise, where xold comprises the sigma points $\mathcal{X}^{x,+}_{k-1}$, xnoise comprises the sigma points $\mathcal{X}^{w,+}_{k-1}$, and current is the prior measured input current. Using efficient MATLAB matrix and vector operations, all output sigma points are computed simultaneously from the input sigma points, without requiring program loops:

```
% Calculate new states for all of the old state vectors in xold.
function xnew = stateEqn(xold,current,xnoise)
  current = current + xnoise; % noise adds to current
  xnew = 0*xold; % create space for xnew
```

12:07:34.

```
    xnew(irInd,:) = RC*xold(irInd,:) + (1-RC)*current;
    Ah = exp(-abs(current*G*deltat/(3600*Q)));  % hysteresis factor
    xnew(hkInd,:) = Ah.*xold(hkInd,:) + (Ah-1).*sign(current);
    xnew(zkInd,:) = xold(zkInd,:) - current/3600/Q;
    xnew(hkInd,:) = min(1,max(-1,xnew(hkInd,:)));
    xnew(zkInd,:) = min(1.05,max(-0.05,xnew(zkInd,:)));
  end
```

Similarly, the nested output equation function uses input `xhat` equal to $\mathcal{X}_k^{x,-}$, `current` equal to present measured current, and `ynoise` equal to $\mathcal{X}_k^{v,+}$. The output sigma points $\mathcal{Y}_k$ are computed simultaneously using efficient MATLAB matrix and vector operations:

```
  % Calculate cell output voltage for all of state vectors in xhat
  function yhat = outputEqn(xhat,current,ynoise,T,model)
    yhat = OCVfromSOCtemp(xhat(zkInd,:),T,model);
    yhat = yhat + M*xhat(hkInd,:) + M0*signIk;
    yhat = yhat - R*xhat(irInd,:) - R0*current + ynoise(1,:);
  end

  % "Safe" square root
  function X = SQRT(x)
    X = sqrt(max(0,x));
  end
end
```

Finally, the `SQRT` function computes a square root that guarantees a real result even if the input has become negative due to numeric imprecision.

The SPKF was executed on the same data as used in the example in Sect. 3.8.2. Results are shown in Fig. 3.30. In this example, RMS SOC estimation error was 0.84 %, and the true SOC values were outside the estimation-error bounds of the filter for 10.5 % of the time. This is a marked improvement when compared to the results from the EKF.

However, also recall that this particular example uses data from a cell test conducted at 5 °C. The nonlinear hysteresis is much more significant at this temperature than at more typical (warmer) operating temperatures. At warm temperatures, the EKF and SPFK tend to give very similar performance because the cell model is not as nonlinear then. Also notice that both types of nonlinear Kalman filter appear to be giving poorer estimates at very low state of charge (when the true state of charge drops below about 10 %). The biggest contributing factor to this error is most likely that the cell model describes open-circuit voltage poorly in that range at this temperature. The filter is relying on this poor description and it is biasing the result. If a more accurate OCV relationship could be obtained, then both estimators would work better at these low SOCs. Further, the inaccurate error bounds are not as significant as it might first appear since most battery-pack applications will not run the cell in that range.

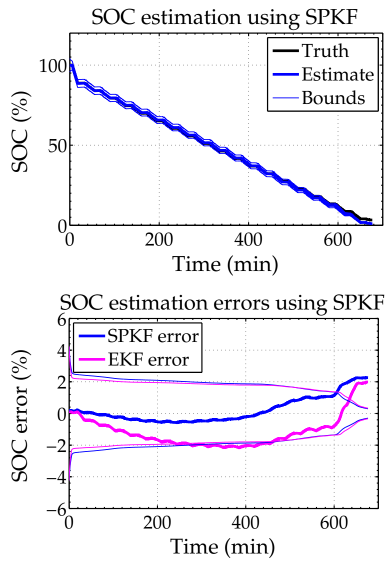


Figure 3.30: SPKF SOC estimates, compared to equivalent EKF estimates.

12:07:34.

3.12 Real-world issues pertaining to sensors, initialization

Most SOC estimators can appear to work well in simulation. However, the acid test is how well they work in real-world environments. In real applications, the estimators must be able to deal with nonideal sensors, with sensor faults, and so forth. In this section, we look at several real-world issues that the BMS engineer should design for, and see how the Kalman-filtering based approaches handle these problems very naturally.

3.12.1 Current-sensor bias

Current sensors often have a dc bias (i.e., a deterministic offset) in their readings. The measured current is equal to the true current plus a bias

$$i_k = i_k^{\text{true}} + i_k^{\text{bias}}.$$

For example, Hall-effect sensors experience hysteresis due to their underlying dependence on magnetic circuitry, which biases the sensor reading directly. Special compensation circuitry can help reduce this bias but not eliminate it completely. Shunt current sensors do not have a bias intrinsically, but the electronics that amplify the voltage drop over the shunt resistor may introduce a bias to the measured result.

We can compensate for bias in software to some extent. Suppose that we take a measurement of battery-pack current before the contactors are closed. Then we know that the true current i_0^{true} must be zero and thus that the measured current must be equal to the bias current. This bias estimate $\hat{i}_0^{\text{bias}}$ can be subtracted from all subsequent current measurements to approximate the true current

$$i_k^{\text{true}} \approx i_k - \hat{i}_0^{\text{bias}}.$$

However, this is only partially effective. Current-sensor bias often drifts with time and temperature. To be able to subtract bias out of the measured current effectively, we would need to update the bias estimate periodically. If the application allows communication with the load, this could be possible—the load could report back to the BMS whenever the load current is itself zero, and the BMS could then set the bias estimate equal to the present current-sensor measurement.

Bias compensation of some kind is critical for most SOC estimation methods. It is most important for coulomb-counting methods which integrate the measured current. Even if the cell's initial SOC and total capacity are known precisely, we have

$$\begin{aligned}\hat{z}_k &= z_0 - \frac{\Delta t}{Q}\sum_{j=0}^{k-1}\eta_k\left(i_k^{\text{true}} + i_k^{\text{bias}}\right)\\ &= z_k - \underbrace{\frac{\Delta t}{Q}\sum_{j=0}^{k-1}\eta_k i_k^{\text{bias}}}_{\text{estimation error}},\end{aligned}$$

which introduces an ever-growing error in the state of charge estimate.

Because Kalman filters use voltage measurements as feedback to update the state estimate, they do not tend to be as sensitive to bias as the coulomb-counting method. However, Kalman-filter theory assumes that the process noise has known mean and so an unknown current-sensor bias can still introduce permanent state of charge estimation error when the accumulated ampere-hours of bias move the state of charge estimate faster than the measurement updates can correct.

The best solution would be to design sensing hardware to eliminate current-sensor bias but this can be done only approximately. We can compensate in software by subtracting out a bias estimate that was measured when we knew current was zero, as discussed above, but this is a static estimate and not a dynamic estimate of a drifting bias.

However, the Kalman-filter based methods provide another avenue of compensation. We can estimate the unknown time-varying bias dynamically, and then subtract this dynamic estimate from the measured current. Using the ESC cell model as an example, we denote the time-varying bias estimate as i_k^{bias} and rewrite the state equation as

$$\begin{aligned} z_k &= z_{k-1} - (i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1})\Delta t/Q \\ \boldsymbol{i}_{R,k} &= \boldsymbol{A}_{RC}\boldsymbol{i}_{R,k-1} + \boldsymbol{B}_{RC}(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1}) \\ A_{h,k} &= \exp\left(-\left|(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1})\gamma\Delta t/Q\right|\right) \\ h_k &= A_{h,k}h_{k-1} + (A_{h,k} - 1)\operatorname{sgn}(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1}) \\ s_k &= \begin{cases}\operatorname{sgn}(i_k - i_k^{\text{bias}} + w_k), & |i_k - i_k^{\text{bias}} + w_k| > 0 \\ s_{k-1}, & \text{otherwise.}\end{cases}\end{aligned}$$

We then *augment* the state vector with a single bias state. We don't have an equation that describes the dynamics of the bias; instead, we model it as being essentially constant but having the ability to move via the addition of zero-mean white process noise n_{k-1}^{bias}:

$$i_k^{\text{bias}} = i_{k-1}^{\text{bias}} + n_{k-1}^{\text{bias}}.$$

This noise is different from the process noise w_k that affects the remaining state elements and is entirely fictitious. It exists only in the model and not in the real cell. It has very small covariance, but large enough to allow the Kalman filter to adapt the bias state. The output equation is also modified to include the bias:

$$y_k = \mathrm{OCV}(z_k) + Mh_k + M_0 s_k - \sum_j R_j i_{R_j,k} - R_0(i_k - i_k^{\text{bias}}) + v_k.$$

To implement this in a nonlinear Kalman filter, we must realize that the process-noise vector has been augmented. The w_k part influences only the primary states, and the n_k^{bias} part influences only the bias state. The augmented nature of the new process noise must be taken into account when computing the $\hat{\boldsymbol{B}}_k$ matrix for the EKF, and when determining the augmented sigma points in the SPKF method. The state description is augmented with the bias state in both nonlinear Kalman filters.

As the Kalman filter executes, the augmented state is adapted so that the input/output behavior of the model matches the input/output behavior of the cell as closely as possible. If there is a current-sensor bias, this will be accomplished when the bias estimate matches the true bias. The adaptability of the Kalman filter, combined with a good cell model, yields good state estimates.

3.12.2 *Voltage-sensor faults*

While voltage sensors can also have bias, it is less clear how to detect and correct the problem. We never have a "true" reading that we can use to calibrate the sensor. Voltage-sensor bias is also much less of an issue, because the biases tend to be on the order of a millivolt or so, and thus introduce very little state of charge estimation error.

A more common issue with voltage sensors occurs primarily when using external specialized integrated circuits to measure module voltages. Sometimes, communications between the primary BMS processor and these external measurement circuits can become corrupted or even lost. It is not always obvious that this has happened, although most modern chipsets have checksums built into the communications to assist with error detection, and some can even detect an open connection to a cell and report back an error condition.

In any case, having a software backup error-detection code can add robustness to the battery-management solution. And we have already seen how to implement such a method. Referring back to Sect. 3.6.5, we defined a normalized estimation error squared as

$$e_k^2 = \tilde{\boldsymbol{y}}_k^T \boldsymbol{\Sigma}_{\tilde{y},k}^{-1} \tilde{\boldsymbol{y}}_k.$$

12:07:34.

Then e_k^2 is a chi-square RV with m degrees of freedom, where m is the dimension of $\tilde{y}_k$. If e_k^2 has an unusually high value, then the sensor measurement is very unlikely. If $\tilde{y}_k > \chi_U^2$, then the measurement considered faulty and is discarded. Otherwise, the measurement is kept.

3.12.3 *Other sensor faults*

It is not obvious how to detect temperature-sensor and current-sensor faults. True battery-pack current can change from $i_{\min}$ to $i_{\max}$ in a single sample period, so simple rate-based tests cannot be implemented to detect a faulted sensor. The best check would probably be to use redundant sensing—if the load has a current-sensor reading, then compare the battery-management-system current-sensor reading to the load's measured value. If they are highly correlated, then both are probably working properly. Otherwise, at least one has probably failed.

To detect temperature-sensor faults, we might consider implementing a thermal model of cells or modules. This could allow us to reduce the number of required temperature sensors by using the model to interpolated between measured locations. It could also enable catching sensor faults using the same method as just described for the voltage sensor.

3.12.4 *Initialization*

A real-world issue that we have not thoroughly considered until now is, "How do we initialize the state estimate when the battery-management system is turned on?" In the simple EKF and SPKF examples of Sect. 3.7.2 and 3.10.2, we conveniently knew the true expected value and covariance of the state, so used that to initialize the filter. However, we do not have the same luxury in a real application.

Proper initialization requires that the BMS has a real-time clock that can report the duration of time that the battery pack has been idle. If the pack has rested for a relatively long time before being started, then we can assume that the battery cells are in electrochemical equilibrium and that cell voltage is equivalent to OCV. So, we reset the SOC estimates based on the measured OCV, we set the diffusion currents to zero, and we keep the prior hysteresis state (because hysteresis does not change when the cell is resting). This is the method that was used to initialize the simulations in Sect. 3.8 and 3.11.

If the pack has been resting for a short period of time, this method will not work well. Imagine a vehicular application where you are late for work but desperately need your morning coffee. You race

12:07:34.

into the parking lot of your favorite convenience store, jam on the brakes, and turn off the vehicle. You race into the store, purchase your coffee, and return to the vehicle. It's been less than two minutes, a personal record. You turn on your car again, and the BMS must make a good initial SOC estimate.

The battery cells are not in steady-state. It often takes tens of minutes or even hours to reach equilibrium. In fact, because you had been driving aggressively and since you jammed on the brakes as you were parking, a large regeneration pulse was sent into the battery pack just prior to turning the vehicle off, when the battery-management system last stored cell state estimates to nonvolatile memory. The stored estimates may be good, but the present measured voltages will be much higher than OCV. We cannot simply initialize the same way as if the pack were in equilibrium.

Instead, we set up and execute a simple time and measurement update (via a simple Kalman filter) for the state equations involving SOC and diffusion currents. Hysteresis voltages do not change, since the pack has been resting. A single-step execution of this Kalman filter will update the state estimate and its covariance matrix based on the total time the battery pack has been resting.

3.13 *Reduced computational complexity using bar-delta filtering*

As motivation for the last major section of this chapter, we consider again a philosophical question with very important practical implications. Consider the trivial series-connected battery pack in Fig. 3.31. What is the battery-pack SOC? We know that the SOC cannot be 0 % because we cannot charge the battery pack without overcharging the lower cell. Neither can SOC be 100 % because we cannot discharge without over-discharging the upper cell. It also cannot be the average of the two, 50 %, because we can neither charge nor discharge. So, as we have discussed before, "battery-pack SOC" is not a well-defined or helpful concept, by itself.

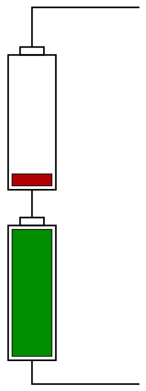


Figure 3.31: A trivial series-connected battery pack.
(Duplicated from Fig. 1.20.)

The example considers an extreme and unlikely scenario, but illustrates the importance of estimating the states-of-charge of *all* cells in a battery pack, even in the typical case. However, doing so introduces a real-world concern. Kalman filters are computationally complex. Running one nonlinear Kalman filter for a single-cell battery pack is okay, but running 100 nonlinear Kalman filters for a 100-cell battery pack will probably require more computation than many battery-management systems can provide. In this section we talk about efficient state of charge estimation for all individual cells in a large battery pack.

The modified method that we propose is based on the observation that while "battery-pack SOC" does not make sense, the concept of "battery-pack-*average* SOC" is a useful one. Battery-pack-average SOC is computed in the same way that we might think of computing "battery-pack SOC" but we are careful to include "average" in the name so that we think about this term in a correct way.

Since all cells in a series-connected battery pack experience the same current, we expect their individual state of charge values to: 1) collectively move in the same direction for any given applied current, 2) by a similar amount (but somewhat different because of unequal cell total capacities). We take advantage of this similarity by creating one algorithm to determine the composite average behavior of all cells in the battery pack, and a second algorithm to determine the individual differences between the state of specific cells and that composite average behavior.

We define pack-average state "x-bar" as[30]

$$\bar{\boldsymbol{x}}_k = \frac{1}{N_s}\sum_{i=1}^{N_s} \boldsymbol{x}_k^{(i)},$$

where $\boldsymbol{x}_k^{(i)}$ is the state of the ith cell. We can then write an individual cell's state vector as $\boldsymbol{x}_k^{(i)} = \bar{\boldsymbol{x}}_k + \Delta\boldsymbol{x}_k^{(i)}$ where $\Delta\boldsymbol{x}_k^{(i)}$ (called "delta-x") is the difference between the state vector of cell i and the pack-average state vector. The method to be developed is called "bar-delta filtering," as inspired by the "x-bar" and "delta-x" naming convention.[31]

We use one nonlinear Kalman filter to estimate the pack-average state, and N_s nonlinear Kalman filters to estimate the delta states. This is illustrated in Fig. 3.32.

[30] Note, for example, that within the pack-average state vector and individual state vectors we have $0 \le \min_i(z_k^{(i)}) \le \bar{z}_k \le \max_i(z_k^{(i)}) \le 1$. Therefore, the range of $\bar{z}_k$ is within the standard state of charge range.

[31] This was first reported in Plett, G., "Efficient Battery Pack State Estimation Using Bar-Delta Filtering," in *Proc. 24th Electric Vehicle Symposium* (EVS24), Stavanger, Norway, 2009.

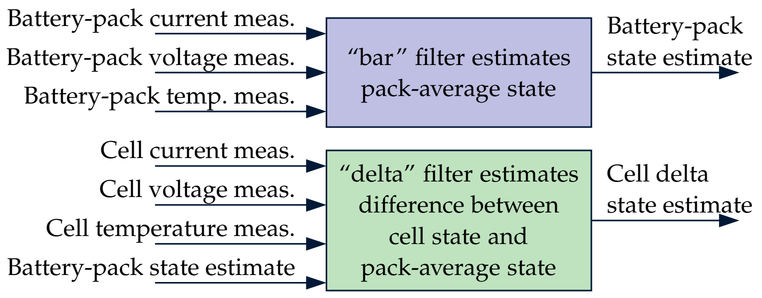

Battery-pack current meas.
Battery-pack voltage meas.
Battery-pack temp. meas.
"bar" filter estimates pack-average state
Battery-pack state estimate
Cell current meas.
Cell voltage meas.
Cell temperature meas.
Battery-pack state estimate
"delta" filter estimates difference between cell state and pack-average state
Cell delta state estimate

Figure 3.32: The bar-delta filter concept.

It may seem that we have taken a problem of complexity N_s and have replaced it with a problem of complexity $N_s + 1$. However, this is not the case—the three different types of estimator involved are not of identical computational complexity. The bar filter *is* of the same computational complexity as the individual state estimators that it uses as a basis (e.g., perhaps SPKF running on a single cell, as described up to this point in this chapter). However, the delta filters can be made very simple. Also, the delta states change much more slowly than the average state, so the delta filters can be run less frequently, as low as $1/N_s$ times the rate of the bar filter. The overall complexity can be reduced from order N_s to order 1^+. This is illustrated in Fig. 3.33. Instead of N_s high-complexity nonlinear Kalman filters, the bar-delta method uses a single high-complexity filter to estimate the battery-pack-average state, and N_s low-complexity filters, each one perhaps executed only once every N_s measurement intervals.

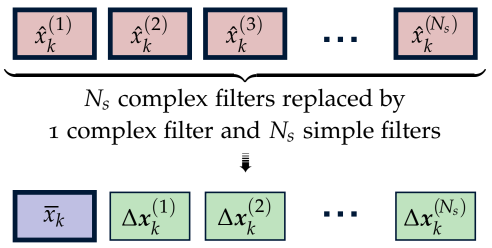


Figure 3.33: Reduced complexity of the $\bar{\Delta}$ filter.

3.13.1 *Bar-delta filtering using the ESC cell model: The bar filter*

To make this discussion more concrete, we show how to implement the method using the ESC cell model. In the implementation that we describe here, a "bar" filter estimates the pack-average state of charge, pack-average diffusion current(s), the pack-average hysteresis voltage, and the current-sensor bias.

As in Sect. 3.12.1, we model current-sensor bias as

$$i_k^{\text{bias}} = i_{k-1}^{\text{bias}} + n_{k-1}^{\text{bias}},$$

where n_k^{bias} is a fictitious noise source that is included in the model to allow SPKF to adapt the bias estimate. This current-sensor bias is common to all cells in a series-connected battery pack, so needs to be estimated only once (not once per cell).

We now need to find the pack-average-quantity state equations. For example, starting with a single-cell state of charge equation, we recall

$$z_k^{(i)} = z_{k-1}^{(i)} - i_{k-1}\Delta t/Q^{(i)}.$$

Summing this equation over all cells, and dividing both sides of the equation by N_s gives

$$\begin{aligned}\frac{1}{N_s}\sum_{i=1}^{N_s} z_k^{(i)} &= \frac{1}{N_s}\sum_{i=1}^{N_s} z_{k-1}^{(i)} - \frac{i_{k-1}\Delta t}{N_s}\sum_{i=1}^{N_s}\frac{1}{Q^{(i)}}\\ &= \frac{1}{N_s}\sum_{i=1}^{N_s} z_{k-1}^{(i)} - \frac{i_{k-1}\Delta t}{N_s}\sum_{i=1}^{N_s} Q_{\text{inv}}^{(i)}\\ \bar{z}_k &= \bar{z}_{k-1} - i_{k-1}\Delta t\bar{Q}_{\text{inv}},\end{aligned}$$

12:07:34.

where we have introduced the new concept of inverse total capacity $Q_{\text{inv}}^{(i)}$ of cell i to make the equations simpler, and where $\bar{Q}_{\text{inv}} = \sum_{i=1}^{N_s} Q_{\text{inv}}^{(i)}/N_s$. If we are estimating all cells' total capacities, we then use the time-varying quantity $\bar{Q}_{\text{inv},k-1}$. And, if we also consider the current-bias state,

$$\bar{z}_k = \bar{z}_{k-1} - (i_{k-1} - i_{k-1}^{\text{bias}})\Delta t \bar{Q}_{\text{inv},k-1}.$$

Similarly, the dynamics of all pack-average states and parameters of interest may be summarized as

$$\begin{aligned}
\bar{z}_k &= \bar{z}_{k-1} - (i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1})\Delta t \bar{Q}_{\text{inv},k-1} \\
\bar{\boldsymbol{i}}_{R,k} &= \boldsymbol{A}_{RC}\bar{\boldsymbol{i}}_{R,k} + \boldsymbol{B}_{RC}(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1}) \\
A_{h,k} &= \exp\left(-\left|(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1})\gamma\Delta t \bar{Q}_{\text{inv},k-1}\right|\right) \\
\bar{h}_k &= A_{h,k}\bar{h}_{k-1} + (A_{h,k} - 1)\operatorname{sgn}(i_{k-1} - i_{k-1}^{\text{bias}} + w_{k-1}) \\
\bar{s}_k &= \begin{cases} \operatorname{sgn}(i_k - i_k^{\text{bias}} + w_{k-1}), & |i_k - i_k^{\text{bias}} + w_{k-1}| > 0 \\ \bar{s}_{k-1}, & \text{otherwise} \end{cases} \\
i_k^{\text{bias}} &= i_{k-1}^{\text{bias}} + n_{k-1}^{\text{bias}}.
\end{aligned}$$

If we further wish to adapt estimates of the battery-pack average series resistance and inverse capacity, we can include the states

$$\begin{aligned}
\bar{R}_{0,k} &= \bar{R}_{0,k-1} + n_{k-1}^{\bar{R}_0} \\
\bar{Q}_{\text{inv},k} &= \bar{Q}_{\text{inv},k-1} + n_{k-1}^{\bar{Q}_{\text{inv}}}
\end{aligned}$$

to estimate these quantities in much the same way we estimated the current-sensor bias, where $n_k^{\bar{R}_0}$ and $n_k^{\bar{Q}_{\text{inv}}}$ are fictitious noise sources that allow the nonlinear Kalman filter to adapt the corresponding pack-average parameters.

The bar-filter for the pack employs a nonlinear Kalman filter that uses this model of pack-average states and the measurement equation

$$\bar{y}_k = \text{OCV}(\bar{z}_k) + M\bar{h}_k + M_0\bar{s}_k - \sum_j R_j \bar{i}_{R_j,k} - \bar{R}_{0,k}(i_k - i_k^{\text{bias}}) + v_k.$$

3.13.2 *Bar-delta filtering using the ESC cell model: The cell delta filters*

The quantities that we are most interested in estimating at the individual cell level are state of charge, resistance, and capacity. These all factor into determining pack available power and state-of-health estimates.

We first consider the delta-filter approach to determining cell state of charge. Note from before, $\Delta z_k^{(i)} = z_k^{(i)} - \bar{z}_k$. Then, using prior

12:07:34.

equations for the dynamics of $z_k^{(i)}$ and $\bar{z}_k$, we find:

$$\begin{aligned}\Delta z_k^{(i)} &= z_k^{(i)} - \bar{z}_k \\ &= \left(z_{k-1}^{(i)} - (i_{k-1} - i_{k-1}^{\text{bias}})\Delta t Q_{\text{inv},k-1}^{(i)}\right) \\ &\quad - \left(\bar{z}_{k-1} - (i_{k-1} - i_{k-1}^{\text{bias}})\Delta t \bar{Q}_{\text{inv},k-1}\right) \\ &= \Delta z_{k-1}^{(i)} - (i_{k-1} - i_{k-1}^{\text{bias}})\Delta t \Delta Q_{\text{inv},k-1}^{(i)},\end{aligned}$$

where $\Delta Q_{\text{inv},k}^{(i)} = Q_{\text{inv},k}^{(i)} - \bar{Q}_{\text{inv},k}$. Because $\Delta Q_{\text{inv},k}^{(i)}$ tends to be very small, the delta state $\Delta z_k^{(i)}$ does not change quickly, and can be updated at a slower rate than the pack-average SOC by accumulating $(i_{k-1} - i_{k-1}^{\text{bias}})\Delta t$ in between updates.

An output equation suitable for combining with this state equation is

$$\begin{aligned}y_k^{(i)} &= \text{OCV}(\bar{z}_k + \Delta z_k^{(i)}) + M\bar{h}_k + M_0\bar{s}_k - \sum_j R_j \bar{i}_{R_j,k} \\ &\quad - (\bar{R}_{0,k} + \Delta R_{0,k}^{(i)})(i_k - i_k^{\text{bias}}) + v_k.\end{aligned}$$

To estimate $\Delta z_k^{(i)}$, we could use an SPKF with these two equations. Because it is a single-state SPKF, it is very fast.

As a preview of parameter estimation (talked about in greater detail in Chap. 4), we can similarly make state-space models of the delta-resistance and delta capacity states. A simple state-space model of the delta-resistance state is

$$\begin{aligned}\Delta R_{0,k}^{(i)} &= \Delta R_{0,k-1}^{(i)} + n_{k-1}^{\Delta R_0} \\ y_k &= \text{OCV}(\bar{z}_k + \Delta z_k^{(i)}) - (\bar{R}_{0,k} + \Delta R_{0,k}^{(i)})(i_k - i_k^{\text{bias}}) + v_k^{\Delta R_0},\end{aligned}$$

where $\Delta R_{0,k}^{(i)} = R_{0,k}^{(i)} - \bar{R}_{0,k}$ and is modeled as a constant value with a fictitious noise process $n_k^{\Delta R_0}$ allowing adaptation, y_k is a crude estimate of the cell's voltage, and $v_k^{\Delta R_0}$ models estimation error. The dynamics of the delta-resistance state are simple and linear enough to use a single-state EKF rather than an SPKF.

To estimate cell capacity using an EKF, we model

$$\begin{aligned}\Delta Q_{\text{inv},k}^{(i)} &= \Delta Q_{\text{inv},k-1}^{(i)} + n_{k-1}^{\Delta Q_{\text{inv}}} \\ d_k &= (z_k^{(i)} - z_{k-1}^{(i)}) + (i_{k-1} - i_{k-1}^{\text{bias}})\Delta t \\ &\quad \times \left(\bar{Q}_{\text{inv},k-1} + \Delta Q_{\text{inv},k-1}^{(i)}\right) + e_k.\end{aligned}$$

The second equation is a reformulation of the state of charge state equation such that the expected value of d_k is equal to zero by construction. As the EKF runs, the computation for d_k in the second

12:07:34.

equation is compared to the known value (zero, by construction), and the difference is used to update the inverse-capacity estimate. Note that good estimates of present and previous states-of-charge are required for this estimate of total capacity to work well. Here, they come from the pack-average bar SPKF combined with the delta cell SPKF.

The output of the delta filters is computed by combining the average battery-pack state with the battery-cell delta states produced by the individual Kalman filters. Namely,

$$\begin{aligned} z_k^{(i)} &= \bar{z}_k + \Delta z_k^{(i)} \\ R_{0,k}^{(i)} &= \bar{R}_{0,k} + \Delta R_{0,k}^{(i)} \\ Q_k^{(i)} &= \frac{1}{\bar{Q}_{\text{inv},k} + \Delta Q_{\text{inv},k}^{(i)}}. \end{aligned}$$

3.13.3 *Example of bar-delta filtering, using desktop validation*

We illustrate bar-delta filtering with an example, but in the process consider the bigger issue of how one should validate a BMS algorithm. There are three different levels of validation.

The most important level of validation uses a hardware prototype. The final battery-management-system electronics plus software are connected to a battery pack. The battery pack is connected to its load, the entire system is exercised, and the algorithm predictions are evaluated. This is the most significant level of validation because it includes all real-world issues expected in a production system. However, it is also the most expensive, and truth values for cell states are difficult or impossible to determine post facto, even based on recorded data. There is great value in being able to eliminate most algorithm issues before arriving at the prototype stage.

The simplest level of validation uses only software on a desktop computer, a process we call *desktop validation*. This is similar in many ways to what we have already seen in this chapter. However, we do not use recorded cell-test data to exercise the algorithms. Instead, we use the ESC cell model to create synthetic test data for a wide variety of test-case scenarios. This allows access to the truth values of all cell and algorithm states, since the true SOCs, diffusion currents, hysteresis states (and so forth) are known when generating the synthetic data. The nonlinear Kalman-filter state estimates can be compared directly to the true cell states. This is very useful when tuning algorithms but the validity of the results are limited by the accuracy of the cell model.

An intermediate phase of validation involves *hardware-in-the-loop (HIL)* validation. In this scenario, the final BMS code is executed on

12:07:34.

the final BMS hardware. However, the input signals to the algorithms are synthetically generated in the same way as they were for desktop validation. This validation phase verifies that the final code in the final hardware produces identical estimates to the desktop case for the same scenarios.

Desktop and HIL validation strategies are illustrated in Fig. 3.34. Both require a data generation component that creates synthetic BMS data based on drive cycles and other initialization parameters. Both require an algorithm simulation component that simulates the algorithms using the synthetic data as input, based on various initialization parameters. The desktop-validation method executes these algorithms on a PC, perhaps using MATLAB. The HIL validation method executes these algorithms on the final hardware, perhaps implemented in C. Both should give identical results, and the results of both are evaluated compared to the known truth states provided by the data-generator system.

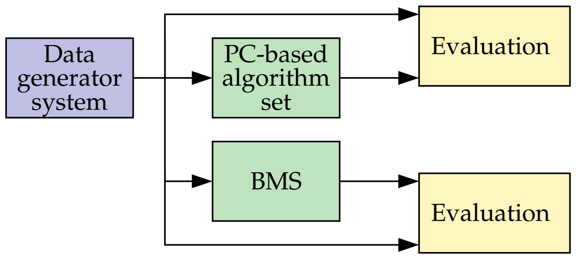


Figure 3.34: Desktop and HIL validation.

In either case, validation scenarios should include: normal operation, tests to see whether the algorithms recover from improper state of charge initialization, tests for robustness to sensor failures (sensor faults, sensor bias, and sensor noise), accuracy of algorithms when temperatures change, and when new and old cells are mixed in a battery pack. These scenarios should consider different load profiles over the entire required operational range of the battery pack. Insight and validation of the methods is gained via analysis and display of outputs.

3.13.4 *Examples of bar-delta accuracy and speed*

As an example, we consider a desktop-validation scenario of the bar-delta method. We simulated cycling of a four-cell battery pack with a UDDS drive-cycle profile, a rest period, the same UDDS drive-cycle profile, and a rest period. This profile is shown in the top frame of Fig. 3.35. The simulated battery pack comprised cells having true capacities of 6.5, 7.0, 7.5, and 8.0 Ah, resistances of 2.0, 2.25, 2.5, and 2.75 mΩ, and initial SOC values of 40, 45, 50, and 55 %. The

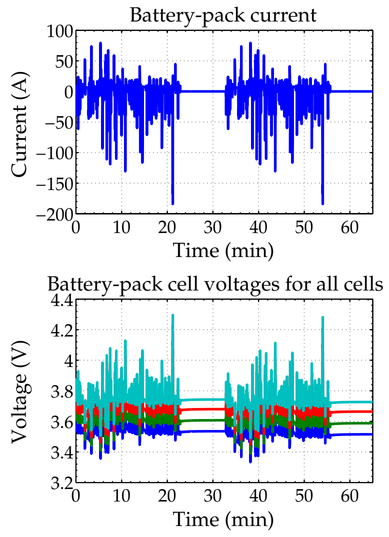


Figure 3.35: Cell current/voltage input/output data.

12:07:34.

current-sensor bias was 0.5 A. Simulated cell voltage measurements are shown in the bottom frame of Fig. 3.35.

The bar-delta algorithms were initialized with all cells having estimated capacity of 6.2 Ah, estimated resistances of 2.25 mΩ, estimated current-sensor bias of 0 A, and initial state of charge estimates based on initial voltages. SPKFs were used for the bar filter and the state of charge delta filters, and extended Kalman filters were used for the resistance and capacity-inverse delta filters.

Fig. 3.36 shows the overall bar-delta estimates of cell SOCs and resistance, compared to the truth values known from the process that generated the synthetic current/voltage input/output data. The SOC estimates are quite good, and improve over time as the filter learns the individual cell resistances and capacities and the current-sensor bias. Similarly, while the cell resistances are initialized to very crude estimates of the true value, these estimates improve relatively quickly over time and converge toward the truth by the end of the simulation.

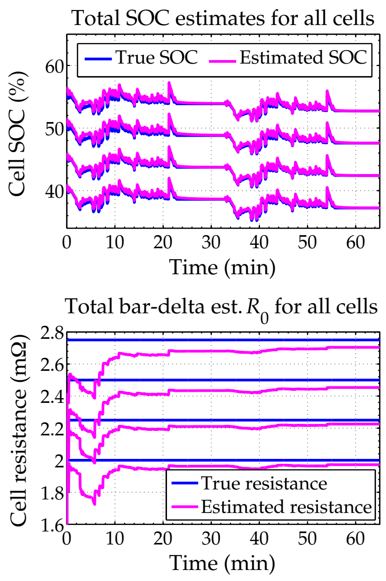


Figure 3.36: SOC and resistance estimates using bar-delta filtering.

Fig. 3.37 presents a slightly different point of view for these results. The top frame shows the battery-pack-average state of charge estimation error, plus bounds. The error is always less than about 1 %, and is always within the SPKF's 3σ error bounds, as desired. The bottom frame shows the overall SOC estimation errors for all four cells—considering the contributions of both the bar filter and the delta filters—plus their error bounds. Again, SOC estimation error is within about 1 % for all cells, and the errors are always within the error bounds produced by the filters. (The overall estimation-error variance is computed as the sum of the bar-filter variance plus the delta-filter variance, assuming that the errors are uncorrelated. The bounds are computed as three times the square root of the overall estimation-error variance.)

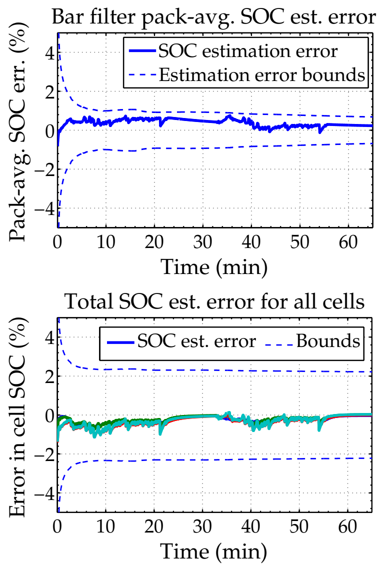


Figure 3.37: Bar-delta state of charge estimation errors.

Fig. 3.38 shows some other bar-delta estimates. The top frame shows battery-pack-average resistance estimates, which are intentionally initialized to incorrect values to show that they converge quickly to the true values. The truth is within the error bounds of the filter at all times, and the error bounds shrink over time as the filter gains confidence in its estimates. The bottom frame shows that the current-sensor-bias state also converges to the correct value over a reasonable period of time, that the true bias is always within the confidence interval of the estimator, and that the error bounds on the estimate decrease over time. We also notice that the bar-delta method produces good state estimates even during the interval when the current-sensor bias is not well estimated, but that the filter's estimates do improve when the bias is better modeled.

Finally, we comment without showing result figures that cell capac-

12:07:34.

ity estimates evolve in a similar way to resistance estimates. However, the time scale of adaptation is much longer since capacity is very weakly linked to the output measurement, as we will discuss further in Chap. 4. Abrupt changes in capacity will not be tracked very quickly by this method; but, slow capacity fade due to normal aging will be tracked well.

The bar-delta method was created to reduce computational complexity. So, is it faster? Table 3.3 presents timing results of the method. These were generated using SPKF and bar-delta algorithms hand-optimized in compiled C code, run on a (now-dated, but still representative) G4 processor. For a battery pack comprising 100 cells in series, 5.272 ms was required during each iteration to update 100 SPKFs. This scenario is the baseline for comparison, so is assigned a speedup value of 1.0. In contrast, a single bar filter required 0.067 ms per iteration to execute. This is somewhat more than one one-hundredth of the baseline case since the bar filter incorporated a bias-state estimate whereas the baseline case did not and because there is some overhead in the wrapper code that cannot be eliminated. The overall speedup of this case is 78.7.

Next, the entire bar-delta method was implemented, with one bar filter and 100 delta filters, where every delta filter was updated each iteration. This scenario required 0.190 ms per iteration, for a speedup of 27.7. Finally, a bar-delta method having one bar filter, 100 delta filters, but where only half of the delta filters were updated each iteration was implemented. This required 0.123 ms per iteration, for an overall speedup of 42.9. We see that a wide variety of speedups are possible using the bar-delta method.

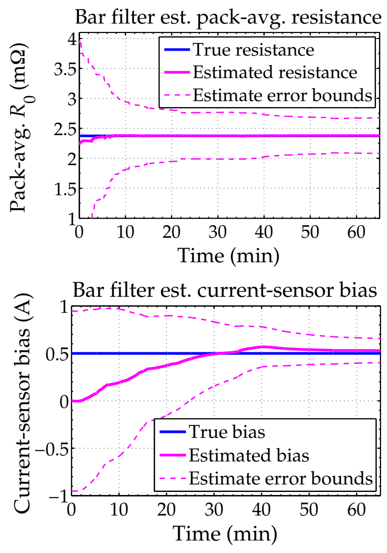


Figure 3.38: Bar-delta auxiliary state estimates.

3.14 *Where to from here?*

In this chapter, we have seen both good and bad ways to estimate state of charge for all cells in a battery pack. Model-based methods are preferred over some simpler but less robust methods. Kalman-filter-based methods are optimal under a very specific set of operat-

Test description (for pack having 100 cells)	CPU time per iteration	Speedup
One SPKF per cell	5.272 ms	1.0
One bar filter only, no delta filters	0.067 ms	78.7
1 bar, 100/100 delta filters updated ea. iter.	0.190 ms	27.7
1 bar, 50/100 delta filters updated ea. iter.	0.123 ms	42.9

Table 3.3: Speedup of bar-delta method for several scenarios

12:07:34.

ing conditions, but prove very robust even when the assumptions made in the derivations are not exactly met in practice.

Two additional benefits of Kalman filtering are that the algorithm produces confidence bounds on its estimates, and that it produces estimates for the entire state vector of the cell model, not only state of charge. Estimates of these additional state-vector values will be very valuable when considering how to produce battery-pack power estimates in Chap. 6.

There are a lot of nuances to Kalman filtering that are beyond the scope of this chapter. However, there are also quite a number of very good textbooks and online courses (including notes and videos of lectures) on the subject of Kalman filtering in general. The interested reader is referred to those for greater depth of understanding.[32]

[32] For one example, see `http://mocha-java.uccs.edu/ECE5550/`.

Referring back to Fig. 3.1, we have now addressed the state-estimation requirment of a BMS. Our next step is to look at state-of-health estimation, which is a form of model-parameter estimation. This is the main focus of Chap. 4.

3.15 *Appendices: Algorithms for state estimation*

The following pages have summary tables for the algorithms developed in this chapter.

General sequential-probabilistic-inference solution

General state-space model:

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \\ \boldsymbol{y}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k), \end{aligned}$$

where $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are independent Gaussian noise processes having covariance matrices $\boldsymbol{\Sigma}_{\widetilde{w}}$ and $\boldsymbol{\Sigma}_{\widetilde{v}}$, respectively.

Definitions: Let

$$\tilde{\boldsymbol{x}}_k^- = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^- \qquad \text{and} \qquad \tilde{\boldsymbol{y}}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_k.$$

Initialization: For $k = 0$, set

$$\begin{aligned} \hat{\boldsymbol{x}}_0^+ &= \mathbb{E}[\boldsymbol{x}_0] \\ \boldsymbol{\Sigma}_{\tilde{x},0}^+ &= \mathbb{E}\big[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)^T\big]. \end{aligned}$$

Computation: For $k = 1, 2, \ldots$ compute:

State-prediction time update: $\hat{\boldsymbol{x}}_k^- = \mathbb{E}\big[\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \mid \mathbb{Y}_{k-1}\big]$

Error-covariance time update: $\boldsymbol{\Sigma}_{\tilde{x},k}^- = \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^-)(\tilde{\boldsymbol{x}}_k^-)^T\big]$

Output estimate: $\hat{\boldsymbol{y}}_k = \mathbb{E}\big[\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k) \mid \mathbb{Y}_{k-1}\big]$

Estimator gain matrix:[*] $\boldsymbol{L}_k = \mathbb{E}\big[(\tilde{\boldsymbol{x}}_k^-)(\tilde{\boldsymbol{y}}_k)^T\big]\Big(\mathbb{E}\big[(\tilde{\boldsymbol{y}}_k)(\tilde{\boldsymbol{y}}_k)^T\big]\Big)^{-1}$

State-estimate meas. update: $\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k\big(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\big)$

Error-covariance meas. update: $\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k \boldsymbol{\Sigma}_{\tilde{y},k} \boldsymbol{L}_k^T$

[*]If a measurement is missed for some reason, then simply skip the measurement update for that iteration. That is, $\boldsymbol{L}_k = 0$ and $\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^-$ and $\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^-$.

Summary of the linear Kalman filter

Linear state-space model:

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$
$$\mathbf{y}_k = \mathbf{C}_k\mathbf{x}_k + \mathbf{D}_k\mathbf{u}_k + \mathbf{v}_k,$$

where $\mathbf{w}_k$ and $\mathbf{v}_k$ are independent, zero-mean, Gaussian noise processes of covariance matrices $\mathbf{\Sigma}_{\widetilde{w}}$ and $\mathbf{\Sigma}_{\tilde{v}}$, respectively.

Initialization: For $k = 0$, set

$$\hat{\mathbf{x}}_0^+ = \mathbb{E}[\mathbf{x}_0]$$
$$\mathbf{\Sigma}_{\tilde{x},0}^+ = \mathbb{E}\big[(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)(\mathbf{x}_0 - \hat{\mathbf{x}}_0^+)^T\big].$$

Computation: For $k = 1, 2, \ldots$ compute:

State-prediction time update:	$\hat{\mathbf{x}}_k^- = \mathbf{A}_{k-1}\hat{\mathbf{x}}_{k-1}^+ + \mathbf{B}_{k-1}\mathbf{u}_{k-1}$
Error-covariance time update:	$\mathbf{\Sigma}_{\tilde{x},k}^- = \mathbf{A}_{k-1}\mathbf{\Sigma}_{\tilde{x},k-1}^+\mathbf{A}_{k-1}^T + \mathbf{\Sigma}_{\widetilde{w}}$
Output estimate:	$\hat{\mathbf{y}}_k = \mathbf{C}_k\hat{\mathbf{x}}_k^- + \mathbf{D}_k\mathbf{u}_k$
*Estimator gain matrix:**	$\mathbf{L}_k = \mathbf{\Sigma}_{\tilde{x},k}^-\mathbf{C}_k^T[\mathbf{C}_k\mathbf{\Sigma}_{\tilde{x},k}^-\mathbf{C}_k^T + \mathbf{\Sigma}_{\tilde{v}}]^{-1}$
State-estimate meas. update:	$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{L}_k\big(\mathbf{y}_k - \hat{\mathbf{y}}_k\big)$
Error-covariance meas. update:	$\mathbf{\Sigma}_{\tilde{x},k}^+ = \mathbf{\Sigma}_{\tilde{x},k}^- - \mathbf{L}_k\mathbf{\Sigma}_{\tilde{y},k}\mathbf{L}_k^T$

*If a measurement is missed for some reason, then simply skip the measurement update for that iteration. That is, $\mathbf{L}_k = 0$ and $\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^-$ and $\mathbf{\Sigma}_{\tilde{x},k}^+ = \mathbf{\Sigma}_{\tilde{x},k}^-$.

12:07:34.

Summary of the nonlinear extended Kalman filter

Nonlinear state-space model:

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \\ \boldsymbol{y}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k), \end{aligned}$$

where $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are independent, Gaussian noise processes with means $\bar{w}$ and $\bar{v}$ and covariance matrices $\boldsymbol{\Sigma}_{\widetilde{w}}$ and $\boldsymbol{\Sigma}_{\tilde{v}}$, respectively.

Definitions:

$$\hat{\boldsymbol{A}}_k = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{w}_k)}{\mathrm{d}\boldsymbol{x}_k}\bigg|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_k^+} \qquad \hat{\boldsymbol{B}}_k = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{w}_k)}{\mathrm{d}\boldsymbol{w}_k}\bigg|_{\boldsymbol{w}_k=\bar{\boldsymbol{w}}_k}$$

$$\hat{\boldsymbol{C}}_k = \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{x}_k}\bigg|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_k^-} \qquad \hat{\boldsymbol{D}}_k = \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{v}_k}\bigg|_{\boldsymbol{v}_k=\bar{\boldsymbol{v}}_k}$$

Initialization: For $k = 0$, set

$$\begin{aligned} \hat{\boldsymbol{x}}_0^+ &= \mathbb{E}[\boldsymbol{x}_0] \\ \boldsymbol{\Sigma}_{\tilde{x},0}^+ &= \mathbb{E}\big[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)^T\big]. \end{aligned}$$

Computation: For $k = 1, 2, \ldots$ compute:

State-prediction time update:	$\hat{\boldsymbol{x}}_k^- = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1})$
Error-covariance time update:	$\boldsymbol{\Sigma}_{\tilde{x},k}^- = \hat{\boldsymbol{A}}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\hat{\boldsymbol{A}}_{k-1}^T + \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\Sigma}_{\widetilde{w}}\hat{\boldsymbol{B}}_{k-1}^T$
Output estimate:	$\hat{\boldsymbol{y}}_k = \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \bar{\boldsymbol{v}}_k)$
*Estimator gain matrix:**	$\boldsymbol{L}_k = \boldsymbol{\Sigma}_{\tilde{x},k}^-\hat{\boldsymbol{C}}_k^T[\hat{\boldsymbol{C}}_k\boldsymbol{\Sigma}_{\tilde{x},k}^-\hat{\boldsymbol{C}}_k^T + \hat{\boldsymbol{D}}_k\boldsymbol{\Sigma}_{\tilde{v}}\hat{\boldsymbol{D}}_k^T]^{-1}$
State-estimate meas. update:	$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k)$
Error -covariance meas. update:	$\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k\boldsymbol{\Sigma}_{\tilde{y},k}\boldsymbol{L}_k^T$

*If a measurement is missed for some reason, then simply skip the measurement update for that iteration. That is, $\boldsymbol{L}_k = 0$ and $\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^-$ and $\boldsymbol{\Sigma}_{\tilde{x},k}^+ = \boldsymbol{\Sigma}_{\tilde{x},k}^-$.

12:07:34.

Summary of the nonlinear sigma-point Kalman filter

Nonlinear state-space model:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$
$$y_k = h(x_k, u_k, v_k),$$

where w_k and v_k are independent, Gaussian noise processes with means $\bar{w}$ and $\bar{v}$ and covariance matrices $\Sigma_{\widetilde{w}}$ and $\Sigma_{\widetilde{v}}$, respectively.

Definitions: Let

$$x_k^a = \left[x_k^T,\ w_k^T,\ v_k^T\right]^T, \quad \mathcal{X}_k^a = \left[(\mathcal{X}_k^x)^T,\ (\mathcal{X}_k^w)^T,\ (\mathcal{X}_k^v)^T\right]^T,$$
$$p = 2 \times \dim(x_k^a).$$

Initialization: For $k = 0$, set

$$\hat{x}_0^+ = \mathbb{E}[x_0] \qquad \Sigma_{\tilde{x},0}^+ = \mathbb{E}\left[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\right]$$
$$\hat{x}_0^{a,+} = \mathbb{E}[x_0^a] = \left[(\hat{x}_0^+)^T,\ \bar{w},\ \bar{v}\right]^T \qquad \Sigma_{\tilde{x},0}^{a,+} = \mathbb{E}\left[(x_0^a - \hat{x}_0^{a,+})(x_0^a - \hat{x}_0^{a,+})^T\right] = \operatorname{diag}\left(\Sigma_{\tilde{x},0}^+, \Sigma_{\widetilde{w}}, \Sigma_{\widetilde{v}}\right).$$

Computation: For $k = 1, 2, \ldots$ compute:

State-estimate time update:
$$\mathcal{X}_{k-1}^{a,+} = \left\{\hat{x}_{k-1}^{a,+}, \hat{x}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}}, \hat{x}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}}\right\}$$
$$\mathcal{X}_{k,i}^{x,-} = f(\mathcal{X}_{k-1,i}^{x,+}, u_{k-1}, \mathcal{X}_{k-1,i}^{w,+})$$
$$\hat{x}_k^- = \textstyle\sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{X}_{k,i}^{x,-}$$

Error-covariance time update:
$$\widetilde{\mathcal{X}}_{k,i}^{x,-} = \mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-$$
$$\Sigma_{\tilde{x},k}^- = \textstyle\sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\widetilde{\mathcal{X}}_{k,i}^{x,-})(\widetilde{\mathcal{X}}_{k,i}^{x,-})^T$$

Output estimate:
$$\mathcal{Y}_{k,i} = h(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+})$$
$$\hat{y}_k = \textstyle\sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{Y}_{k,i}$$

*Estimator gain matrix:**
$$\widetilde{\mathcal{Y}}_{k,i} = \mathcal{Y}_{k,i} - \hat{y}_k$$
$$\Sigma_{\tilde{y},k} = \textstyle\sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\widetilde{\mathcal{Y}}_{k,i})(\widetilde{\mathcal{Y}}_{k,i})^T$$
$$\Sigma_{\tilde{x}\tilde{y},k}^- = \textstyle\sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\widetilde{\mathcal{X}}_{k,i}^{x,-})(\widetilde{\mathcal{Y}}_{k,i})^T$$
$$L_k = \Sigma_{\tilde{x}\tilde{y},k}^- \Sigma_{\tilde{y},k}^{-1}$$

State-estimate meas. update:
$$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$$

Error-covariance meas. update:
$$\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^- - L_k \Sigma_{\tilde{y},k} L_k^T$$

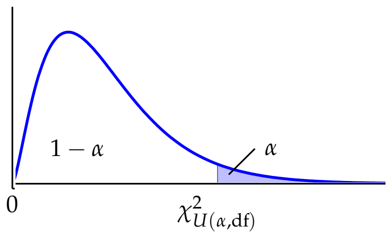
---

*If a measurement is missed for some reason, then simply skip the measurement update for that iteration. That is, $L_k = 0$ and $\hat{x}_k^+ = \hat{x}_k^-$ and $\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^-$.

12:07:34.

Appendix: Critical Values of $\chi^2_{U(\alpha,\text{df})}$

For a chi-squared distribution with some number of degrees of freedom, each entry in the following table represents the critical value of $\chi^2_{U(\alpha,\text{df})}$ for a specified upper tail area α.



Degrees of freedom	Upper tail areas											
	0.995	0.99	0.975	0.95	0.90	0.75	0.25	0.10	0.05	0.025	0.01	0.005
1	0.000	0.000	0.001	0.004	0.016	0.102	1.323	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	0.575	2.773	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	1.213	4.108	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	1.923	5.385	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	2.675	6.626	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	3.455	7.841	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	4.255	9.037	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	5.071	10.219	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	5.899	11.389	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	6.737	12.549	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	7.584	13.701	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	8.438	14.845	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	9.299	15.984	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	10.165	17.117	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	11.037	18.245	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	11.912	19.369	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	12.792	20.489	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	13.675	21.605	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	14.562	22.718	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	15.452	23.828	28.412	31.410	34.170	37.566	39.997

12:07:34.

4
Battery Health Estimation

4.1 Need for health estimates

Over time, cells in a battery pack will age and their performance will degrade. They will eventually reach a point where they no longer meet the performance requirements of the battery pack, which we consider to be the pack's end of life (although so-called "second-life" applications might be able to make use of the remaining diminished capability). Between a battery pack's beginning of life and end of life it is important to have knowledge regarding the present degradation status of its cells to be able to make accurate calculations of state of charge, available energy and available power.

In Volume I of this series, we focused on understanding *ideal* battery cells: cells that do not age and hence have constant model parameter values. Sadly, such cells do not exist. We now turn our attention to designing algorithms that work with *normal* battery cells: cells that do age, and hence have operational characteristics that degrade. We need to understand which parameters in the model change over time, which of these are most significant, and how to modify our battery-management methods to account for aging.

Note that normal aging is only one cause of cell failure. Failures can also occur because of cell design faults, poorly controlled manufacturing processes or impurities in the materials used during manufacture, abuse, and uncontrolled operations. Cells that have design or manufacturing faults or are abused often appear normal for a period of time and then fail very rapidly. The methods discussed in this chapter cannot predict this sudden-onset failure (it remains an open problem to do so), but there are some remediation methods that can maximize safety even if cells fail in this way.[1] Proper battery management will prevent uncontrolled operation while the BMS is active, but has no influence over external factors or over ambient conditions

[1] See, for example, Kim, G-H, Smith, K, Ireland, J, and Pesaran, A., "Fail-safe design for large capacity lithium-ion battery systems," *Journal of Power Sources*, 210, 2012, pp. 243–253.

while the BMS is inactive (e.g., physical damage during an xEV collision, or an out-of-bounds ambient temperature while the BMS and its thermal controls are turned off).

Referring back to Fig. 3.1, which illustrates the main topics of this book, it turns out that state estimation using an equivalent-circuit cell model works just as well for normal battery cells as it does for ideal battery cells so long as every cell's model parameter values are continuously updated to reflect the cell's present aged characteristics. In this chapter, we begin to examine how to estimate the cell-model parameter values that change relatively slowly. Our focus is on understanding and tracking normal aging processes as well as uncontrolled operations in the sense that overvoltage, overtemperature, and so forth accelerate the normal degradation mechanisms. The battery pack can still be used during the period when it is aging normally until its end of life, but at reduced levels of performance. We do not look at internal faults and abuse, which are usually detected using other means (e.g., those discussed in Chap. 1) and may require shutting down parts or all of the battery pack for immediate servicing in order to prevent propagation of the failure. The BMS designer should work closely with the battery-cell electrochemists and production engineers to understand all of the probable failure mechanisms and their characteristics fully to be able adapt the knowledge from this chapter into their individual designs.

In particular, we are most interested in those quantities that reflect a change in the performance that the battery pack can deliver. These are indicators of battery-pack SOH. There is no universally agreed-upon definition of SOH, but the most commonly estimated quantities used to summarize battery pack health include the present total capacity and present equivalent series resistance of every cell. Accurate estimates of total capacity and equivalent series resistance allow us to compute reliable total energy and available power estimates for the battery pack over its service lifetime.

4.1.1 Total capacity

As a battery cell ages, its total capacity Q decreases. In a lithium-ion cell, this is due primarily to unwanted side reactions that consume lithium that could otherwise be used during charge and discharge of the cell, and to structural deterioration of the electrode active materials that eliminates lithium storage sites.

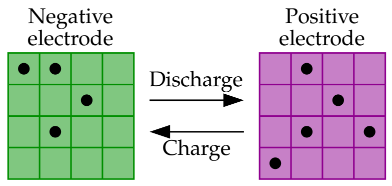


Figure 4.1: Ideal-cell operation.

Fig. 4.1 illustrates a simplified example of ideal-cell behavior. In the figure, both the negative and positive electrodes have sixteen sites that could hold lithium. Presently, four negative-electrode sites and five positive-electrode sites are shown to be occupied. When the

12:07:34.

cell is fully charged, there will be nine occupied sites in the negative electrode, and no occupied sites in the positive electrode. When the cell is fully discharged, there will be no occupied sites in the negative electrode, and nine occupied sites in the positive electrode.[2] The total capacity of the cell is equal to the minimum of the number of storage sites in the negative electrode, the number of storage sites in the positive electrode, and the amount of lithium that can be cycled. In this example, the total capacity is the minimum of 16, 16, and 9, which yields a total capacity of nine lithium atoms.

[2] This is a simplification. Neither electrode is ever completely full or completely empty during operation. Instead, a cell is considered fully charged or fully discharged when its open-circuit voltage reaches predetermined maximum and minimum levels.

Continuing the example, a side reaction is an undesired chemical process that consumes lithium while it is in transit from one electrode to another and removes it from cycling. Most side reactions happen while the cell is being charged. This is illustrated in Fig. 4.2, where one lithium atom is shown being consumed by a side reaction as the cell is being charged. The total capacity has now been reduced to the minimum of 16, 16, and 8, which yields a total capacity of eight lithium atoms.

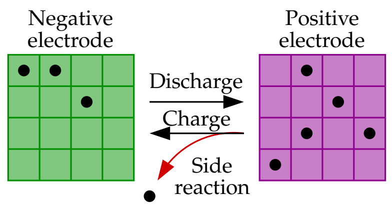


Figure 4.2: Capacity loss due to side reaction.

Structural deterioration is something that eliminates lithium storage sites from one of the electrodes, perhaps due to a collapse of part of the crystal structure of the electrode itself. This is illustrated in Fig. 4.3, where damage to the positive electrode is shown as a white scar. Some lithium may be trapped in the structure such that it is no longer free to cycle back and forth as the cell is charged and discharged, so capacity is lost. In the figure, the lithium atom in the lower-right corner of the positive electrode is trapped by the structural collapse. Structural collapse can also eliminate lithium storage sites. In the figure, the positive electrode has only ten good storage sites, but one of those is isolated from cycling, so there are only nine useable storage sites. So, the total capacity is the minimum of 16, 9, and 8, yielding a final total capacity of 8 lithium atoms.

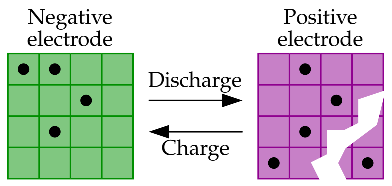


Figure 4.3: Capacity loss due to material loss.

This slow reduction in capacity is often referred to as *capacity fade*. We require algorithms that track capacity fade to provide the other battery-management-system algorithms with up-to-date estimates of every cell's total capacity. This knowledge is critical to be able to calculate battery-pack available energy accurately, where total capacity is a major contributing factor (as we saw in Sect. 1.14). If coulomb counting is being used for SOC estimation, an accurate estimate of total capacity is also needed; however, if Kalman filters are being used instead, the state of charge estimates turn out to be fairly insensitive to a poor total-capacity estimate since the built-in feedback correction mechanism is able to compensate for moderate errors in the total-capacity estimate. The dependence of available-power estimates on the value of total capacity also turns out to be minimal.

4.1.2 Equivalent series resistance (ESR)

As a battery cell ages normally, its equivalent series resistance R_0 increases. This is also due primarily to unwanted side reactions and structural deterioration. The side reactions tend to form resistive films on the surface of the active-material particles that impede the ionic conductivity. Structural deterioration severs electronic pathways between particles and decreases the electronic conductivity.

Having an up-to-date knowledge of equivalent-series resistance is important because it turns out to be a major contributing factor to the available-power calculation.[3] It can be a major contributing factor to state of charge estimation for some voltage-based methods; however, for Kalman-filter-based methods it it is a minor factor, and it does not affect coulomb-counting methods at all. It does not have a significant role in available-energy estimation.

[3] Because resistance and power are so tightly coupled, resistance rise in a cell is commonly referred to as *power fade*.

4.1.3 Other cell parameters

As the cell ages, other cell-model parameter values will change as well. For example, while the open-circuit-potential relationships of each electrode remain fixed by the chemistry of the crystal structures, a cell's overall OCV relationship can change due to shifts in the stoichiometric operating windows used by each electrode that occur when capacity is lost due to side reactions and structural deterioration. This effect tends not to be large, but may need to be tracked by future BMS.

Other cell-model parameter values almost certainly change as well; however, few if any present BMS make efforts to estimate the changes. Overall, changes to the equivalent-series resistance and total capacity have the dominant impact on BMS performance.

As WE PROCEED IN THIS CHAPTER, we will first discuss in greater detail the primary specific known electrochemical and structural mechanisms by which a lithium-ion cell can degrade. We will see that this is a lot more complicated than the simple example just presented, but that the dominant outcomes are still well represented by changes in resistance and total capacity in an equivalent-circuit cell model. There are other secondary degradation mechanisms and doubtless some that have not yet been discovered, but so long as they manifest as slow changes in cell resistance and total capacity, the methods of this chapter will continue to work.

We will then explore the concept of *sensitivity*, leading to an understanding of how *observable* the changes in total capacity and resistance are from cell input/output (current/voltage) measurements. This will lead directly to a simple method that can estimate equivalent-series resistance with a good degree of accuracy. However, it will also show that finding a good estimate of cell total capacity is a very challenging task.

We know that a nonlinear Kalman filter can be used to estimate the quickly time-varying states of a cell model. It turns out that nonlinear Kalman filters can also be used to estimate the slowly time-varying parameter values. We will investigate how to do so, and some remedies to a common pitfall when trying to estimate the states and parameters of a model at the same time. An advantage of this approach is that it can be used to estimate any time-varying parameter in the cell model—not only resistance and capacity.

If we cannot afford the complexity of a Kalman-filter method and yet desire to estimate total capacity, we might consider a regression method. However, it turns out that methods for estimating total capacity that are based in principle on ordinary least-squares regression will yield biased results. We spend a considerable amount of time in this chapter seeing why this is true, and showing how to estimate total capacity correctly using regression. We further explore how we can know that our estimate is correct and how to compute confidence intervals for our total-capacity estimates.

4.2 *Negative-electrode aging*

Lithium-ion battery-cell operational characteristics change slowly over time as the cell ages. This aging can be captured in an equivalent-circuit cell model by adapting the parameter values within the model so that good predictions are made over the entire lifetime of the battery cell. However, since equivalent-circuit-model parameters do not individually describe any of the physical electrochemical processes taking place inside the cell, these changing parameter values do not give any insight into why or how the aging has occurred.

To understand aging, we must consider a cell from a physics-based perspective.[4] In the next sections, we will seek to describe aging *qualitatively*, as even this simplified degree of understanding is valuable to the BMS algorithm designer. For example, it helps to explain why manufacturers put voltage limits and current limits on cells.

[4] Because aging depends on the physics of the cell, aging mechanisms of a lithium-ion cell will be different from those of other types of cell. We restrict our discussion to lithium-ion cells here, basing this section on the excellent paper by Vetter et al., "Ageing mechanisms in lithium-ion batteries," *Journal of Power Sources*, 147, 2005, 269–281.

For maximum impact, we would need to be able to model aging *quantitatively* as well. Two crude reduced-order quantitative cell-degradation models are introduced in Chap. 7. These models are simple enough to be used alongside an equivalent-circuit model and can be used to track some kinds of aging, leading to some preliminary physics-based control methods. For more advanced controls, which provide more accurate limits on the power available from the battery pack, one would need to combine reduced-order physics-based degradation models with a reduced-order physics-based cell model, such as was developed in Vol. I. This is one topic we will cover in the planned Vol. III of this series.

In a lithium-ion cell, aging processes occur within both the negative and positive electrodes. We consider degradation in the negative electrode first, where aging effects are seen at three scales. First, some aging occurs at the surface of the electrode active-material particles; that is, at the interface between the solid and the electrolyte. Second, other aging mechanisms take place within the interior of the active-material particles. Third, aging processes can take place within the overall composite electrode structure, including changes to the active materials, the conductive additives, the binder materials, the current collectors, the porosity, and so forth. We will consider aging at these three scales separately in the next subsections.

4.2.1 *Negative electrode aging at surface of particles*

Most commercial lithium-ion battery cells have negative-electrode active materials that are comprised of a synthetic or natural graphite. Graphite has good lithium storage capacity, can be charged and discharged repeatedly, and is inexpensive and nontoxic. Most important, perhaps, is that lithiated graphite has very low voltage with respect to a lithium-metal reference. This is key to maximizing overall cell voltage, since cell voltage is equal to the positive-electrode potential minus the negative-electrode potential.

Fig. 4.4 plots the open-circuit-potential relationships for graphite and for another candidate negative-electrode active material: lithium-titanate-oxide (LTO). The horizontal axis is the present operating stoichiometry of the electrode and corresponds to the value of x in Li_xC_6, for graphite or in $Li_{4+3x}Ti_5O_{12}$ for LTO. The black dot at $x = 0.55$ shows that the open-circuit potential of graphite is about 0.1 V at that point. Charging a cell increases x and causes the potential to decrease; discharging does the opposite.

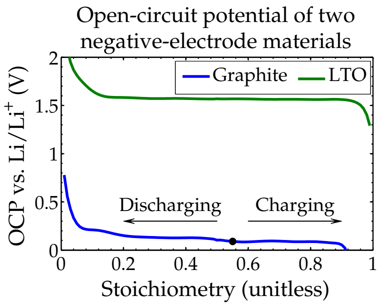


Figure 4.4: Open-circuit-potential relationships for two negative-electrode active materials.

While the low potential of graphite over most of its operational range makes it highly desirable for high-voltage lithium-ion cells, it is also the cause of the dominant degradation mechanism that takes place inside a lithium-ion cell. This low electrical potential is outside the electrochemical voltage stability window of the organic solvents used in lithium-ion cell electrolytes. When the electrolyte solvents come into contact with lithiated graphite, reductive electrolyte decomposition takes place at the electrode/electrolyte interface. The rate

12:07:34.

of decomposition is accelerated by lower electrode potentials, which occur when the electrode (and the cell) are in a high SOC.[5]

[5] LTO has high-enough potential that this side reaction does not occur, greatly extending life. However, it also lowers the overall cell voltage, and hence the cell's energy density.

Lithium-ion cells are constructed in a fully discharged state (all of the lithium is in the positive electrode). The nonlithiated graphite is then at a high-enough potential that the electrolyte-reduction reaction does not occur during cell fabrication. However, when the cell is charged for the first time, the graphite becomes lithiated and its potential drops. Solvent in the electrolyte that comes into contact with the surface of the particle is reduced and forms reaction products that coat the surface of the electrode particle with a *solid–electrolyte interphase (SEI)* surface film. The SEI is a passivating layer that partially insulates the graphite from the remaining solvent in the electrolyte and hence slows down further reaction. Hence, most of the SEI is formed during the initial charge process of the cell, which leads to this first charge being termed the *formation process*. The growth of SEI is illustrated in Fig. 4.5.

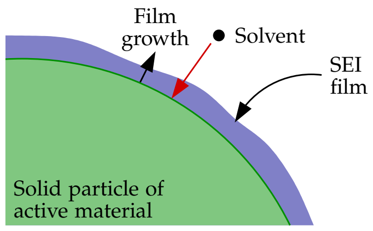


Figure 4.5: Growth of a SEI film.

The side-reaction that produces SEI film consumes lithium while creating the film products. Therefore, cell total capacity decreases due to SEI growth. The SEI film is porous enough to allow intercalation and deintercalation of lithium to and from the graphite, but decreases the conductivity of ion transfer and hence increases cell resistance. Therefore, SEI growth leads to both capacity fade and power fade.

The exact nature of the SEI layer is complicated and not completely understood. It is suspected that numerous reaction products form, then decompose, and then combine into more stable products. However, we do know that once lithium is consumed by SEI growth it is never returned to a form that enables cycling. That is, once capacity is lost to grow SEI it is permanently lost.

While SEI grows fastest during the formation cycle, it continues to build over time. Anything that exposes graphite to solvents in the electrolyte will cause SEI growth. For example, while the SEI film tends to impede the solvent from reaching the graphite surface, the film has enough porosity that some solvent continues to permeate the film and contact the surface of the particle. When this happens, more SEI forms, and the SEI layer grows. High temperatures can lead to reactions that break down the SEI layer, which can lead to new SEI forming on the newly exposed graphite.

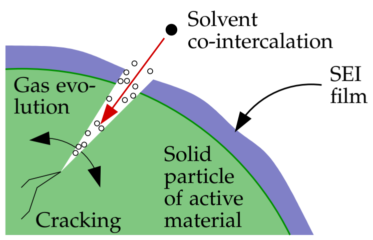


Figure 4.6: Gas generation and cracking due to solvent co-intercalation.

Charging at high rates can force solvent to co-intercalate into the graphite along with lithium and so the SEI reaction can take place *inside* a graphite particle. When this happens, gasses generated during SEI formation cause expansive pressures to build up inside the particle, which tend to crack it along internal grain boundaries or to flake off layers (a process termed *exfoliation*). Both of these expose

12:07:34.

more fresh graphite to solvent in the electrolyte, leading to more SEI formation. This is illustrated in Fig. 4.6.

Trace water in the electrolyte combines with ionized fluorine from the electrolyte salt $LiPF_6$ to form hydrofluoric acid, HF. This acid attacks the SEI, thinning it and allowing more solvent to contact the graphite, forming more SEI. The acid can also accelerate positive-electrode degradation (as we will see in Sect. 4.3.1), leading to dissolved ionized metals such as manganese or cobalt in the electrolyte. These can become part of products comprising the SEI layer when they propagate through the separator to the negative-electrode region of the cell, a process known as *anode poisoning*. The SEI products formed by these metals tend not to have high electronic conductivity and so increase cell resistance. They can also plug pores that would otherwise be used for lithium, preventing lithium cycling and causing capacity fade. These two mechanisms are illustrated in Fig. 4.7.

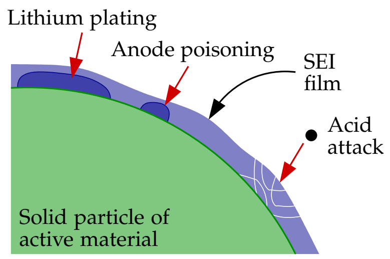


Figure 4.7: Acid attack, anode poisoning, and lithium plating.

A final surface effect that we consider is that of *lithium plating*. This side reaction can cause severe capacity loss and is most acute at cold temperatures where diffusion of lithium in the solid particles is slower.[6] If charging is forced, the local particle surface overpotential can reach a level that causes lithium ions from the electrolyte to join with electrons from the external circuit and plate solid lithium metal on the surface of the particle (this happens when the surface solid–electrolyte potential difference drops below 0 V). Capacity is irreversibly lost. The lithium metal tends to further catalyze SEI growth and forms a metallic annealing site that promotes growth of metal dendrites, which can penetrate the separator and eventually lead to a cell short circuit. This is also illustrated in Fig. 4.7.

[6] For this reason, great care should be taken when charging a lithium-ion cell at cold temperatures. Many manufacturers do not recommend charging a cell at ambient temperatures below about 0 °C.

4.2.2 *Negative electrode aging in bulk*

Charging and discharging lithium-ion cells increases and decreases the amount of lithium present in the negative-electrode active particles. Lithiation causes stresses that tend to lead to an increase in volume; delithiation tends to decrease volume. In graphite, this volume change is modest, usually less than 10 %.[7]

[7] In silicon, a very-high-capacity possible graphite replacement in future lithium-ion cells, the volume changes can be more than 300 %!

Over time, these volume changes can lead to cracking and splitting of particles along internal grain boundaries. As mentioned in Sect. 4.2.1, solvent co-intercalation and gas formation can also split particles. In graphite electrodes, this will cause more SEI to form on the exposed graphite. Alternately, the volume changes can crack only the SEI layer, which will expose more surface graphite and lead to more SEI formation.

12:07:34.

4.2.3 *Negative electrode aging in composite electrode*

Within the composite negative electrode of a lithium-ion cell, conductive additives such as carbon black are often added to improve the electrode's electronic conductivity and binders such as PVdF are added to help maintain contact between particles. These inactive components are not often mentioned when describing the parts of a lithium-ion cell as they do not take part in charge and discharge operations. However, they are critical to the proper functioning of the cell, and a great deal of care is taken when designing a cell to arrive at a good ratio of the inactive materials to the active materials in the electrode.[8] Fig. 4.8 illustrates the structure of the electrode, including the binders and conductive additives that coat the particles.

[8] See, for example, Y-H Chen, C-W Wang, G. Liu, X-Y Song, V.S. Battaglia, and A.M. Sastry, "Selection of Conductive Additives in Li-Ion Battery Cathodes, A Numerical Study," *Journal of the Electrochemical Society*, 154(10), 2007, pp. A978–986.

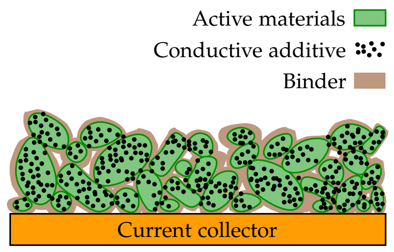


Figure 4.8: Composite electrode structure.

When lithiating and delithiating the active materials in an electrode, stresses leading to deformations can cause the binder to fail, leading to mechanical and electronic contact loss between the graphite particles themselves, between the particles and the current collector, between the binder and the particles, and between the binder and the current collector. This results in higher cell resistance as fewer pathways are available for electrons to flow through the electrode matrix. It can also lead to capacity loss if particles become completely disconnected electronically from the current collectors.

Porosity of the electrode can be reduced by volume changes and by the evolution of the SEI layer, which grows into the space normally occupied by the electrolyte. This impedes movement of lithium ions through the electrolyte and increases cell resistance.

If a cell becomes overdischarged, the open-circuit potential of its graphite material can increase to the point where copper in the negative-electrode current collector corrodes, releasing Cu^{2+} into the electrolyte. This has several consequences. First, there is reduced current-collector/electrode contact, which leads to higher cell resistance. Second, corrosion products that deposit on electrode particles have poor electronic conductivity, which increases SEI film resistance and hence overall cell resistance. Third, the corroded current collector has uneven resistance, which can lead to inhomogeneous current and potential distributions across the cell plate area, resulting in accelerated aging in parts of the cell and a preference toward lithium plating. Finally, copper plating on the negative-electrode particles also make metallic annealing sites that can accelerate lithium plating, dendrite growth, and hence short circuits.

The negative-electrode degradation mechanisms are summarized in Table 4.1. The entries highlighted with a bold font are considered to be the more serious. In particular, we note that some mechanisms primarily cause power fade while others primarily cause capacity fade. Therefore, changes in cell resistance and total capacity are not necessarily proportional, depending on exactly how the cell has aged.

Table 4.1: Principal aging mechanisms in the negative electrode (adapted from Table 1 in Vetter et al., "Ageing Mechanisms in Lithium-Ion Batteries," *Journal of Power Sources*, 147, 2005, 269–281)

Cause	Effect	Leads to	Enhanced by
Continuous low-rate electrolyte decomposition reaction builds SEI	Lithium loss, impedance rise	**Capacity fade, Power fade**	High temperatures, high cell SOC
Solvent co-intercalation, gas evolution and subsequent graphite exfoliation	Loss of active material, lithium loss	Capacity fade	Overcharge
Decrease of accessible surface area due to continuous SEI growth	Impedance rise	**Power fade**	High temperatures, high cell SOC
Changes in porosity due to volume change and SEI growth	Impedance rise, larger overpotentials	Power face	High cycling rate, high cell SOC
Contact loss of active material particles due to volume changes during cycling	Loss of active material	**Capacity fade**	High cycling rate, low cell SOC
Decomposition of binder	Lithium loss, loss of mechanical stability	Capacity fade	High cell SOC, high temperatures
Current collector corrosion	Larger overpotentials and impedance; Inhomogeneous distribution of current and potential	Power fade, Enhances other aging mechanisms	Overdischarge, low cell SOC
Metallic lithium plating and subsequent electrolyte decomposition by metallic lithium	Lithium loss (electrolyte loss)	Capacity fade (power fade)	Low temperature, high charge rates, geometric misfits

4.3 Positive-electrode aging

As with the negative electrode, aging occurs in three locations in the positive electrode: at the surface of the particles, within the active material particles themselves, and in the bulk positive electrode. We discuss these mechanisms in the next three subsections.

4.3.1 Positive electrode aging at surface of particles

In the positive electrode, researchers have found that a film can grow on the surface of the active-material particles as well. In part, this is due to a chemical reaction between the solvent in the electrolyte and the positive-electrode active materials; however, this mechanism is not as pronounced as it is in negative electrodes.

A bigger factor is the dissolution of metals from the electrode crystal structures into the electrolyte and products formed from these metals that can reprecipitate onto the particle surface as a high-resistance film. This dissolution is accelerated by hydrofluoric acid in the electrolyte, initiated by trace amounts of water that combine with the $LiPF_6$ salt.

Metal dissolution via acid attack is a primary cause of capacity loss for lithium-manganese-oxide cells as the loss of manganese destroys the crystal structure and eliminates lithium storage sites.[9] Lithium-cobalt-oxide cells also lose capacity due to cobalt loss, but at a slower rate. The actual mechanism depends on which oxide is used in the positive electrode but tends to happen predominantly at low or high cell states of charge and can be accelerated greatly by high temperature and by any HF acid that might be dissolved in the electrolyte.

[9] See, for example, Dai, Y., Cai, L., and White, R.E., "Capacity Fade Model for Spinel $LiMn_2O_4$ Electrode," *Journal of the Electrochemical Society*, 160(1), 2013, A182–A190.

A side effect of metal dissolution is that the ionized metals can migrate across the separator and poison the negative-electrode, as mentioned in Sect. 4.2.1. This increases cell resistance and lowers total capacity.

4.3.2 Positive electrode aging in bulk

When lithium intercalates into and deintercalates out of the positive-electrode active particles, stresses cause strains known as *phase transitions* that distort the shape of the crystal structure of the electrode materials without changing the overall structure itself. Transitions in phase are caused by the presence or absence of lithium in the storage sites, leading to different local molecular forces. Some of these phase transitions are normal and reversible, but others lead to collapse of the electrode structure and rapid capacity decrease due to the resulting loss of lithium storage sites. This is most common when overcharging a cell: too much lithium is removed from the positive electrode, which causes the lithium pathways to collapse.

These cycling stresses can also lead to a phenomena known as *structural disordering* where the crystal structure of the electrode materials breaks down. Chemical bonds between atoms in the crystal are broken and then later reform to different atoms. This collapses the tunnel-like structures that allow lithium movement, which can cause lithium to become trapped within the crystal structure and also the loss of lithium storage sites. Both of these effects decrease the total capacity of the cell.

In some chemistries, phase transitions near the surface have been observed to lead to the formation of permanent subsurface layers that do not allow lithium to move as freely as in the unaltered crystal structure. This increases the resistance of the cell.

Finally, particles in lithium-iron-phosphate positive electrodes have been observed to grow over time as adjacent particles apparently sinter together. This reduces the total surface area of the electrode, and results in a higher cell resistance.

4.3.3 *Positive electrode aging in the composite electrode*

The composite positive electrode experiences degradation in the same ways as the composite negative electrode. Over time, the binder can decompose, the conductive additives can become oxidized, the current collector can become corroded, and contact among particles and between particles and the current collector can become lost due to volume changes.

THE POSITIVE-ELECTRODE DEGRADATION MECHANISMS are summarized in Table 4.2. While the dominant degradation in the negative electrode tends to be due to side reactions that grow the SEI layer, the dominant mechanisms in the positive electrode tend to be those that result in material loss. Both result in decreased cell total capacity and increased cell resistance.

4.4 *Sensitivity of voltage to R_0*

Having discussed the dominant aging mechanisms in a lithium-ion cell from a qualitative point of view, we now turn our attention to the task of estimating the present state of health of the battery cell. Based on our observations to this point, we see that being able to estimate the cell's present total capacity and equivalent-series resistance will go a long way toward a good description of the cell's present health.

The task of estimating total capacity and resistance must some-

Cause	Effect	Leads to	Enhanced by
Phase transitions	Cracking of active particles	Capacity fade	High rates, high/low SOC
Structural disordering	Lithium sites lost and lithium trapped	Capacity fade	High rates, high/low SOC
Metal dissolution and/or electrolyte decomposition	Migration of soluble species,	Capacity fade	High/low SOC, high temp-erature
	Re-precipitation of new phases,	Power fade	
	Surface layer formation	Power fade	
Electrolyte decomposition	Gas evolution		High temp-erature
Binder decomposition	Loss of contact	Power fade	
Oxidation of conductive agent	Loss of contact	Power fade	
Corrosion of current collector	Loss of contact	Power fade	High SOC

Table 4.2: Principal aging mechanisms at positive electrode.

how use input/output (current/voltage) data from the cell.[10] As a measure of how readily a good estimate may be made, we can then consider the sensitivity of the cell voltage signal to changes in resistance and capacity. This will give us a feel for how easy or difficult it is to estimate these quantities and will reveal that voltage is very sensitive to a change in resistance but very insensitive to a change in total capacity. This result will motivate a simple way to estimate cell resistance; however, we will need to investigate more complex methods in order to make good total-capacity estimates.

Estimating a cell's equivalent series resistance turns out to be relatively simple because it is highly observable from voltage measurements. To see this, consider the cell's voltage equation,

$$v_k = \text{OCV}(z_k) + Mh_k + M_0 s_k - \sum_i R_i i_{R_i,k} - i_k R_0.$$

We define the unitless *sensitivity* of the voltage measurement to a change in resistance as

$$S_{v_k}^{R_0} = \frac{R_0}{v_k} \frac{\mathrm{d}v_k}{\mathrm{d}R_0}$$

$$= \frac{-R_0}{v_k} i_k.$$

A cell's resistance is usually on the order of milliohms, and its voltage is usually a few volts. When multiplied by cell current i_k, which can be large, the absolute sensitivity of voltage to R_0 is relatively high (a magnitude of 0.01 would not be out of the ordinary). This implies that it should be fairly simple to estimate resistance from the voltage signal.

One approach to estimating R_0 is to subtract voltages at two adjacent time samples

$$\begin{array}{rl} v_k = & \text{OCV}(z_k) + Mh_k + M_0 s_k - \sum_i R_i i_{R_i,k} - i_k R_0 \\ v_{k-1} = & \text{OCV}(z_{k-1}) + Mh_{k-1} + M_0 s_{k-1} - \sum_i R_i i_{R_i,k-1} - i_{k-1} R_0 \\ \hline v_k - v_{k-1} \approx & R_0\,(i_{k-1} - i_k) + M_0(s_k - s_{k-1}), \end{array}$$

where we use our knowledge that that cell state of charge z_k, diffusion currents $i_{R_i,k}$, and analog hysteresis h_k change relatively slowly compared to how quickly i_k changes.

So, we can estimate

$$\widehat{R}_{0,k} = \frac{(v_k - v_{k-1}) - M_0\,(s_k - s_{k-1})}{i_{k-1} - i_k}. \tag{4.1}$$

However, when using this method we run into an immediate divide-by-zero problem when $\Delta i_k = i_{k-1} - i_k = 0$, as would happen during a constant-current event or during cell rest. So, we skip updates to $\widehat{R}_{0,k}$ when $|\Delta i_k|$ is small, which eliminates the divide-by-zero problem and avoids amplification of measurement and approximation noise in Eq. (4.1) as well.

Because of the imperfect fidelity of the ESC cell model with respect to the true cell behavior and because of the inaccuracy introduced via our specific approximations, the estimate of $\widehat{R}_{0,k}$ from Eq. (4.1) is quite noisy. To make a better estimate of R_0, we can filter the $\widehat{R}_{0,k}$ signal. For example, we might implement the one-pole digital filter

$$\widehat{R}_{0,k}^{\text{filt}} = \alpha \widehat{R}_{0,k-1}^{\text{filt}} + (1-\alpha)\widehat{R}_{0,k}, \tag{4.2}$$

where $0 \ll \alpha < 1$. This method, while very simple, tends to work quite well. Later in this chapter we will see a second method, which is based on nonlinear Kalman filtering, that also works well.

Another factor that merits consideration is that a cell's resistance is both SOC dependent and temperature dependent. If a scalar resistance estimate is updated, it will adapt to model the cell resistance at the present SOC and temperature as both of those change over time. If an adaptive function that describes the entire relationship of resistance versus state of charge is required instead of an adaptive scalar, then a functional form will need to be proposed and the coefficients of the form will need to be adapted based on the present

[10] Here, we focus on passive methods that simply monitor current and voltage and perform computations using these measurements but do not inject signals into the cell. It is also possible to modulate the input current to a cell purposefully using sinusoidal or pulse signals to measure indicators of health more directly. If the battery pack is permanently connected to the load, the passive approach tends to be preferred since the battery state is not altered by the health measurement, but if the battery pack can be disconnected from the load for diagnostics (even for a short period of time), the latter approach is also a possibility. Note, however, that the sinusoidal response of a battery can be computed from the discrete Fourier transform of passively measured data so long as the cell is *persistently excited*, and the pulse responses can be measured opportunistically when balancing circuits are activated and deactivated, so passive approaches are not as limited as it might appear at first.

state of charge, temperature, and operating resistance. For example, temperature dependence can often be well modeled using an Arrhenius-inspired relationship

$$R_0 = R_{0,\text{ref}} \exp\left(\frac{E_{R_0,\text{ref}}}{R}\left(\frac{1}{T_\text{ref}} - \frac{1}{T}\right)\right),$$

where only the $R_{0,\text{ref}}$ and $E_{R_0,\text{ref}}$ values need to be adapted online to fix the temperature dependence. However, care must be taken in crafting the adaptation algorithm if the battery pack dwells near one temperature for an extended period of time, because adaptation can overcorrect the predictions at that temperature and cause predictions for resistance at other temperatures to become biased.

Temperature and SOC dependence can also be handled by modeling resistance using local basis functions. Adaptation of the parameters of these basis functions will cause only local updates to the resistance relationship and will not bias predictions of resistance at other temperature and state of charge operating points. However, operational points that are not often visited by the application will not have their resistance updated frequently and the estimates will tend to become out of date over time.

4.5 *Code to estimate R_0*

In this section, we see how to implement the simple equivalent-series-resistance estimator we have just seen in MATLAB. In the code, we first load a datafile comprising voltage and current data measured from a cell in the laboratory. This particular cell has a known $R_0 = 2.53\,\text{m}\Omega$ from the system identification procedure discussed in Chap. 2. It is our goal to estimate this value of R_0 using the simple estimator.

```
%% Initialize "truth" values
R0 = 2.53e-3; % from external system ID of cell
R0_Vect = R0*ones(1,length(voltage));

%% Load data file
load('Resistance_data.mat');
```

For the example we consider here, the profiles of current and voltage versus time are plotted in Fig. 4.9.

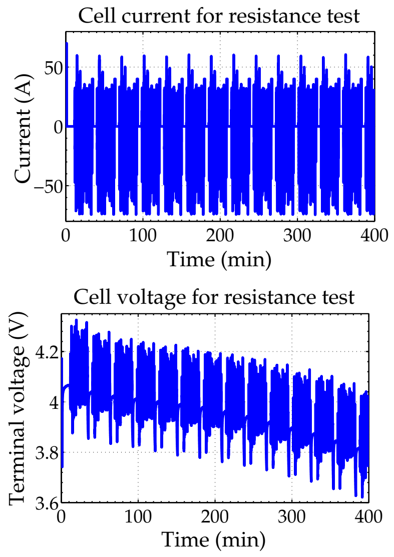


Figure 4.9: Current and voltage data used to estimate cell resistance.

Next, we compute and plot the unfiltered resistance estimate. The threshold on $|\Delta i_k|$ was chosen to be 16.5 A in this example, which is approximately equal to the 2C rate for this cell. The code simply loops through all data, evaluating Eq. (4.1) for each time step and updating the resistance estimate if the threshold criterion is met:

```
%% Estimate R0 - unfiltered
threshold = 16.5;   % Define threshold
```

```
R0_hat = 0*R0_Vect; % Reserve memory for estimate

for k = 2:length(v)
  num = voltage(k) - voltage(k-1);
  den = current(k-1) - current(k);
  R0_hat(k) = num / den;
  if abs(den < threshold),
    R0_hat(k) = R0_hat(k-1);
  end
end

% Plot unfiltered R0 estimate
figure(1); plot(time/60,1000*R0_hat,time/60,1000*R0_Vect);
title('R0 estimate (unfiltered)'); grid on;
xlabel('Time (min)'); ylabel('R0 values (m\Omega)'); ylim([0 4]);
legend('Estimate','Actual','location','southeast');
```

Next, we filter the estimate using $\alpha = 0.999$. The code loops through all raw estimates, applies Eq. (4.2) to each point, and plots the final results:

```
%% Estimate R0 - filtered version
alpha = 0.999;        % define filter pole
R0_hat_filt = R0_hat; % initialize filtered estimate

for k = 2:length(voltage)
  R0_hat_filt(k) = alpha*R0_hat_filt(k-1) + (1-alpha)*R0_hat(k);
end

% Plot filtered R0 estimate
figure(2); plot(time/60,1000*R0_hat_filt,time/60,1000*R0_Vect);
title('R0 estimate (filtered)'); grid on;
xlabel('time (min)'); ylabel('R0 values (m\Omega)'); ylim([0 4]);
legend('Estimate','Actual','location','southeast');
```

Fig. 4.10 shows some results from running this program. The top-left frame compares the unfiltered estimate at every time step to the true value. We see that the estimate has reasonable average value, but that it is very noisy. The top-right frame shows a zoom into one portion of the data segment to highlight some detail. In particular, the places where the estimate flattens out are the result of the $|\Delta i_k|$ threshold criterion not being met, which causes the estimates to retain their prior values.

The lower-left frame shows the filtered version of the estimate versus the truth. Because the filtered estimate is initialized to zero, it takes some time to converge to the neighborhood of the true value of resistance. A better initial guess of the true resistance will lead to faster convergence of the filtered estimate. The lower-right frame shows the unfiltered and filtered estimates compared to each other. We see that the filter has greatly reduced the amount of noise in the estimate. After convergence, the estimate stays within about 7.5 % of the true value, which is very nice performance for such a simple algorithm.

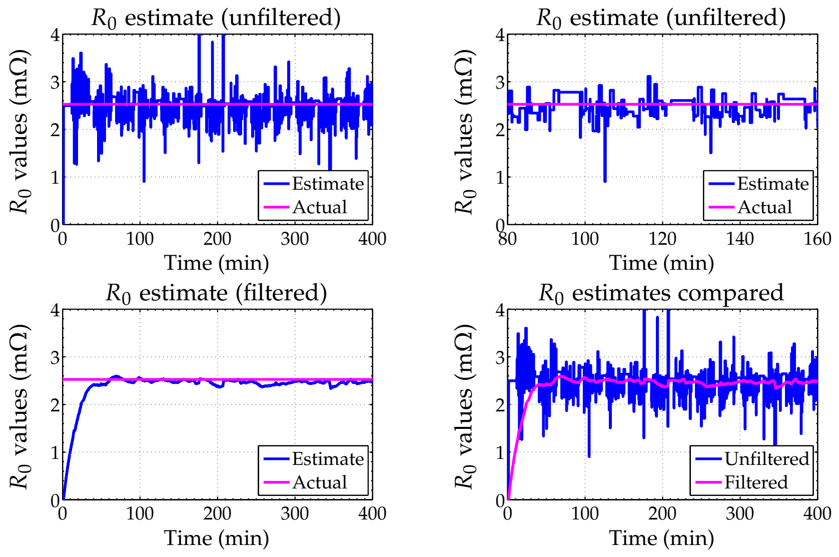


Figure 4.10: Intermediate and final resistance estimates.

4.6 *Sensitivity of voltage to total capacity Q*

Estimating total capacity well turns out to be very difficult. To see why, consider the sensitivity of the voltage measurement to a cell's total capacity

$$S_{v_k}^{Q} = \frac{Q}{v_k}\frac{\mathrm{d}v_k}{\mathrm{d}Q}$$
$$= \frac{Q}{v_k}\frac{\mathrm{d}}{\mathrm{d}Q}\left(\mathrm{OCV}(z_k) + Mh_k + M_0 s_k - \sum_i R_i i_{R_i,k} - i_k R_0\right).$$

While Q/v_k is a reasonably large value, the term inside the parenthesis does not appear to contain capacity at all! To uncover the voltage's sensitivity to total capacity, we need to unfold the total derivatives.

Consider the first term, which we evaluate using the chain rule of total derivatives to be

$$\frac{\mathrm{dOCV}(z_k)}{\mathrm{d}Q} = \frac{\partial \mathrm{OCV}(z_k)}{\partial z_k}\frac{\mathrm{d}z_k}{\mathrm{d}Q}.$$

For most cells, the slope of the open-circuit-voltage curve is very shallow, so $\partial\mathrm{OCV}(z_k)/\partial z_k$ is very small. Further, we expand the total derivative of the SOC equation:

$$\frac{\mathrm{d}z_k}{\mathrm{d}Q} = \frac{\mathrm{d}z_{k-1}}{\mathrm{d}Q} - \eta_{k-1} i_{k-1}\Delta t\frac{\mathrm{d}(1/Q)}{\mathrm{d}Q}$$
$$= \frac{\mathrm{d}z_{k-1}}{\mathrm{d}Q} + \frac{\eta_{k-1} i_{k-1}\Delta t}{Q^2}.$$

The first term on the right-hand side of this result can be calculated recursively. It grows in magnitude when i_k has the same sign for a considerable number of consecutive iterations and shrinks when i_k changes direction. For random i_k (e.g., for a hybrid-electric-vehicle application) it tends to be about the same order of magnitude as the second term on the right-hand side.

The second term has Q^2 in the denominator. Since Q must be expressed in coulombs for its units to agree with the other terms of the equation, Q^2 is a very large number, making the second term very small. So, $\mathrm{d}z_k/\mathrm{d}Q$ is small and overall sensitivity of voltage to capacity through the OCV term is very small.

Similarly, the sensitivity of the voltage to capacity through the analog hysteresis term is small (it is zero through the other terms). As a consequence, individual voltage measurements have very little information regarding capacity. We must somehow combine many voltage measurements to estimate total capacity well, and it helps if current has been generally in the same direction for most of the period spanning those measurements. That is, a large change in SOC over an interval makes it easier to estimate total capacity using the data collected during that interval than does a small change in SOC.

Generally speaking, simple methods like the one used to estimate $\widehat{R}_0$ will not work well. So, in the remainder of this chapter, we explore two different and more complicated approaches. First, we look at using a nonlinear Kalman filter, which can work well, and which has the additional benefit of being able to estimate all cell-model parameter values simultaneously as they change over time. However, these Kalman-filter-based approaches are relatively complex and there are some algorithm stability issues that must be addressed before the estimates can be trusted. Simpler regression techniques can be used to estimate total capacity, but the most straightforward ordinary least-squares approach gives biased results. So, we will also spend considerable time exploring a total-least-squares approach, which uses the noisy input measurements in an optimal way to produce unbiased total-capacity estimates.

4.7 *Estimating parameters via Kalman filters*

In Chap. 3, we saw that a Kalman filter can be used to estimate the state of a dynamic system using noisy input/output measurements if the model parameter values are known precisely. It turns out that it is also possible to use a nonlinear Kalman filter to estimate the parameters of a dynamic system's model using noisy input/output measurements if its state is known precisely. Because the SOH of a battery cell is summarized by the cell-model parameter values, this approach promises to be very useful.

Unfortunately, we do not know the state precisely. However, if we are careful, we can also use a nonlinear Kalman filter to estimate *both* the state and the parameters of the system model simultaneously using noisy input/output measurements.

In this section, we proceed by first considering how to estimate the parameter values of a system if its state is known. Then, we combine the state-estimation and parameter-estimation methods to show how to estimate both state and parameter simultaneously.[11]

[11] The EKF presentation is based on Plett, G., "Extended Kalman Filtering for Battery Management Systems of LiPB-Based HEV Battery Packs—Part 3: State and Parameter Estimation," *Journal of Power Sources*, 134(2), 2004, pp. 277–92, and the SPKF presentation is based on Plett, G., "Sigma-Point Kalman Filters for Battery Management Systems of LiPB-Based HEV Battery Packs—Part 2: Simultaneous State and Parameter Estimation," *Journal of Power Sources*, 161(2), 2006, pp. 1369–84.

4.7.1 *A generic approach to parameter estimation*

We begin by collecting the true parameters of a particular model into the vector $\boldsymbol{\theta}$. We will use Kalman-filtering techniques to estimate the parameter values as we did to estimate the state-vector values. Therefore, we require a discrete-time state-space model of the dynamics of the parameters.

By assumption, parameter values change very slowly, so we model the present parameter vector as being equal to the prior parameter vector plus some small perturbation $\boldsymbol{r}_k$

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \boldsymbol{r}_{k-1}. \tag{4.3}$$

The small zero-mean white-noise input $\boldsymbol{r}_k$ is fictitious, and is used in Eq. (4.3) to model a forcing input for the slow drift in the parameter values of the system. We do not ever synthesize this noise as an input to the Kalman filter; instead, the covariance matrix $\boldsymbol{\Sigma}_{\tilde{r}}$ is used to indicate how quickly we believe the parameter values can change, which causes the Kalman filter to adjust how quickly it updates parameter value estimates.

The model of Eq. (4.3) is known as a *random walk*. In the absence of feedback, it describes a path for the parameter values that comprises a sequence of purely random steps, where the variance—and hence the uncertainty—of $\boldsymbol{\theta}_k$ increases linearly over time. In the model, the parameter values are just as likely to increase as they are to decrease in value from one time step to another.

Based on the qualitative understanding of cell degradation that we have gained so far in this chapter, we recognize that the random walk is probably not a good description of how cell parameter values actually change. In a random walk, a parameter value is just as likely to increase as it is to decrease. However, if we make a bold generalization, we can state that a real cell's total capacity only decreases with time; it does not increase.[12] Similarly, its resistance only increases with time; it does not decrease.[13]

[12] Over the first few cycles of a lithium-ion battery cell, the total capacity often *does* increase, as the electrolyte seeps into voids it had not properly wet during cell construction. This creates ionic connection to parts of the cell that were previously isolated, and adds their capacity to the total. Also, if a cell has been stored idle for a long period of time, some researchers have noted that subsequent cycling will recover some lost capacity although the mechanism for doing so is not clear.

[13] Again, resistance has been observed to decrease for the first few cycles of a lithium-ion cell before it begins to increase. This is thought to be caused by the formation of micro-cracks on the surface of the active particles due to intercalation stresses exciting internal defects in the new active materials. These cracks increase the surface area of the graphite that is exposed, decreasing resistance. However, the micro-cracks stop forming after the initial cycles, and the general trend of cell resistance over time is in the increasing direction.

While the random walk is not an accurate open-loop model for how parameter values change, to model the changes more descriptively would require a detailed quantitative description of how the degradation mechanisms actually work. There is very little literature on this subject, although we do look at two such models in Chap. 7. It turns out that even though Eq. (4.3) is not a very accurate description of the details of cell degradation, the feedback mechanism of the Kalman filter will correct for the modeling errors and still make good estimates of the model's parameter values.

To incorporate feedback, we require a model output equation, which must be a measurable function of the system parameters. We use

$$\boldsymbol{d}_k = \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k), \tag{4.4}$$

where $\boldsymbol{g}(\cdot)$ is the output equation of the system model being identified, and $\boldsymbol{e}_k$ models the sensor noise and modeling error.[14] Then, the sequence of all measurements made for parameter identification can be written as $\mathbb{D}_k = \{\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$.

[14] Note that $\boldsymbol{d}_k$ is usually the same measurement as $\boldsymbol{y}_k$, but we maintain a distinction here in case separate outputs are used. Also, note that the noises $\boldsymbol{e}_k$ in Eq. (4.4) and $\boldsymbol{v}_k$ in Eq. (4.6) often play the same role, but are considered distinct here.

We also slightly revise the mathematical model of system state-vector dynamics and measurement relationship to be

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{\theta}, \boldsymbol{w}_{k-1}) \tag{4.5}$$

$$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{v}_k), \tag{4.6}$$

which includes the parameters $\boldsymbol{\theta}$ in the model explicitly. Time-invariant numeric values required by the model may be embedded within $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ and are not included in $\boldsymbol{\theta}$.

4.8 EKF parameter estimation

With the new parameter-dynamics equation Eq. (4.3), parameter-output equation Eq. (4.4), revised state equation Eq. (4.5), and model-output equation Eq. (4.6), we are now ready to see how to perform parameter estimation using a nonlinear Kalman filter.

We proceed by first deriving the extended Kalman filter for parameter estimation following the same six steps of sequential-probabilistic inference introduced in Chap. 3.

EKF step 1a: Parameter prediction time update.

The parameter prediction step finds the expected value of the parameter update equation Eq. (4.3), which is

$$\begin{aligned}\hat{\boldsymbol{\theta}}_k^- &= \mathbb{E}[\boldsymbol{\theta}_k \mid \mathbb{D}_{k-1}] \\ &= \mathbb{E}[\boldsymbol{\theta}_{k-1} + \boldsymbol{r}_{k-1} \mid \mathbb{D}_{k-1}] \\ &= \hat{\boldsymbol{\theta}}_{k-1}^+, \end{aligned} \tag{4.7}$$

since we have assumed that the fictitious noise $\boldsymbol{r}_k$ is zero mean. That is, the predicted parameter vector for this time step is equal to the estimated parameter vector computed at the end of the prior time step. This result makes sense because the parameters are assumed to be essentially constant, with very slow variation due to cell aging. We do see, however, that the changes in parameter values due to degradation will be captured only via the measurement feedback because the prediction step in Eq. (4.7) does not forecast any degradation in their values.

EKF step 1b: Error covariance time update.

The covariance matrix of the parameter prediction error is found by first computing $\tilde{\boldsymbol{\theta}}_k^-$, as

$$\begin{aligned}\tilde{\boldsymbol{\theta}}_k^- &= \boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k^- \\ &= \boldsymbol{\theta}_{k-1} + \boldsymbol{r}_k - \hat{\boldsymbol{\theta}}_{k-1}^+ \\ &= \tilde{\boldsymbol{\theta}}_{k-1}^+ + \boldsymbol{r}_k.\end{aligned}$$

We then compute the desired covariance directly as

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{\theta},k}^- &= \mathbb{E}[\tilde{\boldsymbol{\theta}}_k^-(\tilde{\boldsymbol{\theta}}_k^-)^T] \\ &= \mathbb{E}[(\tilde{\boldsymbol{\theta}}_{k-1}^+ + \boldsymbol{r}_k)(\tilde{\boldsymbol{\theta}}_{k-1}^+ + \boldsymbol{r}_k)^T] \\ &= \boldsymbol{\Sigma}_{\tilde{\theta},k-1}^+ + \boldsymbol{\Sigma}_{\tilde{r}},\end{aligned}$$

where we have assumed that the zero-mean fictitious noise is uncorrelated with the parameter prediction error. Therefore, mathematically, the time-updated parameter-error covariance is equal to the covariance prior to the time update, but having additional uncertainty due to the fictitious noise that is assumed to be driving the change in parameter values. Physically, this corresponds to the increasing uncertainty in parameter values over time due to aging unless we can somehow adapt the parameter values via feedback, which is what we will do in step 2b.

EKF step 1c: Output prediction.

The measurable system output based on the parameter model is predicted using Eq. (4.4) and EKF assumption 1 to be

$$\begin{aligned}\hat{\boldsymbol{d}}_k &= \mathbb{E}[\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k) \mid \mathbb{D}_{k-1}] \\ &\approx \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{e}}_k).\end{aligned}$$

That is, we assume that a good way to approximate the mean of the parameter output equation is by propagating the expected parameter vector $\hat{\boldsymbol{\theta}}_k^-$ and the mean estimation error through the parameter output equation.

EKF step 2a: Estimator gain matrix.

12:07:34.

The output prediction error may then be written as

$$\begin{aligned}\tilde{\boldsymbol{d}}_k &= \boldsymbol{d}_k - \hat{\boldsymbol{d}}_k \\ &= \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k) - \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{e}}_k).\end{aligned}$$

We employ EKF assumption 2 to linearize this relationship using a Taylor-series expansion of the first term on the left-hand side of the equation:

$$\begin{aligned}\boldsymbol{d}_k \approx \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{e}}_k) &+ \underbrace{\left.\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k^-}}_{\text{Defined as } \hat{\boldsymbol{C}}_k^\theta} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_k^-) \\ &+ \underbrace{\left.\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{e}_k}\right|_{\boldsymbol{e}_k=\bar{\boldsymbol{e}}_k}}_{\text{Defined as } \hat{\boldsymbol{D}}_k^\theta} (\boldsymbol{e}_k - \bar{\boldsymbol{e}}_k).\end{aligned}$$

From this, we can compute such necessary quantities as:

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{d},k} &\approx \hat{\boldsymbol{C}}_k^\theta \boldsymbol{\Sigma}_{\tilde{\theta},k}^- (\hat{\boldsymbol{C}}_k^\theta)^T + \hat{\boldsymbol{D}}_k^\theta \boldsymbol{\Sigma}_{\tilde{e}} (\hat{\boldsymbol{D}}_k^\theta)^T \\ \boldsymbol{\Sigma}_{\tilde{\theta}\tilde{d},k}^- &\approx \mathbb{E}[(\tilde{\boldsymbol{\theta}}_k^-)(\hat{\boldsymbol{C}}_k^\theta \tilde{\boldsymbol{\theta}}_k^- + \hat{\boldsymbol{D}}_k^\theta \tilde{\boldsymbol{e}}_k)^T] = \boldsymbol{\Sigma}_{\tilde{\theta},k}^- (\hat{\boldsymbol{C}}_k^\theta)^T.\end{aligned}$$

These terms may be combined to compute the Kalman gain

$$\boldsymbol{L}_k^\theta = \boldsymbol{\Sigma}_{\tilde{\theta},k}^- (\hat{\boldsymbol{C}}_k^\theta)^T \big[\hat{\boldsymbol{C}}_k^\theta \boldsymbol{\Sigma}_{\tilde{\theta},k}^- (\hat{\boldsymbol{C}}_k^\theta)^T + \hat{\boldsymbol{D}}_k^\theta \boldsymbol{\Sigma}_{\tilde{e}} (\hat{\boldsymbol{D}}_k^\theta)^T\big]^{-1}.$$

Note the superscript θ used in the notation for $\hat{\boldsymbol{C}}_k^\theta$, $\hat{\boldsymbol{D}}_k^\theta$, and $\boldsymbol{L}_k^\theta$, which is added to keep these EKF matrices for parameter estimation distinct from their counterparts for state estimation. This will be important later when we estimate both states and parameters simultaneously.

We must be very careful when computing $\hat{\boldsymbol{C}}_k^\theta$. By the chain rule of total differentials, we have

$$\begin{aligned}\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k) = &\frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{x}_k}\mathrm{d}\boldsymbol{x}_k + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{u}_k}\mathrm{d}\boldsymbol{u}_k \\ &+ \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{\theta}}\mathrm{d}\boldsymbol{\theta} + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{e}_k}\mathrm{d}\boldsymbol{e}_k.\end{aligned}$$

Dividing both sides of this equation by $\mathrm{d}\boldsymbol{\theta}$, we find

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{x}_k}\frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{u}_k}\underbrace{\frac{\mathrm{d}\boldsymbol{u}_k}{\mathrm{d}\boldsymbol{\theta}}}_{0} \\ &\quad + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{\theta}}\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{e}_k}\underbrace{\frac{\mathrm{d}\boldsymbol{e}_k}{\mathrm{d}\boldsymbol{\theta}}}_{0} \\ &= \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}, \boldsymbol{e}_k)}{\partial \boldsymbol{x}_k}\frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\boldsymbol{\theta}}.\end{aligned}$$

12:07:34.

As with state estimation, some terms drop out since the deterministic cell input current is not a function of cell parameter values, nor is the measurement error e_k. However, by Eq. (4.5), the model state *is* a function of the parameter values and so we must include the $\mathrm{d}x_k/\mathrm{d}\theta$ term in the expansion. This is different from what we found for state-vector estimation in Chap. 3, and is the reason why we had a discussion of total versus partial derivatives at the corresponding point in the derivation of the EKF there. Here, we absolutely must use the total derivative instead of the partial derivative, or else the parameter estimator will not work!

We evaluate this new term recursively.

$$\frac{\mathrm{d}x_k}{\mathrm{d}\theta} = \frac{\partial f(x_{k-1}, u_{k-1}, \theta, w_{k-1})}{\partial \theta} + \frac{\partial f(x_{k-1}, u_{k-1}, \theta, w_{k-1})}{\partial x_{k-1}} \frac{\mathrm{d}x_{k-1}}{\mathrm{d}\theta}.$$

We see that $\mathrm{d}x_k/\mathrm{d}\theta$ is a function of $\mathrm{d}x_{k-1}/\mathrm{d}\theta$, and so this relationship evolves over time as the state evolves. The term $\mathrm{d}x_0/\mathrm{d}\theta$ is initialized to zero unless side information gives a better estimate of its value.

In summary, in order to calculate $\hat{C}_k^{\theta}$ for any specific model structure, we require methods to calculate all of the above partial derivatives for the model to find the needed total derivatives.

EKF step 2b: Parameter estimate measurement update.

The parameter estimate is computed by updating the prediction using the estimator gain and the output prediction error $d_k - \hat{d}_k$:

$$\hat{\theta}_k^+ = \hat{\theta}_k^- + L_k^{\theta}(d_k - \hat{d}_k).$$

This step is unchanged in principle from the standard EKF.

EKF step 2c: Error covariance measurement update.

Finally, the updated parameter estimation-error covariance is computed as

$$\Sigma_{\tilde{\theta},k}^+ = \Sigma_{\tilde{\theta},k}^- - L_k^{\theta}\Sigma_{\tilde{d},k}(L_k^{\theta})^T.$$

EKF for parameter estimation is summarized in the Appendix on p. 229. When starting the algorithm for the first time, we initialize the parameter-vector estimate with our best information regarding its value, $\hat{\theta}_0^+ = \mathbb{E}[\theta_0]$, we initialize the parameter estimation error covariance matrix with our best information regarding our uncertainty in the parameter-vector estimate

$$\Sigma_{\tilde{\theta},0}^+ = \mathbb{E}\big[(\theta - \hat{\theta}_0^+)(\theta - \hat{\theta}_0^+)^T\big],$$

and we initialize $\mathrm{d}x_0/\mathrm{d}\theta = 0$ unless side information is available.

4.9 *SPKF parameter estimation*

Given the time we invested in Chap. 3 studying sigma-point methods, it is also relatively straightforward to show how to estimate parameter values using a sigma-point Kalman filter. As always, we proceed by deriving the six essential steps of sequential probabilistic inference.

SPKF step 1a: Parameter prediction time update.

The parameter prediction step is approximated as

$$\begin{aligned}\hat{\boldsymbol{\theta}}_k^- &= \mathbb{E}[\boldsymbol{\theta}_k \mid \mathbb{D}_{k-1}]\\ &= \mathbb{E}[\boldsymbol{\theta}_{k-1} + \boldsymbol{r}_{k-1} \mid \mathbb{D}_{k-1}]\\ &= \hat{\boldsymbol{\theta}}_{k-1}^+.\end{aligned}$$

This outcome is the same as with EKF since the dynamics of the parameters follow a linear model.

SPKF step 1b: Error covariance time update.

Again, because of the linearity of the parameter dynamics, the error-covariance update equation is the same as for EKF.

$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k-1}^+ + \boldsymbol{\Sigma}_{\tilde{\boldsymbol{r}}}.$$

SPKF step 1c: Predict system output $\boldsymbol{d}_k$.

To predict the system output, we first define the augmented random vector

$$\boldsymbol{\theta}_k^a = \begin{bmatrix} \boldsymbol{\theta}_k \\ \boldsymbol{e}_k \end{bmatrix},$$

which combines the randomness of the parameter estimates and the sensor noise. We then compute a set of $p+1$ sigma points describing this augmented random vector, which we will denote as $\boldsymbol{\mathcal{W}}_k^{a,-}$:

$$\boldsymbol{\mathcal{W}}_k^{a,-} = \left\{ \hat{\boldsymbol{\theta}}_k^{a,-}, \hat{\boldsymbol{\theta}}_k^{a,-} + \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^{a,-}}, \hat{\boldsymbol{\theta}}_k^{a,-} - \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^{a,-}} \right\}.$$

From the augmented sigma points, the $p+1$ vectors comprising the parameters portion $\boldsymbol{\mathcal{W}}_k^{\theta,-}$ and the $p+1$ vectors comprising the modeling error portion $\boldsymbol{\mathcal{W}}_k^{e,-}$ are extracted.

The output equation Eq. (4.4) is evaluated using all pairs of $\boldsymbol{\mathcal{W}}_{k,i}^{\theta,-}$ and $\boldsymbol{\mathcal{W}}_{k,i}^{e,-}$ (where the subscript i denotes that the ith vector is being extracted from the original set), yielding the sigma points $\boldsymbol{\mathcal{D}}_{k,i}$ for time step k. That is,

$$\boldsymbol{\mathcal{D}}_{k,i} = \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\mathcal{W}}_{k,i}^{\theta,-}, \boldsymbol{\mathcal{W}}_{k,i}^{e,-}).$$

Finally, the output prediction is computed as

$$\begin{aligned}\hat{d}_k^- &= \mathbb{E}\big[g(x_k, u_k, \theta, e_k) \mid \mathbb{D}_{k-1}\big] \\ &\approx \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} g(x_k, u_k, \mathcal{W}_{k,i}^{\theta,-}, \mathcal{W}_{k,i}^{e,-}) \\ &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{D}_{k,i}.\end{aligned}$$

SPKF step 2a: Estimator gain matrix L_k^θ.

To compute the estimator gain matrix, we must first compute the required covariance matrices.

$$\begin{aligned}\Sigma_{\tilde{d},k} &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\mathcal{D}_{k,i} - \hat{d}_k\big)\big(\mathcal{D}_{k,i} - \hat{d}_k\big)^T \\ \Sigma_{\tilde{\theta}\tilde{d},k}^- &= \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\mathcal{W}_{k,i}^{\theta,-} - \hat{\theta}_k^{a,-}\big)\big(\mathcal{D}_{k,i} - \hat{d}_k\big)^T.\end{aligned}$$

Then, we simply compute $L_k^\theta = \Sigma_{\tilde{\theta}\tilde{d},k}^- \Sigma_{\tilde{d},k}^{-1}$.

SPKF step 2b: Parameter estimate measurement update.

The parameter estimate is created from the predicted parameters and the measurement innovation using the equation from the optimal formulation

$$\hat{\theta}_k^+ = \hat{\theta}_k^- + L_k^\theta (d_k - \hat{d}_k).$$

SPKF step 2c: Error covariance measurement update.

Finally, the covariance of the parameter estimation error is computed as

$$\Sigma_{\tilde{\theta},k}^+ = \Sigma_{\tilde{\theta},k}^- - L_k^\theta \Sigma_{\tilde{d},k} (L_k^\theta)^T.$$

SPKF for parameter estimation is summarized in the Appendix on p. 230.

4.10 Joint and dual estimation

Having seen how to use Kalman filters to perform state estimation and parameter estimation separately, we now turn our attention to using nonlinear Kalman filters to perform the task of estimating both the state and parameter vectors at the same time if the state-filter output y_k is the same as the parameter-filter output d_k. There are two approaches to doing so: *joint estimation* and *dual estimation*. These are discussed in the next sections.

4.10.1 *Generic joint estimation*

In joint estimation, the states and parameters are combined into a single high-dimension vector, and a nonlinear Kalman filter simultaneously estimates the values within this augmented state vector. Doing so has the disadvantages of large matrix operations due to the high dimensionality of the resulting augmented model and potentially poor numeric conditioning due to the vastly different time scales of the states and parameters in the augmented state vector. However, it is quite straightforward to implement.

To see how to do so, we first combine the state and parameter vectors to form augmented dynamics

$$\begin{aligned}\begin{bmatrix} x_k \\ \theta_k \end{bmatrix} &= \begin{bmatrix} f(x_{k-1}, u_{k-1}, \theta_{k-1}, w_{k-1}) \\ \theta_{k-1} + r_{k-1} \end{bmatrix} \\ y_k &= h(x_k, u_k, \theta_k, v_k).\end{aligned}$$

To simplify notation, let $\mathbf{X}_k$ be the augmented state, $\mathbf{W}_k$ be the augmented noise, and $\mathbf{F}(\cdot)$ be the augmented state equations. Then,

$$\begin{aligned}\mathbf{X}_k &= \mathbf{F}(\mathbf{X}_{k-1}, u_{k-1}, \mathbf{W}_{k-1}) \\ y_k &= h(\mathbf{X}_k, u_k, v_k).\end{aligned}$$

With this augmented discrete-time state-space model of the system state dynamics and parameter dynamics, we simply apply a nonlinear Kalman-filter method.

4.10.2 *Generic dual estimation*

In dual estimation, separate nonlinear Kalman filters are used for state estimation and parameter estimation. The computational complexity is smaller than for joint estimation because smaller matrices are involved in the computations and the matrix operations may be better conditioned numerically. However, by decoupling state from parameters any cross-correlations between the two are lost, leading to the possibility of poorer estimation accuracy.

The mathematical model of state dynamics again explicitly includes the parameters as the vector θ_k

$$\begin{aligned}x_k &= f(x_{k-1}, u_{k-1}, w_{k-1}, \theta_{k-1}) \\ y_k &= h(x_k, u_k, v_k, \theta_{k-1}).\end{aligned}$$

Time-invariant numeric values required by the model may be embedded within $f(\cdot)$ and $h(\cdot)$ and are not included in θ_k. We also slightly revise the mathematical model of parameter dynamics to include the

12:07:34.

effect of the state equation explicitly.

$$\begin{aligned}\boldsymbol{\theta}_k &= \boldsymbol{\theta}_{k-1} + \boldsymbol{r}_{k-1}\\ \boldsymbol{d}_k &= \boldsymbol{h}\left(\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \bar{\boldsymbol{w}}_{k-1}, \boldsymbol{\theta}_{k-1}), \boldsymbol{u}_k, \boldsymbol{e}_k, \boldsymbol{\theta}_{k-1}\right).\end{aligned}$$

The interactions between the two Kalman filters are illustrated in Fig. 4.11, where the details will become clearer in the following subsections. We see from the diagram that the dual-estimation process essentially comprises two Kalman filters running in parallel—with KF$_x$ adapting the state and KF$_\theta$ adapting parameters—with some information exchange between the filters.

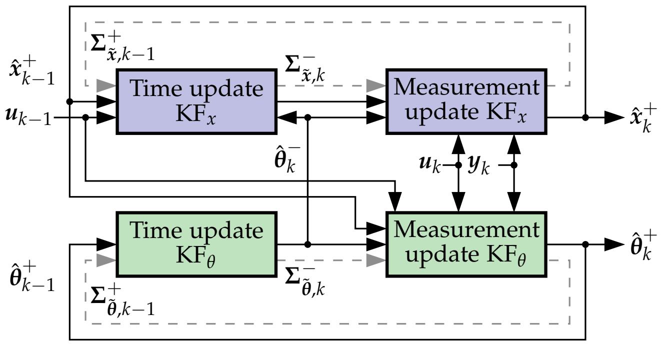

$\boldsymbol{\Sigma}^+_{\tilde{x},k-1}$
$\hat{\boldsymbol{x}}^+_{k-1}$
$\boldsymbol{u}_{k-1}$
$\boldsymbol{\Sigma}^-_{\tilde{x},k}$
Time update KF$_x$
Measurement update KF$_x$
$\hat{\boldsymbol{x}}^+_k$
$\hat{\boldsymbol{\theta}}^-_k$
$\boldsymbol{u}_k$
$\boldsymbol{y}_k$
Time update KF$_\theta$
Measurement update KF$_\theta$
$\hat{\boldsymbol{\theta}}^+_k$
$\hat{\boldsymbol{\theta}}^+_{k-1}$
$\boldsymbol{\Sigma}^-_{\tilde{\theta},k}$
$\boldsymbol{\Sigma}^+_{\tilde{\theta},k-1}$

Figure 4.11: Interactions between the two nonlinear Kalman filters used for dual estimation.
(Reproduced from Fig. 8 of Plett, G.L., "Extended Kalman Filtering for Battery Management Systems of LiPB-Based HEV Battery Packs—Part 3: State and Parameter Estimation," *Journal of Power Sources*, 134(2), 2004, pp. 277–92.)

With generic joint and dual estimation defined, we now show how to use these approaches with EKF and SPKF.

4.10.3 *Joint state and parameter estimation via EKF*

Applying EKF to the joint-estimation problem is straightforward. But, we must be careful to evaluate the recursive calculation of $\mathrm{d}\mathbf{F}/\mathrm{d}\mathbf{X}$ correctly when computing the $\hat{C}_k$ matrix. The method is summarized in the Appendix on p. 233.

4.10.4 *Dual state and parameter estimation via EKF*

When implementing dual state and parameter estimation using EKF, two EKFs are implemented with their signals mixed. Again, we need to be careful when computing $\hat{C}^\theta_k$, which requires a total-differential expansion to be correct:

$$\hat{C}^\theta_k = \left.\frac{\mathrm{d}\boldsymbol{g}(\hat{\boldsymbol{x}}^-_k, \boldsymbol{u}_k, \boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}^-_k}$$

$$\frac{\mathrm{d}\boldsymbol{g}(\hat{\boldsymbol{x}}^-_k, \boldsymbol{u}_k, \boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \boldsymbol{g}(\hat{\boldsymbol{x}}^-_k, \boldsymbol{u}_k, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{g}(\hat{\boldsymbol{x}}^-_k, \boldsymbol{u}_k, \boldsymbol{\theta})}{\partial \hat{\boldsymbol{x}}^-_k}\frac{\mathrm{d}\hat{\boldsymbol{x}}^-_k}{\mathrm{d}\boldsymbol{\theta}}$$

12:07:34.

$$\begin{aligned}
\frac{\mathrm{d}\hat{x}_k^-}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\partial f(\hat{x}_{k-1}^+, \boldsymbol{u}_{k-1}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial f(\hat{x}_{k-1}^+, \boldsymbol{u}_{k-1}, \boldsymbol{\theta})}{\partial \hat{x}_{k-1}^+} \frac{\mathrm{d}\hat{x}_{k-1}^+}{\mathrm{d}\boldsymbol{\theta}} \\
\frac{\mathrm{d}\hat{x}_{k-1}^+}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\mathrm{d}\hat{x}_{k-1}^-}{\mathrm{d}\boldsymbol{\theta}} - \boldsymbol{L}_{k-1}^x \frac{\mathrm{d}g(\hat{x}_{k-1}^-, \boldsymbol{u}_{k-1}, \boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}}.
\end{aligned}$$

In the notation, $\boldsymbol{L}_k^x$ is the Kalman gain for the state filter, which is assumed not to be a function of $\boldsymbol{\theta}$. In fact, it *is* a weak function of $\boldsymbol{\theta}$, but the consensus of the literature is that the dependence is weak enough that the simplifications achieved by the assumption outweigh any gains that might be made by a more careful analysis.

To implement this method, the three total derivatives $\mathrm{d}g/\mathrm{d}\boldsymbol{\theta}$, $\mathrm{d}\hat{x}_{k-1}^+/\mathrm{d}\boldsymbol{\theta}$, and $\mathrm{d}\hat{x}_k^-/\mathrm{d}\boldsymbol{\theta}$ are initialized to zero and computed recursively as the filters operate. The method is summarized in the Appendix on p. 233.

4.10.5 Joint state and parameter estimation via SPKF

Joint state and parameter estimation using sigma-point Kalman filters uses the standard SPKF method where the state vector is augmented with the parameters. No other changes are made. The method is summarized in the Appendix on p. 233.

4.10.6 Dual state and parameter estimation via SPKF

Dual state and parameter estimation using SPKF, just like dual estimation using EKF, uses two filters. Both employ the SPKF algorithm and intermix signals. The method is summarized in the Appendix on p. 234.

4.11 Robustness and speed

4.11.1 Ensuring correct convergence

Dual and joint filtering adapt the state estimate $\hat{x}_k^+$ and the parameter estimate $\hat{\boldsymbol{\theta}}_k^+$ so that the model input/output relationship matches the measured input/output data closely. However, there is no built-in guarantee that the state of the model converges to anything with physical meaning. It is possible for the state and parameter estimates to diverge from their true values and, because of cancellations between errors caused by inaccurate state and parameter estimates, still have good agreement with input/output measurements, at least for a time.

When estimating the ESC cell-model state and parameter vectors, we are concerned that the they converge to their true meaning. This will not happen by default with the dual or joint estimation meth-

12:07:34.

ods as described so far. Special steps must be taken to ensure that convergence occurs.

Consider the SOC state z_k inside the model state vector. We may use a very crude cell model to derive a synthetic measurement of this state, as we saw in Chap. 3. Specifically, we can model cell terminal voltage as

$$v_k \approx \text{OCV}(z_k) - R_0 i_k.$$

Then, given a measurement of voltage, we can estimate SOC as

$$\hat{z}_k = \text{OCV}^{-1}(v_k + R_0 i_k).$$

While this result is too noisy to use as the primary state of charge estimate, it still has some nice properties. First, if current is zero and hysteresis is negligible, then the estimate converges to the true state of charge. Second, even if current is not zero, the estimate tends to have very little bias: it is noisy, as we saw in Fig. 3.4, but the noise is close to zero mean.

Therefore, by measuring the cell voltage under load v_k, the cell current i_k, and having knowledge of R_0 and the cell's inverse OCV function, we can compute a noisy estimate $\hat{z}_k$ of SOC. We then augment the true measurement of cell voltage with this synthetic measurement of cell SOC in the output equation of the model, which then becomes:

$$\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}) = \begin{bmatrix} \text{OCV}(z_k) + M h_k + M_0 s_k - \sum_i R_i i_{R_i,k} - R_0 i_k \\ z_k \end{bmatrix}.$$

The dual or joint nonlinear Kalman filter is run using this modified model, with the measurement used in the measurement update replaced by

$$\boldsymbol{y}_k = \begin{bmatrix} \text{cell voltage at time } k \\ \text{crude SOC estimate } \hat{z}_k \end{bmatrix}.$$

While the errors in $\hat{z}_k$ due to ignoring short-term hysteresis bias and diffusion voltages prohibit it from being used as the primary estimator of SOC, its expected long-term behavior in a dynamic environment is accurate, and it maintains the accuracy of the state of charge state in the dual and joint filters.

4.12 Unbiased estimate of total capacity using linear regression

The computational complexity incurred by using a nonlinear Kalman filter to estimate all states and parameters simultaneously is relatively high. If we require estimates of only resistance and total capacity, this complexity is not warranted. The simple estimator of Sect. 4.4

12:07:34.

can be used to estimate resistance. But, how can we then estimate total capacity?

Recall from Sect. 4.6 that the sensitivity of voltage to total capacity is low, so separating capacity information from noisy input/output data is difficult, and the noise easily biases the results. The sensitivity improves when SOC has changed appreciably between updates, so we would expect that iteration-by-iteration updates to be unnecessary and computationally inefficient. Fortunately, capacity changes slowly, so updating total-capacity estimates infrequently and only after significant changes to state of charge occur is adequate.[15]

[15] The remainder of this chapter is based in large part on Plett, G., "Recursive approximate weighted total least squares estimation of battery cell total capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2319–2331.

4.12.1 *The problem with least-squares capacity estimates*

For a fresh look at estimating total capacity, consider again the cell state of charge equation, where we compute a cell's state of charge at time index $k_2 > k_1$ starting with an initial known state of charge at time index k_1

$$z_{k_2} = z_{k_1} - \frac{1}{Q}\sum_{k=k_1}^{k_2-1} \eta_k i_k.$$

We can rearrange the terms in this expression to get

$$\underbrace{-\sum_{k=k_1}^{k_2-1} \eta_k i_k}_{y} = Q\,\underbrace{\left(z_{k_2} - z_{k_1}\right)}_{x},$$

where the obvious linear structure of $y = Qx$ becomes apparent. Using a regression technique, for example, we may compute estimates of Q. We need only to find sets of values for x and y in this equation to do so.[16]

[16] Note that to use this method, coulomb counting cannot be used as the state of charge estimator, since this results in a degenerate equation: we would be comparing coulomb counts on one side of the equation with exactly the same coulomb counts on the other side of the equation. The state of charge estimate must be based—at least in part—on voltage measurements. The Kalman-filter based methods from Chap. 3 for state of charge estimation have enough voltage feedback that they work with this method.

The most common linear regression techniques are based on *ordinary least squares* (*OLS* or simply LS). OLS assumes that the independent variable x is known exactly but that the dependent variable y may have some uncertainty. That is, it finds a solution to the problem $(y - \Delta y) = Qx$ by using known values of x but only partially known values of y to find the best Q to minimize the difference between the line fit and the data in y. We write $y - \Delta y$ in the equation because the true value of the dependent variable is an unknown distance Δy away from the known measurement y.

The issue with using standard ordinary-least-squares regression on the total-capacity-estimation problem is that both the summed current value y and the difference between state of charge values x have sensor noise or estimation noise associated with them. Not only does our coulomb count y have sensor noise, but our estimates of state of charge are also generally imperfect, causing uncertainty on the x

12:07:34.

variable as well. That is, the total-capacity-estimation problem is implicitly of the form $(y - \Delta y) = Q(x - \Delta x)$ since both the integrated current and state of charge estimates have errors. This is a different problem from the one solved by ordinary least squares, so using the ordinary-least-squares solution biases the results.

The common approach to counteract this problem is to try to ensure that the state of charge estimates used in the regression are as accurate as possible and then use standard least-squares estimation anyway. For example, we might put constraints on when and how the total capacity is estimated. We could force the cell current to be zero for some period before the test begins and after the test ends so that the cell is in an equilibrium state and so the SOC estimates are as accurate as possible. This procedure eliminates to a large extent—but not completely—the error in the x variable, and makes the regression reasonably accurate. This method still does not handle the residual uncertainty in x correctly: while it minimizes the error, it never totally eliminates it.

A better approach is to recognize that the solution to problems that look like $(y - \Delta y) = Q(x - \Delta x)$ is to use *total-least-squares* regression instead of ordinary-least-squares regression. Total least squares takes into account the uncertainties in both the x and y variables when finding an estimate of Q. However, there are challenges when attempting to make total-least-squares solutions computationally efficient so that they can be implemented on a cost-effective BMS.

In the next sections, we derive the ordinary-least-squares and total-least-squares solutions and some variants thereof to show their similarities and differences. We will develop a computationally efficient approximate total-least-squares solution that works very well to estimate battery-cell capacity from noisy data.

4.13 *Weighted ordinary least squares*

Both ordinary least squares (OLS) and total least squares (TLS), as applied to battery-cell total-capacity estimation, seek to find a constant $\widehat{Q}_n$ such that $\boldsymbol{y} \approx \widehat{Q}_n \boldsymbol{x}$ using n-vectors of measured data $\boldsymbol{x}$ and $\boldsymbol{y}$. The ith data pair, comprising x_i in $\boldsymbol{x}$ and y_i in $\boldsymbol{y}$, correspond to data collected from a cell over the ith interval of time, where x_i is the estimated change in state of charge over that interval and y_i is the accumulated ampere-hours passing through the cell during that period.

Specifically,

$$x_i = \hat{z}_{k_2^{(i)}} - \hat{z}_{k_1^{(i)}} \qquad \text{for time interval } i$$

12:07:34.

$$y_i = -\sum_{k=k_1^{(i)}}^{k_2^{(i)}-1} \eta_k i_k,$$

where $k_1^{(i)}$ is the discrete-time index of the start of the ith interval, and $k_2^{(i)}$ is the discrete-time index of the end of the ith interval. The data vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ used to produce a capacity estimate must be at least one sample long ($n \geq 1$), but larger values of n may be used to obtain better estimates as the uncertainties in the data can be averaged out over many measurements.

The OLS approach assumes that there is no error on the x_i and models the data as $\boldsymbol{y} = Q\boldsymbol{x} + \Delta\boldsymbol{y}$, where $\Delta\boldsymbol{y}$ is a vector of unknown measurement errors. This is illustrated in Fig. 4.12, where the error bars on the data points are meant to depict the uncertainties. We assume that $\Delta\boldsymbol{y}$ comprises zero-mean Gaussian random variables with known variances $\sigma_{y_i}^2$ that may be different for every data point.

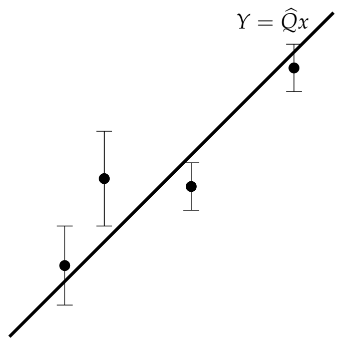


Figure 4.12: The OLS paradigm. (Reproduced from Fig. 1a of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

OLS attempts to find an estimate $\widehat{Q}_n$ of the true cell total capacity Q, based on n data pairs (x_i, y_i), that minimizes the sum of squared errors Δy_i. We generalize the OLS approach slightly here to allow for finding a $\widehat{Q}_n$ that minimizes the sum of *weighted* squared errors, where the weighting takes into account the uncertainty of the measurement. That is, we desire to find a $\widehat{Q}_n$ that minimizes the *weighted least squares (WLS)* cost function[17]

$$\chi_{\text{WLS}}^2 = \sum_{i=1}^{n} \frac{(y_i - Y_i)^2}{\sigma_{y_i}^2} = \sum_{i=1}^{n} \frac{(y_i - \widehat{Q}_n x_i)^2}{\sigma_{y_i}^2}.$$

[17] The cost function is named with the symbol χ^2 since it turns out to be a chi-squared random variable: it is the sum of the squares of uncorrelated zero-mean unit-variance Gaussian random variables.

In this equation, Y_i is the final optimized mapping of the data (x_i, y_i) to the line. That is, it is a point on the line $Y_i = \widehat{Q}_n x_i$ corresponding to the measured data pair (x_i, y_i), where y_i is assumed to have noise but x_i has no uncertainty.

There are a number of approaches that may be taken to solve this problem, but one that will serve our purposes well is to differentiate the cost function with respect to $\widehat{Q}_n$ and then to solve for $\widehat{Q}_n$ by setting the partial derivative to zero. The biggest need for care is to recognize that all the $\sigma_{y_i}^2$ inside the summation may be distinct, so we cannot simply multiply both sides of the equation by some constant σ_y^2 to eliminate it from the final result.

First, differentiating gives us

$$\frac{\partial \chi_{\text{WLS}}^2}{\partial \widehat{Q}_n} = -2\sum_{i=1}^{n} \frac{x_i(y_i - \widehat{Q}_n x_i)}{\sigma_{y_i}^2} = 0.$$

Then, dividing both sides of the equation by -2, splitting the summation into two separate summations, and solving for $\widehat{Q}_n$ gives

$$\widehat{Q}_n \sum_{i=1}^{n} \frac{x_i^2}{\sigma_{y_i}^2} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_{y_i}^2}$$

$$\widehat{Q}_n = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_{y_i}^2} \Bigg/ \sum_{i=1}^{n} \frac{x_i^2}{\sigma_{y_i}^2}.$$

If we define two new variables

$$c_{1,n} = \sum_{i=1}^{n} \frac{x_i^2}{\sigma_{y_i}^2} \qquad \text{and} \qquad c_{2,n} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_{y_i}^2},$$

then we can write $\widehat{Q}_n = c_{2,n}/c_{1,n}$.

The two quantities $c_{1,n}$ and $c_{2,n}$ may be computed recursively to minimize storage requirements and to even out computational requirements when updating $\widehat{Q}_n$ for large n via

$$\begin{aligned} c_{1,n} &= c_{1,n-1} + x_n^2/\sigma_{y_n}^2 \\ c_{2,n} &= c_{2,n-1} + x_n y_n/\sigma_{y_n}^2. \end{aligned}$$

The recursive approach requires an initial estimate of $c_{1,0}$ and $c_{2,0}$. One possible initialization is simply to set $c_{1,0} = c_{2,0} = 0$. Alternately, we can recognize that a cell with nominal capacity Q_{nom} has that capacity over a SOC range of 1.0. Therefore, we can initialize with a synthetic zeroth "measurement" where $x_0 = 1$ and $y_0 = Q_{\text{nom}}$. The value for $\sigma_{y_0}^2$ can be set to the manufacturing variance of the nominal capacity. That is, $c_{1,0} = 1/\sigma_{y_0}^2$ and $c_{2,0} = Q_{\text{nom}}/\sigma_{y_0}^2$.

This method may be adapted easily to allow fading memory of past measurements. We modify the WLS cost function to place more emphasis on recent measurements by defining the *fading-memory weighted least-squares (FMWLS)* cost function as

$$\chi_{\text{FMWLS}}^2 = \sum_{i=1}^{n} \gamma^{n-i} \frac{(y_i - \widehat{Q}_n x_i)^2}{\sigma_{y_i}^2},$$

where the forgetting factor γ is in the range $0 \ll \gamma \leq 1$. If we expand a few terms, we see that this looks like

$$\begin{aligned} \chi_{\text{FMWLS}}^2 = {} & \frac{(y_n - \widehat{Q}_n x_n)^2}{\sigma_{y_n}^2} + \gamma \frac{(y_{n-1} - \widehat{Q}_n x_{n-1})^2}{\sigma_{y_{n-1}}^2} \\ & + \gamma^2 \frac{(y_{n-2} - \widehat{Q}_n x_{n-2})^2}{\sigma_{y_{n-2}}^2} + \cdots \end{aligned}$$

and so more recent data points have higher weighting in the cost function than do data points collected far in the past. With this fading-memory cost function, the solution becomes

$$\widehat{Q}_n = \sum_{i=1}^{n} \gamma^{n-i} \frac{x_i y_i}{\sigma_{y_i}^2} \Bigg/ \sum_{i=1}^{n} \gamma^{n-i} \frac{x_i^2}{\sigma_{y_i}^2},$$

12:07:34.

which may also be computed easily in a recursive manner. To do so, we keep track of the two running sums:

$$\tilde{c}_{1,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i^2 / \sigma_{y_i}^2$$
$$\tilde{c}_{2,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i y_i / \sigma_{y_i}^2.$$

Then, the optimal total-capacity estimate is

$$\widehat{Q}_n = \tilde{c}_{2,n} / \tilde{c}_{1,n} \tag{4.8}$$

and the value of the cost function for this estimate can be written as

$$\chi^2_{\text{FMWLS}} = \tilde{c}_{1,n}\widehat{Q}_n^2 - 2\tilde{c}_{2,n}\widehat{Q}_n + \tilde{c}_{3,n}. \tag{4.9}$$

When an additional data point becomes available, we update these quantities via

$$\tilde{c}_{1,n} = \gamma \tilde{c}_{1,n-1} + x_n^2 / \sigma_{y_n}^2$$
$$\tilde{c}_{2,n} = \gamma \tilde{c}_{2,n-1} + x_n y_n / \sigma_{y_n}^2.$$

In summary, the WLS and FMWLS solutions have a number of nice properties:

1. They give a closed-form solution for $\widehat{Q}_n$. Only simple operations—multiplication, addition, and division—are required.
2. The solutions can be computed very easily in a recursive manner.
3. Fading memory can be added easily, allowing adaptation of $\widehat{Q}_n$ to adjust for changes in true cell total capacity.

4.14 *Weighted total least squares*

The TLS approach assumes that there are errors on both the x_i and y_i measurements and models the data as $(\boldsymbol{y} - \Delta\boldsymbol{y}) = Q(\boldsymbol{x} - \Delta\boldsymbol{x})$. This is illustrated in Fig. 4.13, where the error bars on the data points are meant to show the uncertainties in each dimension. We assume that $\Delta\boldsymbol{x}$ is a zero-mean Gaussian random vector with known element variances $\sigma_{x_i}^2$ and that $\Delta\boldsymbol{y}$ is a zero-mean Gaussian random vector with known element variances $\sigma_{y_i}^2$, where $\sigma_{x_i}^2$ is not necessarily equal to or related to $\sigma_{y_i}^2$, and where all of the $\sigma_{x_i}^2$ and $\sigma_{y_i}^2$ may be distinct.

TLS attempts to find an estimate $\widehat{Q}_n$ of the true cell total capacity Q that minimizes the sum of squared errors Δx_i plus the sum of squared errors Δy_i. We generalize that approach here slightly to allow for finding a $\widehat{Q}_n$ that minimizes the sum of *weighted* squared errors, where the weighting takes into account the uncertainty of the measurement.

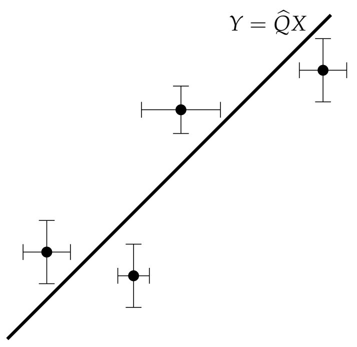


Figure 4.13: The TLS paradigm. (Reproduced from Fig. 1b of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

12:07:34.

That is, we desire to find a $\widehat{Q}_n$ that minimizes the *weighted total-least-squares (WTLS)* cost function

$$\chi^2_{\text{WTLS}} = \sum_{i=1}^{n} \frac{(x_i - X_i)^2}{\sigma^2_{x_i}} + \frac{(y_i - Y_i)^2}{\sigma^2_{y_i}}.$$

In this equation, (X_i, Y_i) is the final optimized mapping of the data (x_i, y_i) to the line. That is, X_i and Y_i are the points on the line $Y_i = \widehat{Q}_n X_i$ corresponding to the noisy measured data pair (x_i, y_i) for the optimized value of $\widehat{Q}_n$. In general, $x_i \neq X_i$ and $y_i \neq Y_i$. As part of the WTLS solution, we will need to find this optimal mapping from (x_i, y_i) to (X_i, Y_i).

Because both x_i and y_i have noise, we must handle this optimization problem differently from the way we handled the WLS problem. We augment the cost function with Lagrange multipliers λ_i to enforce the constraint that $Y_i = \widehat{Q}_n X_i$. This yields augmented cost function

$$\chi^2_{\text{WTLS},a} = \sum_{i=1}^{n} \frac{(x_i - X_i)^2}{\sigma^2_{x_i}} + \frac{(y_i - Y_i)^2}{\sigma^2_{y_i}} - \lambda_i (Y_i - \widehat{Q}_n X_i).$$

To solve, we set the partial derivatives of $\chi^2_{\text{WTLS},a}$ with respect to X_i, Y_i, and λ_i to zero. First, taking the partial derivative with respect to the Lagrange multipliers recovers the constraint:

$$\begin{aligned} \frac{\partial \chi^2_{\text{WTLS},a}}{\partial \lambda_i} &= -(Y_i - \widehat{Q}_n X_i) = 0 \\ Y_i &= \widehat{Q}_n X_i. \end{aligned} \tag{4.10}$$

Next, we take the partial derivative with respect to Y_i. This allows us to solve for the Lagrange multipliers:

$$\begin{aligned} \frac{\partial \chi^2_{\text{WTLS},a}}{\partial Y_i} &= \frac{-2(y_i - Y_i)}{\sigma^2_{y_i}} - \lambda_i = 0 \\ \lambda_i &= \frac{-2(y_i - Y_i)}{\sigma^2_{y_i}}. \end{aligned} \tag{4.11}$$

Finally, we take the partial derivative with respect to X_i:

$$\frac{\partial \chi^2_{\text{WTLS},a}}{\partial X_i} = \frac{-2(x_i - X_i)}{\sigma^2_{x_i}} + \lambda_i \widehat{Q}_n = 0. \tag{4.12}$$

We substitute the solution for the Lagrange multiplier from Eq. (4.11) into Eq. (4.12) and multiply both sides of the equation by $\sigma^2_{y_i}\sigma^2_{x_i}$ to simplify the form

$$\begin{aligned} 0 &= -\frac{2(x_i - X_i)}{\sigma^2_{x_i}} - \frac{2(y_i - Y_i)}{\sigma^2_{y_i}} \widehat{Q}_n \\ &= \sigma^2_{y_i}(x_i - X_i) + \sigma^2_{x_i}(y_i - Y_i)\widehat{Q}_n. \end{aligned}$$

Finally, we substitute the constraint Eq. (4.10) and solve for X_i:

$$\begin{aligned}0 &= \sigma_{y_i}^2 x_i - \sigma_{y_i}^2 X_i + \sigma_{x_i}^2 y_i \widehat{Q}_n - \sigma_{x_i}^2 X_i \widehat{Q}_n^2 \\ X_i &= \frac{x_i \sigma_{y_i}^2 + \widehat{Q}_n y_i \sigma_{x_i}^2}{\sigma_{y_i}^2 + \widehat{Q}_n^2 \sigma_{x_i}^2}.\end{aligned}$$

With these results, we can rewrite the original WTLS cost function in terms of measured and known quantities as

$$\begin{aligned}\chi_{\text{WTLS}}^2 &= \sum_{i=1}^n \frac{(x_i - X_i)^2}{\sigma_{x_i}^2} + \frac{(y_i - Y_i)^2}{\sigma_{y_i}^2} \\ &= \sum_{i=1}^n \frac{\left(x_i - \frac{x_i\sigma_{y_i}^2 + \widehat{Q}_n y_i \sigma_{x_i}^2}{\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2}\right)^2}{\sigma_{x_i}^2} + \frac{\left(y_i - \widehat{Q}_n \frac{x_i\sigma_{y_i}^2 + \widehat{Q}_n y_i \sigma_{x_i}^2}{\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2}\right)^2}{\sigma_{y_i}^2} \\ &= \sum_{i=1}^n \frac{\left(x_i\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right) - \left(x_i\sigma_{y_i}^2 + \widehat{Q}_n y_i \sigma_{x_i}^2\right)\right)^2}{\sigma_{x_i}^2\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right)^2} + \\ &\qquad \frac{\left(y_i\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right) - \widehat{Q}_n\left(x_i\sigma_{y_i}^2 + \widehat{Q}_n y_i \sigma_{x_i}^2\right)\right)^2}{\sigma_{y_i}^2\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right)^2} \\ &= \sum_{i=1}^n \frac{\widehat{Q}_n^2\sigma_{x_i}^4\left(y_i - \widehat{Q}_n x_i\right)^2}{\sigma_{x_i}^2\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right)^2} + \frac{\sigma_{y_i}^4\left(y_i - \widehat{Q}_n x_i\right)^2}{\sigma_{y_i}^2\left(\sigma_{y_i}^2 + \widehat{Q}_n^2\sigma_{x_i}^2\right)^2} \\ &= \sum_{i=1}^n \frac{(y_i - \widehat{Q}_n x_i)^2}{\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2}. \end{aligned} \tag{4.13}$$

To find the value of $\widehat{Q}_n$ that minimizes this cost function, we set the partial derivative $\partial\chi_{\text{WTLS}}^2/\partial\widehat{Q}_n = 0$. After a few steps, we can find that this is equivalent to solving for $\widehat{Q}_n$ in

$$\frac{\partial\chi_{\text{WTLS}}^2}{\partial\widehat{Q}_n} = \sum_{i=1}^n \frac{2(\widehat{Q}_n x_i - y_i)(\widehat{Q}_n y_i \sigma_{x_i}^2 + x_i\sigma_{y_i}^2)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^2} = 0. \tag{4.14}$$

Unfortunately, this solution has none of the nice properties of the WLS solution. Namely,

1. There is no closed-form solution in the general case. Instead, a numerical optimization method must be used in order to find $\widehat{Q}_n$. One possibility is to perform a Newton–Raphson search for $\widehat{Q}_n$, where several iterations of the equation

$$\widehat{Q}_{n,k} = \widehat{Q}_{n,k-1} - \frac{\partial\chi_{\text{WTLS}}^2/\partial\widehat{Q}_{n,k-1}}{\partial^2\chi_{\text{WTLS}}^2/\partial\widehat{Q}_{n,k-1}^2} \tag{4.15}$$

are performed every time the data vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are updated with a new data pair. That is, after n data pairs are gathered, the

12:07:34.

total-least-squares estimate $\widehat{Q}_{n,0}$ is initialized to some value. We could use an estimate $\widehat{Q}_n$ from a weighted-least-squares solution, or we could use the prior estimate $\widehat{Q}_{n-1}$ from a weighted-total-least-squares solution for this initialization. Then, Eq. (4.15) is computed at time index n for $k = 1$ to some number of total iterations K. Then, the final weighted-total-least-squares solution at this time step is set to $\widehat{Q}_n = \widehat{Q}_{n,K}$.

The numerator of Eq. (4.15) is the *Jacobian* of the original cost function and is given by Eq. (4.14), if $\widehat{Q}_n$ is replaced by $\widehat{Q}_{n,k-1}$. The denominator of this update equation is the *Hessian* of the original metric function, which can be found using known quantities to be

$$\frac{\partial^2 \chi^2_{\text{WTLS}}}{\partial \widehat{Q}_n^2} = 2\sum_{i=1}^{n}\left(\frac{\sigma_{y_i}^4 x_i^2 + \sigma_{x_i}^4(3\widehat{Q}_n^2 y_i^2 - 2\widehat{Q}_n^3 x_i y_i)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^3}\right. \\ \left. - \frac{\sigma_{x_i}^2\sigma_{y_i}^2(3\widehat{Q}_n^2 x_i^2 - 6\widehat{Q}_n x_i y_i + y_i^2)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^3}\right). \tag{4.16}$$

This result is used in the Newton–Raphson iteration of Eq. (4.15) with $\widehat{Q}_n$ replaced by $\widehat{Q}_{n,k-1}$.

This Newton–Raphson search has the property that the number of significant figures in the solution doubles with each iteration of the update. In practice, we find that around four iterations produce double-precision results. Also note that the metric function χ^2_{WTLS} is convex, which guarantees that this iterative method will converge to the global solution.

2. There is no recursive update in the general case. This has both storage and computational implications. To use WTLS, the entire vector $\boldsymbol{x}$ and $\boldsymbol{y}$ must be retained, which implies increasing storage as the number of measurements increase. Furthermore, the number of computations grows as n grows. That is, WTLS is not well suited for an embedded-system application that must run in real time within limited memory.

3. There is no fading memory recursive update (because there is no recursive update). A nonrecursive *fading memory weighted total least squares (FMWTLS)* cost function may be defined, however, as

$$\chi^2_{\text{FMWTLS}} = \sum_{i=1}^{n}\gamma^{n-i}\frac{(y_i - \widehat{Q}_n x_i)^2}{\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2}.$$

The Jacobian of this cost function is

$$\frac{\partial \chi^2_{\text{FMWTLS}}}{\partial \widehat{Q}_n} = 2\sum_{i=1}^{n}\gamma^{n-i}\frac{(\widehat{Q}_n x_i - y_i)(\widehat{Q}_n y_i\sigma_{x_i}^2 + x_i\sigma_{y_i}^2)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^2}. \tag{4.17}$$

12:07:34.

The Hessian is

$$\frac{\partial^2 \chi^2_{\text{FMWTLS}}}{\partial \widehat{Q}_n^2} = 2\sum_{i=1}^{n} \gamma^{n-i}\left(\frac{\sigma_{y_i}^4 x_i^2 + \sigma_{x_i}^4(3\widehat{Q}_n^2 y_i^2 - 2\widehat{Q}_n^3 x_i y_i)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^3} - \frac{\sigma_{x_i}^2\sigma_{y_i}^2(3\widehat{Q}_n^2 x_i^2 - 6\widehat{Q}_n x_i y_i + y_i^2)}{(\widehat{Q}_n^2\sigma_{x_i}^2 + \sigma_{y_i}^2)^3}\right). \quad (4.18)$$

Using the Jacobian and Hessian of this cost function, we can use a Newton–Raphson search to find the solution to the fading-memory cost function to find an estimate of Q.

In Sect. 4.17, we will address a special case of WTLS that gives a closed-form solution with recursive update and fading memory. We will then give an approximate solution to the general WTLS problem that also has the nice characteristics of the WLS solution. Before we do so, we first consider two important properties of both the WLS and WTLS solutions.

4.15 *Goodness of model fit*

When the measurement errors Δx and Δy are uncorrelated and Gaussian, the metric functions χ^2_{WLS} and χ^2_{WTLS} are chi-squared random variables.

- χ^2_{WLS} is a chi-squared random variable with $n-1$ degrees of freedom because n data points y_i were used in its creation and one degree of freedom is lost when fitting $\widehat{Q}_n$.
- χ^2_{WTLS} is a chi-squared random variable with $2n-1$ degrees of freedom because n data points x_i and n additional data points y_i are used in its creation and one degree of freedom is lost when fitting $\widehat{Q}_n$.

Knowledge of the distribution and the number of degrees of freedom can be used to determine, from the optimized values of the metric functions, whether the model fit is reliable; that is, whether the linear fit is a good fit to the data and whether the optimized value of $\widehat{Q}_n$ is a good estimate of the cell's total capacity.

The incomplete gamma function $P(\chi^2 \mid \nu)$ is defined as the probability that the observed chi-square for a correct model should be less than a computed value χ^2 for degree of freedom ν. Its complement, $Q(\chi^2 \mid \nu) = 1 - P(\chi^2 \mid \nu)$, is the probability that the observed chi-square will exceed the value χ^2 by chance *even* for a correct model.[18] Therefore, to test for goodness of fit of a model, we must evaluate

$$Q(\chi^2 \mid \nu) = \frac{1}{\Gamma(\nu/2)}\int_{\chi^2/2}^{\infty} e^{-t}t^{(\nu/2-1)}\,\mathrm{d}t. \quad (4.19)$$

Methods for computing this function are built into many engineering analysis programs, and C-language code is also easy to find.

[18] The nomenclature $Q(\chi^2 \mid \nu)$ is standard for the (complementary) incomplete gamma function, and is not to be confused with the symbol used to denote true cell total capacity Q, or with the symbol used to denote the estimate of cell total capacity $\widehat{Q}_n$.

12:07:34.

If the value obtained for $Q(\chi^2 \mid \nu)$ is small for some cost χ^2 and degree of freedom ν, then either the model is wrong and can be statistically rejected, or the variances $\sigma^2_{x_i}$ or $\sigma^2_{y_i}$ are poorly known, or the variances are not actually Gaussian. The third possibility is common, but is also generally benign if we are willing to accept low values of $Q(\chi^2 \mid \nu)$ as representing a valid model. It is not unusual to accept models if $Q(\chi^2 \mid \nu) \gtrsim 0.001$ and to reject them otherwise.

We will see that when the hypothesized model is not a good fit to the data, the value of $Q(\chi^2 \mid \nu)$ becomes extremely small. However, when the hypothesized model is equal to the true model generating the data, even when $\widehat{Q}_n$ is not precisely equal to Q, the value of $Q(\chi^2 \mid \nu)$ tends to be very close to unity. We will use this information later to show that the WLS model is not a good approach to total-capacity estimation, whereas WTLS is much better. It could also be used online in a battery-management system as a check for the validity of the present total-capacity estimate.

4.16 Confidence intervals

When computing an estimate of cell total capacity $\widehat{Q}_n$, it is also important to be able to specify the certainty of that estimate. Specifically, we would like to estimate the variance $\sigma^2_{\widehat{Q}_n}$ of the total capacity estimate, with which we can compute confidence intervals such as three-sigma bounds $(\widehat{Q}_n - 3\sigma_{\widehat{Q}_n}, \widehat{Q}_n + 3\sigma_{\widehat{Q}_n})$ within which we have high certainty that the true value of cell total capacity Q lies.

To derive confidence limits, we must recast the least-squares type optimization problem as a maximum-likelihood optimization problem. With the assumption that all errors are Gaussian, this is straightforward. If we form a vector $\boldsymbol{y}$ comprising elements y_i and a vector $\boldsymbol{x}$ comprising corresponding elements x_i and a diagonal matrix $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}}}$ having corresponding diagonal elements $\sigma^2_{y_i}$, then minimizing χ^2_{WLS} is equivalent to maximizing the value of the multivariable Gaussian pdf:

$$
\begin{aligned}
ML_{\text{WLS}} &= \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}}}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y} - \widehat{Q}_n\boldsymbol{x})^T \boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}}}^{-1}(\boldsymbol{y} - \widehat{Q}_n\boldsymbol{x})\right) \\
&= \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}}}|^{1/2}} \exp\left(-\frac{1}{2}\chi^2_{\text{WLS}}\right),
\end{aligned} \tag{4.20}
$$

which is a maximum-likelihood problem. That is, minimizing χ^2_{WLS} yields the same $\widehat{Q}_n$ that maximizes ML_{WLS}.[19]

[19] The constant to the left of the exponential in Eq. (4.20) causes the function to integrate to 1, yielding a valid probability density function.

Similarly, if we form a vector $\boldsymbol{d}$ by concatenating $\boldsymbol{y}$ and $\boldsymbol{x}$, and a vector $\hat{\boldsymbol{d}}$ by concatenating the corresponding elements Y_i and X_i, and a diagonal matrix $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}}}$ having diagonal elements $\sigma^2_{y_i}$ followed by $\sigma^2_{x_i}$,

12:07:34.

then minimizing χ^2_{WTLS} is equivalent to maximizing the value of

$$\begin{aligned} ML_{\text{WTLS}} &= \frac{1}{(2\pi)^n |\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}}}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{d}-\hat{\boldsymbol{d}})^T \boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}}}^{-1}(\boldsymbol{d}-\hat{\boldsymbol{d}})\right) \\ &= \frac{1}{(2\pi)^n |\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}}}|^{1/2}} \exp\left(-\frac{1}{2}\chi^2_{\text{WTLS}}\right). \end{aligned}$$

The maximum-likelihood formulation makes it possible to determine confidence intervals on $\widehat{Q}_n$. According to the Cramer–Rao theorem, a tight lower bound on the variance of $\widehat{Q}_n$ is given by the negative inverse of the second derivative of the argument of the exponential function, evaluated at the $\widehat{Q}_n$ that minimizes the least-squares cost function or maximizes the maximum-likelihood cost function. Therefore, we have

$$\begin{aligned} \sigma^2_{\widehat{Q}_n} &\geq 2\left(\frac{\partial^2 \chi^2_{\text{WLS}}}{\partial \widehat{Q}_n^2}\right)^{-1} && \text{for WLS} \\ \sigma^2_{\widehat{Q}_n} &\geq 2\left(\frac{\partial^2 \chi^2_{\text{WTLS}}}{\partial \widehat{Q}_n^2}\right)^{-1} && \text{for WTLS.} \end{aligned}$$

These lower bounds on the variance are often quite tight, so are reasonable to be used in computing confidence intervals on our total-capacity estimates.

The second partial derivatives (i.e., the Hessians) of the WTLS and FMWTLS metric functions have already been described in the context of a Newton–Raphson iteration by Eqs. (4.14) and (4.16) for WTLS and Eqs. (4.17) and (4.18) for FMWTLS. For WLS and FMWLS, the situation is easier. We have

$$\begin{aligned} \frac{\partial^2 \chi^2_{\text{WLS}}}{\partial \widehat{Q}_n^2} &= 2\sum_{i=1}^{n} \frac{x_i^2}{\sigma^2_{y_i}} \\ \frac{\partial^2 \chi^2_{\text{FMWLS}}}{\partial \widehat{Q}_n^2} &= 2\sum_{i=1}^{n} \gamma^{n-i} \frac{x_i^2}{\sigma^2_{y_i}}, \end{aligned}$$

which may be computed using the previously defined recursive parameters as:

$$\begin{aligned} \frac{\partial^2 \chi^2_{\text{WLS}}}{\partial \widehat{Q}_n^2} &= 2c_{1,n} \\ \frac{\partial^2 \chi^2_{\text{FMWLS}}}{\partial \widehat{Q}_n^2} &= 2\tilde{c}_{1,n}. \end{aligned} \tag{4.21}$$

We will use these results to produce confidence intervals on our estimate in the examples in Sect. 4.20 and 4.21. Note that this method produces lower bounds on the width of the confidence interval: it is possible for the real bounds to be wider. Keeping this in mind, the

12:07:34.

plots in these future sections are somewhat optimistic. However, we will note that the confidence intervals are still uncomfortably wide in some cases. This is unavoidable. We are computing optimal estimates of total capacity. The wide error bounds are due to the total-capacity estimation problem being very hard and this is the best that can be done.

4.17 *Simplified total least squares*

4.17.1 *TLS with proportional confidence on x_i and y_i*

The general WTLS and FMWTLS solutions provide excellent results but are impractical to implement in an embedded system due to ever-growing memory and computational requirements. Therefore, we search for cases that lead to simpler implementations. Here, we look at an exact solution for the case when the uncertainties on the x_i and y_i data points are proportional to each other for all i, which can be implemented easily in an embedded system. With insights from this solution we will next look at an approximate WTLS solution that also has nice implementation properties.

If $\sigma_{x_i} = k\sigma_{y_i}$ for all i, then the WTLS cost function reduces to a generalization of the standard TLS cost function. We substitute $\sigma_{x_i} = k\sigma_{y_i}$ into χ^2_{WTLS} and associated results to get the proportional total-least-squares (PTLS) cost function

$$\chi^2_{\text{PTLS}} = \sum_{i=1}^{n} \frac{(x_i - X_i)^2}{k^2\sigma^2_{y_i}} + \frac{(y_i - Y_i)^2}{\sigma^2_{y_i}} = \sum_{i=1}^{n} \frac{(y_i - \widehat{Q}_n x_i)^2}{(\widehat{Q}_n^2 k^2 + 1)\sigma^2_{y_i}}.$$

Furthermore, the Jacobian of the WTLS cost function reduces (again, via the substitution $\sigma_{x_i} = k\sigma_{y_i}$) to

$$\frac{\partial \chi^2_{\text{PTLS}}}{\partial \widehat{Q}_n} = 2\sum_{i=1}^{n} \frac{(\widehat{Q}_n x_i - y_i)(\widehat{Q}_n k^2 y_i + x_i)}{(\widehat{Q}_n^2 k^2 + 1)^2 \sigma^2_{y_i}}.$$

This equation may be solved for an exact solution to $\widehat{Q}_n$ without requiring iteration to do so.

We set the Jacobian to zero and collect terms

$$\begin{aligned}
\frac{\partial \chi^2_{\text{PTLS}}}{\partial \widehat{Q}_n} &= 2\sum_{i=1}^{n} \frac{(\widehat{Q}_n x_i - y_i)(\widehat{Q}_n k^2 y_i + x_i)}{(\widehat{Q}_n^2 k^2 + 1)^2 \sigma^2_{y_i}} = 0 \\
&= \widehat{Q}_n^2 \underbrace{\sum_{i=1}^{n} k^2 \frac{x_i y_i}{\sigma^2_{y_i}}}_{a = k^2 c_{2,n}} + \widehat{Q}_n \underbrace{\sum_{i=1}^{n} \frac{x_i^2 - k^2 y_i^2}{\sigma^2_{y_i}}}_{b = c_{1,n} - k^2 c_{3,n}} + \underbrace{\sum_{i=1}^{n} \frac{-x_i y_i}{\sigma^2_{y_i}}}_{c = -c_{2,n}} \\
&= a\widehat{Q}_n^2 + b\widehat{Q}_n + c = 0,
\end{aligned}$$

12:07:34.

where we have defined $c_{3,n} = \sum_{i=1}^{n} y_i^2/\sigma_{y_i}^2$. Note that the solution for $\widehat{Q}_n$ can be found via the quadratic formula

$$\widehat{Q}_n = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Substituting the recursive quantities, we can write

$$\widehat{Q}_n = \frac{-(c_{1,n} - k^2 c_{3,n}) \pm \sqrt{(c_{1,n} - k^2 c_{3,n})^2 + 4k^2 c_{2,n}^2}}{2k^2 c_{2,n}}. \tag{4.22}$$

Which of these two solutions should we to choose for our final total-capacity estimate? We can show using a Routh test that this quadratic equation always has exactly one positive root and one negative root. Because total capacity must be positive, we will choose the positive solution to Eq. (4.22).

To see this, we compute the Routh array for this quadratic equation, which is:

$$\begin{array}{c|cc} \widehat{Q}_n^2 & k^2 c_{2,n} & -c_{2,n} \\ \widehat{Q}_n^1 & c_{1,n} - k^2 c_{3,n} & 0 \\ \widehat{Q}_n^0 & -c_{2,n} & 0 \end{array}$$

The Routh test examines the left column of the Routh array and counts the number of sign changes as we traverse from the first row to the final row. The number of roots in the open right-half complex plane is equal to the number of sign changes that we count. For this array, we note that the top entry must be positive since $c_{2,n}$ is always positive and similarly the bottom entry must be negative. Therefore, regardless of the sign on the middle entry, there is exactly one sign change from positive to negative as we traverse the first column.

Consequently, there is exactly one root of the polynomial in the right-half complex plane. The other root, therefore, must be in the left-half plane or on the imaginary axis. By the fundamental theorem of algebra, because the coefficients $c_{1,n}$, $c_{2,n}$, and $c_{3,n}$ are real, the polynomial roots must either both be real or be complex conjugates. The fact that they are in different halves of the complex plane shows that they cannot be complex conjugates, and therefore must both be real. So, we choose the larger root from the solution of the quadratic equation, which corresponds to the positive root.

Recursive calculation is done via

$$\widehat{Q}_n = \frac{-c_{1,n} + k^2 c_{3,n} + \sqrt{(c_{1,n} - k^2 c_{3,n})^2 + 4k^2 c_{2,n}^2}}{2k^2 c_{2,n}},$$

where initialization is done by setting $x_0 = 1$, $y_0 = Q_{\text{nom}}$, and $\sigma_{y_i}^2$ to a value representing the uncertainty of the total capacity at manufacture. Therefore,

$$\begin{aligned} c_{1,0} &= 1/\sigma_{y_i}^2 \\ c_{2,0} &= Q_{\text{nom}}/\sigma_{y_i}^2 \\ c_{3,0} &= Q_{\text{nom}}^2/\sigma_{y_i}^2 \end{aligned}$$

and

$$\begin{aligned} c_{1,n} &= c_{1,n-1} + x_n^2/\sigma_{y_i}^2 \\ c_{2,n} &= c_{2,n-1} + x_n y_n/\sigma_{y_i}^2 \\ c_{3,n} &= c_{3,n-1} + y_n^2/\sigma_{y_i}^2. \end{aligned}$$

The Hessian, which is required to compute the uncertainty of the estimate, may also be found in terms of the recursive parameters:

$$\begin{aligned} \frac{\partial^2 \chi_{\text{PTLS}}^2}{\partial \widehat{Q}_n^2} = \frac{(-4k^4c_2)\widehat{Q}_n^3 + 6k^4c_3\widehat{Q}_n^2}{(\widehat{Q}_n^2k^2+1)^3} \\ + \frac{(-6c_1+12c_2)k^2\widehat{Q}_n + 2(c_1 - k^2c_3)}{(\widehat{Q}_n^2k^2+1)^3}. \end{aligned}$$

This can be used to predict error bounds on the estimate $\widehat{Q}_n$. One-sigma bounds are computed as

$$\sigma_{\widehat{Q}_n} = \sqrt{2/(\partial^2\chi_{\text{PTLS}}^2/\partial\widehat{Q}_n^2)}.$$

Fading memory may be incorporated easily. Following the same steps, we find that recursive calculation is done via

$$\widehat{Q}_n = \frac{-\tilde{c}_{1,n} + k^2\tilde{c}_{3,n} + \sqrt{(\tilde{c}_{1,n} - k^2\tilde{c}_{3,n})^2 + 4k^2\tilde{c}_{2,n}^2}}{2k^2\tilde{c}_{2,n}}, \tag{4.23}$$

where initialization is done by setting $x_0 = 1$, $y_0 = Q_{\text{nom}}$, and $\sigma_{y_i}^2$ to a value representing the uncertainty of the initial total capacity. Therefore,

$$\begin{aligned} \tilde{c}_{1,0} &= 1/\sigma_{y_i}^2 \\ \tilde{c}_{2,0} &= Q_{\text{nom}}/\sigma_{y_i}^2 \\ \tilde{c}_{3,0} &= Q_{\text{nom}}^2/\sigma_{y_i}^2 \end{aligned}$$

and

$$\begin{aligned} \tilde{c}_{1,n} &= \gamma\tilde{c}_{1,n-1} + x_n^2/\sigma_{y_i}^2 \\ \tilde{c}_{2,n} &= \gamma\tilde{c}_{2,n-1} + x_n y_n/\sigma_{y_i}^2 \\ \tilde{c}_{3,n} &= \gamma\tilde{c}_{3,n-1} + y_n^2/\sigma_{y_i}^2. \end{aligned}$$

After some straightforward manipulations, we can obtain the Hessian in terms of the recursive parameters $\tilde{c}_1$ through $\tilde{c}_3$:

$$\frac{\partial^2 \chi^2_{\text{FMPTLS}}}{\partial \widehat{Q}_n^2} = \frac{(-4k^4\tilde{c}_2)\widehat{Q}_n^3 + 6k^4\tilde{c}_3\widehat{Q}_n^2}{(\widehat{Q}_n^2 k^2 + 1)^3} + \frac{(-6\tilde{c}_1 + 12\tilde{c}_2)k^2\widehat{Q}_n + 2(\tilde{c}_1 - k^2\tilde{c}_3)}{(\widehat{Q}_n^2 k^2 + 1)^3}. \tag{4.24}$$

This can be used to predict error bounds on the estimate $\widehat{Q}_n$. One-sigma bounds are computed as

$$\sigma_{\widehat{Q}_n} = \sqrt{2/(\partial^2 \chi^2_{\text{FMPTLS}}/\partial \widehat{Q}_n^2)}.$$

Further, the cost function for the optimizing $\widehat{Q}_n$ can be computed in terms of the recursive parameters as

$$\chi^2_{\text{FMPTLS}} = \frac{\tilde{c}_{1,n}\widehat{Q}_n^2 - 2\tilde{c}_{2,n}\widehat{Q}_n + \tilde{c}_{3,n}}{\widehat{Q}_n^2 k^2 + 1}. \tag{4.25}$$

In summary, the proportional-total-least-squares solution shares the nice properties of the WLS solution:

1. It gives a closed-form solution for $\widehat{Q}_n$. No iteration or advanced algorithms are required—only simple mathematical operations.
2. The solution can be computed very easily in a recursive manner. We keep track of the three running sums $c_{1,n}$, $c_{2,n}$ and $c_{3,n}$. When an additional data point becomes available, we update the sums and compute an updated total-capacity estimate.
3. Fading memory is easily added.

Unfortunately, this solution does not allow $\sigma^2_{x_i}$ and $\sigma^2_{y_i}$ to be arbitrary—they must be proportionally related by the scaling factor $\sigma_{x_i} = k\sigma_{y_i}$ for every data point. The next section describes an approximation to PTLS that allows an arbitrary relationship.

4.18 *Approximate full solution*

4.18.1 *Deriving the approximate weighted total-least-squares cost function*

We desire an approximate solution to the WTLS problem that allows $\sigma^2_{x_i}$ and $\sigma^2_{y_i}$ to be nonproportional and that yields a recursive solution for feasible implementation in an embedded system. We will do so by considering Fig. 4.14, which illustrates the geometry of the WTLS and PTLS solutions and motivates the geometry of the approximate solution to be developed in this section.

12:07:34.

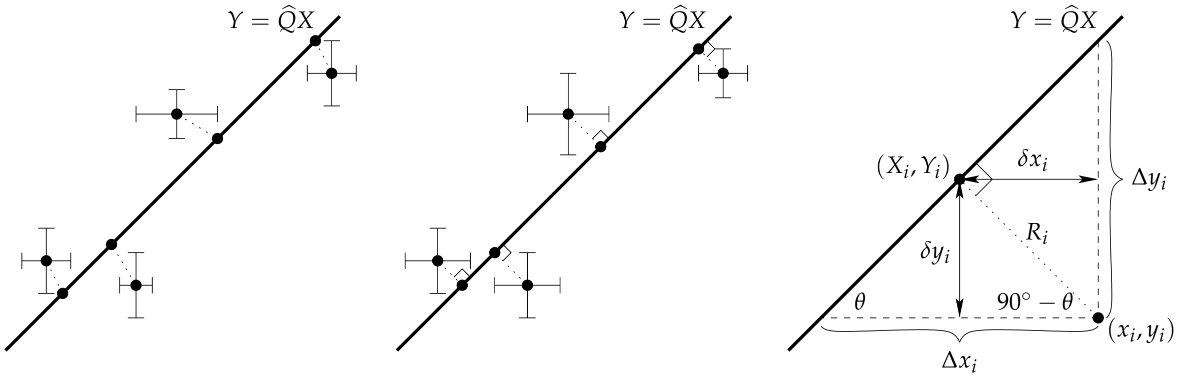


Figure 4.14: Geometry of WTLS, PTLS, and AWTLS.
(Reproduced from Fig. 2 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

The left frame of the figure shows the WTLS relationship between data point (x_i, y_i) and its optimized map (X_i, Y_i) on $Y_i = \widehat{Q}_n X_i$ when $\sigma_{x_i}^2$ and $\sigma_{y_i}^2$ are arbitrary. The data points (x_i, y_i) are drawn with error bars to illustrate the uncertainties in each dimension, which are proportional to σ_{x_i} and σ_{y_i}. A dotted line joins each data point (x_i, y_i) to its map (X_i, Y_i) on the line $Y_i = \widehat{Q}_n X_i$. We see that the distance between x_i and X_i is not necessarily equal to the distance between y_i and Y_i. If the quality of the x_i measurement is better (poorer) than the quality of the y_i measurement, the distance to its map X_i should be shorter (greater) than the distance from y_i to its map Y_i.

The middle frame shows the PTLS relationship between data point (x_i, y_i) and its optimized map (X_i, Y_i) on $Y_i = \widehat{Q}_n X_i$ when $\sigma_{x_i}^2$ and $\sigma_{y_i}^2$ are equal. In this case, the distance between x_i and X_i is equal to the distance between y_i and Y_i, and the line joining data point (x_i, y_i) and its map (X_i, Y_i) is perpendicular to the line $Y_i = \widehat{Q}_n X_i$. If σ_{x_i} and σ_{y_i} are not equal but proportional, either of the x- or y-axes may be scaled to yield transformed data points having equal variances and hence the same idea applies.

The right frame of the figure illustrates definitions that will be used to derive an approximate weighted total-least-squares (AWTLS) solution. As with the PTLS solution, we enforce that the line joining data point (x_i, y_i) and (X_i, Y_i) be perpendicular to the line $Y_i = \widehat{Q}_n X_i$. This will result in a solution that may be solved recursively. However, as with the WTLS solution, we weight the distance between x_i and X_i differently from the distance between y_i and Y_i in the optimization cost function.

We define Δx_i be the x-distance between data point i and the line, and Δy_i be the y-distance between data point i and the line. The slope of the line is $\widehat{Q}_n = \Delta y_i / \Delta x_i$ for all i. The angle of the line is $\theta = \tan^{-1} \widehat{Q}_n$. The shortest distance between the line and a given data point is

$$R_i = \Delta y_i \cos\theta = \Delta y_i / \sqrt{1+\tan^2\theta} = \Delta y_i / \sqrt{1+\widehat{Q}_n^2}.$$

We let $\delta x_i = R_i \sin\theta$ and $\delta y_i = R_i \cos\theta$. These are the x- and y-components of the perpendicular distance between data point i and the fitting line. We then weight our fitting cost function according to these variances. Therefore, we define the *approximate weighted total least squares (AWTLS)* cost function as

$$\chi^2_{\text{AWTLS}} = \sum_{i=1}^{n} \frac{\delta x_i^2}{\sigma_{x_i}^2} + \frac{\delta y_i^2}{\sigma_{y_i}^2}.$$

Note that $\sin^2\theta = 1 - \cos^2\theta = \widehat{Q}_n^2/(1+\widehat{Q}_n^2)$. Therefore,

$$\begin{aligned}\delta x_i^2 &= \left(\frac{\Delta y_i^2}{1+\widehat{Q}_n^2}\right)\left(\frac{\widehat{Q}_n^2}{1+\widehat{Q}_n^2}\right)\\ \delta y_i^2 &= \left(\frac{\Delta y_i^2}{1+\widehat{Q}_n^2}\right)\left(\frac{1}{1+\widehat{Q}_n^2}\right).\end{aligned}$$

Because $\Delta y_i = y_i - \widehat{Q}_n x_i$, we can then write

$$\chi^2_{\text{AWTLS}} = \sum_{i=1}^{n} \frac{(y_i - \widehat{Q}_n x_i)^2}{(1+\widehat{Q}_n^2)^2}\left(\frac{\widehat{Q}_n^2}{\sigma_{x_i}^2} + \frac{1}{\sigma_{y_i}^2}\right).$$

To verify that AWTLS is an approximation to WTLS in at least some cases, we note that the two cost functions are equal when $\sigma_{x_i} = \sigma_{y_i}$. However, they are not equal when $\sigma_{x_i} = k\sigma_{y_i}$, but this will be corrected in Sect. 4.18.4.

4.18.2 Minimizing the AWTLS cost function

The Jacobian of the AWTLS cost function (found with the help of Mathematica) is

$$\begin{aligned}\frac{\partial \chi^2_{\text{AWTLS}}}{\partial \widehat{Q}_n} = \frac{2}{(\widehat{Q}_n^2+1)^3} \sum_{i=1}^{n} \widehat{Q}_n^4 \left(\frac{x_i y_i}{\sigma_{x_i}^2}\right) + \widehat{Q}_n^3 \left(\frac{2x_i^2}{\sigma_{x_i}^2} - \frac{x_i^2}{\sigma_{y_i}^2} - \frac{y_i^2}{\sigma_{x_i}^2}\right)\\ + \widehat{Q}_n^2\left(\frac{3x_i y_i}{\sigma_{y_i}^2} - \frac{3x_i y_i}{\sigma_{x_i}^2}\right) + \widehat{Q}_n\left(\frac{x_i^2 - 2y_i^2}{\sigma_{y_i}^2} + \frac{y_i^2}{\sigma_{x_i}^2}\right) + \left(\frac{-x_i y_i}{\sigma_{y_i}^2}\right).\end{aligned}$$

We define additional recursive quantities

$$c_{4,n} = \sum_{i=1}^{n} \frac{x_i^2}{\sigma_{x_i}^2}, \qquad c_{5,n} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_{x_i}^2}, \qquad c_{6,n} = \sum_{i=1}^{n} \frac{y_i^2}{\sigma_{x_i}^2}.$$

This allows us to write the cost function in terms of recursively computed running summations

$$\begin{aligned}\frac{\partial \chi^2_{\text{AWTLS}}}{\partial \widehat{Q}_n} = \frac{2}{(\widehat{Q}_n^2+1)^3}\Big(c_5\widehat{Q}_n^4 + (2c_4 - c_1 - c_6)\widehat{Q}_n^3\\ + (3c_2 - 3c_5)\widehat{Q}_n^2 + (c_1 - 2c_3 + c_6)\widehat{Q}_n - c_2\Big),\end{aligned}$$

where initialization is done by setting $x_0 = 1$, $y_0 = Q_{\text{nom}}$, $\sigma^2_{y_0}$ to a representative value of the uncertainty of the total capacity, and $\sigma^2_{x_0}$ to a representative value of the uncertainty of a difference between two state of charge estimates.

Therefore, we initialize

$$\begin{aligned} c_{1,0} &= 1/\sigma^2_{y_0} & c_{4,0} &= 1/\sigma^2_{x_0} \\ c_{2,0} &= Q_{\text{nom}}/\sigma^2_{y_0} & c_{5,0} &= Q_{\text{nom}}/\sigma^2_{x_0} \\ c_{3,0} &= Q^2_{\text{nom}}/\sigma^2_{y_0} & c_{6,0} &= Q^2_{\text{nom}}/\sigma^2_{x_0}, \end{aligned}$$

and recursively compute

$$\begin{aligned} c_{1,n} &= c_{1,n-1} + x_n^2/\sigma^2_{y_n} & c_{4,n} &= c_{4,n-1} + x_n^2/\sigma^2_{x_n} \\ c_{2,n} &= c_{2,n-1} + x_n y_n/\sigma^2_{y_n} & c_{5,n} &= c_{5,n-1} + x_n y_n/\sigma^2_{x_n} \\ c_{3,n} &= c_{3,n-1} + y_n^2/\sigma^2_{y_n} & c_{6,n} &= c_{6,n-1} + y_n^2/\sigma^2_{x_n}. \end{aligned}$$

We minimize the cost function by setting its Jacobian to zero. Therefore, any of the roots of the quartic equation

$$c_5\widehat{Q}_n^4 + (2c_4 - c_1 - c_6)\widehat{Q}_n^3 + (3c_2 - 3c_5)\widehat{Q}_n^2 + (c_1 - 2c_3 + c_6)\widehat{Q}_n - c_2 = 0 \tag{4.26}$$

is a candidate solution for $\widehat{Q}_n$. We will discuss how to solve Eq. (4.26) for these roots and select the optimal value in Sect. 4.18.3.

When the assumptions made in deriving AWTLS are approximately true, the Hessian yields a good value for the error bounds on the total-capacity estimate. After some straightforward but messy mathematics, we can find the Hessian in terms of the recursive parameters to be

$$\begin{aligned} \frac{\partial^2 \chi^2_{\text{AWTLS}}}{\partial \widehat{Q}_n^2} = \frac{2}{(\widehat{Q}_n^2+1)^4}\Big(& -2c_5\widehat{Q}_n^5 + (3c_1 - 6c_4 + 3c_6)\widehat{Q}_n^4 \\ & + (-12c_2 + 16c_5)\widehat{Q}_n^3 + (-8c_1 + 10c_3 + 6c_4 - 8c_6)\widehat{Q}_n^2 \\ & + (12c_2 - 6c_5)\widehat{Q}_n + (c_1 - 2c_3 + c_6)\Big). \end{aligned}$$

Fading memory can be incorporated easily. The cost function is

$$\chi^2_{\text{FMAWTLS}} = \sum_{i=1}^{n} \gamma^{n-i} \frac{(y_i - \widehat{Q}_n x_i)^2}{(1+\widehat{Q}_n^2)^2}\left(\frac{\widehat{Q}_n^2}{\sigma^2_{x_i}} + \frac{1}{\sigma^2_{y_i}}\right).$$

The Jacobian is

$$\begin{aligned} \frac{\partial \chi^2_{\text{AWTLS}}}{\partial \widehat{Q}_n} = \frac{2}{(\widehat{Q}_n^2+1)^3}\sum_{i=1}^{n}\gamma^{n-i}\Bigg[& \widehat{Q}_n^4\left(\frac{x_i y_i}{\sigma^2_{x_i}}\right) + \widehat{Q}_n^3\left(\frac{2x_i^2}{\sigma^2_{x_i}} - \frac{x_i^2}{\sigma^2_{y_i}} - \frac{y_i^2}{\sigma^2_{x_i}}\right) \\ & + \widehat{Q}_n^2\left(\frac{3x_i y_i}{\sigma^2_{y_i}} - \frac{3x_i y_i}{\sigma^2_{x_i}}\right) + \widehat{Q}_n\left(\frac{x_i^2 - 2y_i^2}{\sigma^2_{y_i}} + \frac{y_i^2}{\sigma^2_{x_i}}\right) + \left(\frac{-x_i y_i}{\sigma^2_{y_i}}\right)\Bigg], \end{aligned}$$

which can be rewritten in terms of recursively computed running summations as

$$\frac{\partial \chi^2_{\text{FMAWTLS}}}{\partial \widehat{Q}_n} = \frac{2}{(\widehat{Q}_n^2+1)^3}\Big(\tilde{c}_5\widehat{Q}_n^4 + (-\tilde{c}_1 + 2\tilde{c}_4 - \tilde{c}_6)\widehat{Q}_n^3 + (3\tilde{c}_2 - 3\tilde{c}_5)\widehat{Q}_n^2 + (\tilde{c}_1 - 2\tilde{c}_3 + \tilde{c}_6)\widehat{Q}_n - \tilde{c}_2\Big).$$

Initialization is done by setting $x_0 = 1$, $y_0 = Q_{\text{nom}}$, $\sigma^2_{y_0}$ to a representative value of the uncertainty of the total capacity, and $\sigma^2_{x_0}$ to a representative value of the uncertainty of a difference between two state of charge estimates. Therefore, we initialize

$$\begin{aligned} \tilde{c}_{1,0} &= 1/\sigma^2_{y_0} & \tilde{c}_{4,0} &= 1/\sigma^2_{x_0} \\ \tilde{c}_{2,0} &= Q_{\text{nom}}/\sigma^2_{y_0} & \tilde{c}_{5,0} &= Q_{\text{nom}}/\sigma^2_{x_0} \\ \tilde{c}_{3,0} &= Q^2_{\text{nom}}/\sigma^2_{y_0} & \tilde{c}_{6,0} &= Q^2_{\text{nom}}/\sigma^2_{x_0}, \end{aligned}$$

and recursively compute

$$\begin{aligned} \tilde{c}_{1,n} &= \tilde{c}_{1,n-1} + x_n^2/\sigma^2_{y_n} & \tilde{c}_{4,n} &= \tilde{c}_{4,n-1} + x_n^2/\sigma^2_{x_n} \\ \tilde{c}_{2,n} &= \tilde{c}_{2,n-1} + x_n y_n/\sigma^2_{y_n} & \tilde{c}_{5,n} &= \tilde{c}_{5,n-1} + x_n y_n/\sigma^2_{x_n} \\ \tilde{c}_{3,n} &= \tilde{c}_{3,n-1} + y_n^2/\sigma^2_{y_n} & \tilde{c}_{6,n} &= \tilde{c}_{6,n-1} + y_n^2/\sigma^2_{x_n}. \end{aligned}$$

We minimize the cost function by setting its Jacobian to zero. Therefore, any of the roots of the quartic equation

$$\tilde{c}_5\widehat{Q}_n^4 + (2\tilde{c}_4 - \tilde{c}_1 - \tilde{c}_6)\widehat{Q}_n^3 + (3\tilde{c}_2 - 3\tilde{c}_5)\widehat{Q}_n^2 + (\tilde{c}_1 - 2\tilde{c}_3 + \tilde{c}_6)\widehat{Q}_n - \tilde{c}_2 = 0 \tag{4.27}$$

is a candidate solution for $\widehat{Q}_n$. We will discuss how to solve for these roots and select the optimal value in Sect. 4.18.3. The Hessian, which may be used to compute error bounds, is

$$\frac{\partial^2 \chi^2_{\text{FMAWTLS}}}{\partial \widehat{Q}_n^2} = \frac{2}{(\widehat{Q}_n^2+1)^4}\Big(-2\tilde{c}_5\widehat{Q}_n^5 + (3\tilde{c}_1 - 6\tilde{c}_4 + 3\tilde{c}_6)\widehat{Q}_n^4 + (-12\tilde{c}_2 + 16\tilde{c}_5)\widehat{Q}_n^3 + (-8\tilde{c}_1 + 10\tilde{c}_3 + 6\tilde{c}_4 - 8\tilde{c}_6)\widehat{Q}_n^2 + (12\tilde{c}_2 - 6\tilde{c}_5)\widehat{Q}_n + (\tilde{c}_1 - 2\tilde{c}_3 + \tilde{c}_6)\Big). \tag{4.28}$$

4.18.3 Solving the quartic equation

To be able to find total capacity, we must find the roots of a quartic equation, either Eq. (4.26) or (4.27). There are several methods that could be used to do so. In the following, we consider three approaches to finding the roots of the generic quartic

$$x^4 + ax^3 + bx^2 + cx + d = 0. \tag{4.29}$$

First, the roots of the quartic may be found by iterative techniques. The set of four roots of the quartic found at time step $n-1$ are used as the initial guess for the set of roots at time step n. Then, one or more Newton–Raphson iterations are performed to refine the set of roots for time step n. While this approach is conceptually simple, there are some subtle difficulties that can turn into big problems. For example, the roots may start out distinct, but over time two or more roots may come together to make a repeated root. Or, a repeated root may split into multiple distinct roots.[20] It is possible to compute how many real roots a polynomial has within an interval via Sturm's theorem,[21] and to search for only those roots using a line search, but the amount of computation required to do so is considerable.

[20] For this reason, it is unwise to use an iterative technique to track a single real root as the total-capacity estimate, because you may be tracking the wrong root! Instead, the entire set of roots should be solved every iteration.

[21] See, for example, Bruce E. Meserve, *Fundamental Concepts of Algebra*, Dover, 1982.

Second, analytic solutions exist to compute the roots of a quartic equation directly. These include Ferrari's method, Cardano's method, and others. At first, it would seem that this is the best approach to solving the quartic—all roots can be found in closed form every time. However, it turns out that all the known methods for solving quartic equations analytically have numeric instabilities.[22] The methods can be shown to be stable for only certain combinations of the signs of $\{a, b, c, d\}$ in Eq. (4.29). Further, while we know that $d < 0$ for the problem we are trying to solve, we have observed positive, negative, and zero values for b. In the examples later in this chapter, we encounter only negative values for a and c, but there is no guarantee of this in the general case. In short, using a closed-form analytic solution for the roots of the quartic is problematic, at best.

[22] See, for example, D. Herbison-Evans, "Solving quartics and cubics for graphics," *Graphics Gems V (IBM Version)*, 1995, pp. 3–15.

Third, it turns out that the roots of Eq. (4.29) are the eigenvalues of either of the following *companion matrices*:

$$\begin{bmatrix} -a & 1 & 0 & 0 \\ -b & 0 & 1 & 0 \\ -c & 0 & 0 & 1 \\ -d & 0 & 0 & 0 \end{bmatrix}, \quad \text{or} \quad \begin{bmatrix} -a & -b & -c & -d \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{or}$$

$$\begin{bmatrix} 0 & 0 & 0 & -d \\ 1 & 0 & 0 & -c \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -a \end{bmatrix}, \quad \text{or} \quad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -d & -c & -b & -a \end{bmatrix}.$$

While finding the eigenvalues of a general matrix is extremely difficult, it is considerably simpler to find the eigenvalues of a companion matrix.[23] This is also the most stable method numerically and is how MATLAB finds the zeros of a polynomial using its built-in `roots` command. This is probably the best approach for finding the roots of the quartic equation.

[23] See, for example, D.A. Bini, P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg, "A fast implicit QR eigenvalue algorithm for companion matrices," *Linear Algebra and its Applications*, 432, 2010, pp. 2006–2031.

Once the set of roots is found, however, we still need to decide which root is the one to use as the capacity estimate. Negative and complex roots can be discarded immediately, but we have sometimes encountered solutions with multiple positive real roots. Which is the capacity estimate?

The only foolproof method of which we are aware is to evaluate χ^2_{AWTLS} at each of the positive-real candidate solutions and to retain the one that gives the lowest computed value. Computing the cost function may be done very readily if we rewrite it in terms of the running summations and the total-capacity estimate. For the approximate weighted TLS solution, we have

$$\chi^2_{\text{AWTLS}} = \frac{1}{(\widehat{Q}_n^2+1)^2}\left(c_4\widehat{Q}_n^4 - 2c_5\widehat{Q}_n^3 + (c_1+c_6)\widehat{Q}_n^2 - 2c_2\widehat{Q}_n + c_3\right). \tag{4.30}$$

Similarly, for the fading memory version, we have

$$\chi^2_{\text{FMAWTLS}} = \frac{1}{(\widehat{Q}_n^2+1)^2}\left(\tilde{c}_4\widehat{Q}_n^4 - 2\tilde{c}_5\widehat{Q}_n^3 + (\tilde{c}_1+\tilde{c}_6)\widehat{Q}_n^2 - 2\tilde{c}_2\widehat{Q}_n + \tilde{c}_3\right). \tag{4.31}$$

4.18.4 Summary of approximate weighted total least squares

To use the AWTLS or FMAWTLS solution, we first initialize the six recursive variables. Then, every time a data point becomes available, we update the recursive variables and solve the quartic equation Eq. (4.26) or (4.27) for the four possible capacity estimates. We substitute these capacity estimates back into the cost function Eq. (4.30) or (4.31) and choose the value that gives the lowest cost.

Note that the AWTLS cost function does not equal the WTLS cost function when $\sigma_{x_i} = k\sigma_{y_i}$. This can be remedied easily by defining scaled measurements $\tilde{y}_i = ky_i$. Then $\sigma_{\tilde{y}_i} = \sigma_{x_i}$. We invoke the AWTLS or FMAWTLS methods to find total capacity estimate $\widehat{Q}_n$ and Hessian H_n using input sequences composed of the original x vector and the scaled $\tilde{\mathbf{y}}$ vector (i.e., $(x_i, \tilde{y}_i)$ with corresponding variances $(\sigma^2_{x_i}, k^2\sigma^2_{y_i})$). The true slope estimate can be found as $\widehat{Q}_{n,\text{corrected}} = \widehat{Q}_n/k$ and the corrected Hessian can be found as $H_{n,\text{corrected}} = k^2H_n$. This is the method used in the simulation results in Sect. 4.20 and 4.21, where the proportionality constant is estimated as $k = \sigma_{x_1}/\sigma_{y_1}$. This scaling improves results even when σ_{y_i} and σ_{x_i} are not proportionally related if k is chosen to give an order of magnitude proportionality or average proportionality between the uncertainties of x_i and y_i.

In summary, these AWTLS solutions share the nice properties of the WLS solution:

1. They give a closed-form solution for $\widehat{Q}_n$. No iteration is required. The only complication is the need for finding the roots of a quartic

polynomial, but a manageable approach to doing so has been proposed.

2. The solution can be computed very easily in a recursive manner. We keep track of the six running sums $c_{1,n}$ through $c_{6,n}$. When an additional data point becomes available, we update the sums and compute an updated total-capacity estimate.
3. Fading memory can be added easily to allow the estimate $\widehat{Q}_n$ to place greater emphasis on more recent measurements than on earlier measurements, allowing adaptation of $\widehat{Q}_n$ to adjust for true cell total capacity changes.
4. Further, this method is superior to the PTLS solution since it allows individual weighting on the x_i and y_i data points.

4.19 *Code to simulate the methods*

The regressive total-capacity estimation algorithms are implemented in the MATLAB code `xLSalgos.m`. The function begins with some comments that describe its input and output arguments:

```
% Tests the recursive performance of the xLS algorithms on a particular
% dataset
% [Qhat,SigmaQ,Fit] = xLSalgos(measX,measY,SigmaX,SigmaY,gamma,Qnom)
%   - measX = noisy z(2)-z(1)
%   - measY = noisy integral(i(t)/3600 dt)
%   - SigmaX = variance of X
%   - SigmaY = variance of Y
%   - gamma = geometric forgetting factor (gamma = 1 for perfect memory)
%   - Qnom = nominal value of Q: if nonzero, used to initialize recursions
%
%   - Qhat = estimate of capacity at every time step
%     - column 1 = WLS - weighted, recursive
%     - column 2 = WTLS - weighted, but not recursive
%     - column 3 = SCTLS - scaled confidence PTLS; recursive and weighted,
%                  but using SigmaX(1) and SigmaY(1) only to determine
%                  factor by which all SigmaX and SigmaY are assumed to be
%                  related
%     - column 4 = AWTLS - recursive and weighted
%   - SigmaQ = variance of Q, computed via Hessian method (columns
%                  correspond to methods in the same way as for Qhat)
%   - Fit = goodness of fit metric for each method (columns
%                  correspond to methods in the same way as for Qhat)
function [Qhat,SigmaQ,Fit]=xLSalgos(measX,measY,SigmaX,SigmaY,gamma,Qnom)
```

Next, memory is reserved for the function output matrices and the proportionality constant k between σ_x and σ_y is estimated as $k = \sigma_{x_1}/\sigma_{y_1}$. The recursive algorithm parameters are initialized to zero. If, however, we have known nonzero Q_{nom}, then we can initialize to better values. Note that uppercase "`C`" variables are scaled fading-memory recursive parameters for use with AWTLS and lowercase "`c`" variables are fading-memory recursive parameters for all other

12:07:34.

methods (except for nonrecursive WTLS). The recursive parameters are then updated based on the current measurement.

```
% Reserve some memory
Qhat = zeros(length(measX),4); SigmaQ = Qhat; Fit = Qhat;
K = sqrt(SigmaX(1)/SigmaY(1));

% Initialize some variables used for the recursive methods
c1 = 0; c2 =0; c3 = 0; c4 = 0; c5 = 0; c6 = 0;
C1 = 0; C2 =0; C3 = 0; C4 = 0; C5 = 0; C6 = 0;
if Qnom ~= 0,
  c1 = 1/SigmaY(1); c2 = Qnom/SigmaY(1); c3 = Qnom^2/SigmaY(1);
  c4 = 1/SigmaX(1); c5 = Qnom/SigmaX(1); c6 = Qnom^2/SigmaX(1);
  C1 = 1/(K^2*SigmaY(1)); C2 = K*Qnom/(K^2*SigmaY(1));
  C3 = K^2*Qnom^2/(K^2*SigmaY(1));
  C4 = 1/SigmaX(1); C5 = K*Qnom/SigmaX(1); C6 = K^2*Qnom^2/SigmaX(1);
end

for iter = 1:length(measX),
  % Compute some variables used for the recursive methods
  c1 = gamma*c1 + measX(iter)^2/SigmaY(iter);
  c2 = gamma*c2 + measX(iter)*measY(iter)/SigmaY(iter);
  c3 = gamma*c3 + measY(iter)^2/SigmaY(iter);
  c4 = gamma*c4 + measX(iter)^2/SigmaX(iter);
  c5 = gamma*c5 + measX(iter)*measY(iter)/SigmaX(iter);
  c6 = gamma*c6 + measY(iter)^2/SigmaX(iter);

  C1 = gamma*C1 + measX(iter)^2/(K^2*SigmaY(iter));
  C2 = gamma*C2 + K*measX(iter)*measY(iter)/(K^2*SigmaY(iter));
  C3 = gamma*C3 + K^2*measY(iter)^2/(K^2*SigmaY(iter));
  C4 = gamma*C4 + measX(iter)^2/SigmaX(iter);
  C5 = gamma*C5 + K*measX(iter)*measY(iter)/SigmaX(iter);
  C6 = gamma*C6 + K^2*measY(iter)^2/SigmaX(iter);
```

Next, the fading-memory recursive capacity estimate using WLS is evaluated via Eq. (4.8). Its Hessian is computed via Eq. (4.21) to be able to evaluate the estimate's error-bounds variable `SigmaQ`. The value of the cost function χ^2_{FMWLS} is computed using Eq. (4.9) and used to compute goodness of fit, stored in `Fit`, using MATLAB's implementation of the complementary incomplete gamma function and Eq. (4.19).

```
  % Method 1: WLS
  Q = c2./c1; Qhat(iter,1) = Q;
  H = 2*c1;   SigmaQ(iter,1) = 2/H;
  J = Q.^2.*c1 -2*Q.*c2 + c3;
  Fit(iter,1) = gammainc(J/2,(iter-1)/2,'upper');
```

The next code segment computes the present total-capacity estimate using the WTLS method. This method is not recursive, and requires executing multiple Newton–Raphson search iterations to converge toward the solution every time a new data point is measured. Each WTLS solution is initialized with the WLS estimate just computed, and then the cost-function Jacobian and Hessian matrices are repeatedly computed using Eq. (4.17) and (4.18). These are used with Eq. (4.15) to converge toward the WTLS solution. The optimized

12:07:34.

cost-function value is computed via Eq. (4.13) and used to find the goodness of fit of the model.

```
    % Method 2: WTLS -- not recursive
    g = flipud((gamma.^(0:(iter-1)))');   % vector of forgetting factors
    x = measX(1:iter); y = measY(1:iter); % all measurements until now
    sx = sqrt(SigmaX(1:iter));            % all sigma-x until now
    sy = sqrt(SigmaY(1:iter));            % all sigma-y until now
    Q = Qhat(iter,1); % initialize WTLS with WLS total capacity estimate
    for kk = 1:10,    % ten Newton--Raphson iterations
      jacobian = sum(g.*(2*(Ctls2*x-y).*(Ctls2*y.*sx.^2+x.*sy.^2))./...
                    ((Ctls2^2*sx.^2+sy.^2).^2));
      hessian = sum(g.*(2*sy.^4.*x.^2+sx.^4.*...
                   (6*Ctls2^2*y.^2-4*Ctls2^3*x.*y) - ...
                    sx.^2.*sy.^2.*...
                   (6*Ctls2^2*x.^2-12*Ctls2*x.*y+2*y.^2))./...
                   ((Ctls2^2*sx.^2+sy.^2).^3));
      Q = Q - jacobian/hessian;
    end
    Qhat(iter,2) = Q;                     % save capacity estimate
    SigmaQ(iter,2) = 2/hessian;           % save bounds
    J = sum(g.*(y-Q*x).^2./(sx.^2*Q^2+sy.^2)); % cost-function value
    Fit(iter,2) = gammainc(J/2,(2*iter-1)/2,'upper'); % goodness of fit
```

Next, we implement the PTLS method. The total-capacity estimate is computed using Eq. (4.23), its Hessian via Eq. (4.24), and the resulting minimum cost with Eq. (4.25).

```
    % Method 3: PTLS
    Q = (-c1+K^2*c3+sqrt((c1-K^2*c3)^2+4*K^2*c2^2))/(2*K^2*c2);
    Qhat(iter,3) = Q;
    H = ((-4*K^4*c2)*Q^3+6*K^4*c3*Q^2+...
         (-6*c1+12*c2)*K^2*Q+2*(c1-K^2*c3))/(Q^2*K^2+1)^3;
    SigmaQ(iter,3) = 2/H;
    J = (Q^2*c1 -2*Q*c2 + c3)/(Q^2*K^2+1);
    Fit(iter,3) = gammainc(J/2,(2*iter-1)/2,'upper');
```

Finally, we implement the AWTLS method. We first search for roots of the quartic equation of Eq. (4.27). We discard any that are not positive and real. Then, we evaluate the cost function of Eq. (4.31) using the remaining candidate roots and keep the root that minimizes the cost function. The Hessian is computed via Eq. (4.28).

```
    % Method 4: AWTLS with pre-scaling
    r = roots([C5 (-C1+2*C4-C6) (3*C2-3*C5) (C1-2*C3+C6) -C2]);
    r = r(r==conj(r)); % discard complex-conjugate roots
    r = r(r>0); % discard negative roots
    Jr = ((1./(r.^2+1).^2).*(r.^4*C4-2*C5*r.^3+(C1+C6)*r.^2-2*C2*r+C3))';
    J = min(Jr);
    Q = r(Jr==J); % keep Q that minimizes cost function
    H = (2/(Q^2+1)^4)*...
          (-2*C5*Q^5+(3*C1-6*C4+3*C6)*Q^4+(-12*C2+16*C5)*Q^3 ...
          +(-8*C1+10*C3+6*C4-8*C6)*Q^2+(12*C2-6*C5)*Q+(C1-2*C3+C6));
    Qhat(iter,4) = Q/K;
    SigmaQ(iter,4) = 2/H/K^2;
    Fit(iter,4) = gammainc(J/2,(2*iter-1)'/2,'upper');
  end
  Fit = real(Fit);
return
```

12:07:34.

4.20 *Example HEV simulations*

In closing, we examine a number of usage scenarios to exercise the WLS, WTLS, PTLS, and AWTLS total-capacity estimation methods and to compare their performance. All scenarios use the fading-memory version of the four methods, but we omit the prefix "FM" for brevity. Unless otherwise stated, the fading-memory forgetting factor is chosen as $\gamma = 1.0$.

We assume that the individual SOC estimates that are input to these methods can be determined to an accuracy of $\sigma_z = 0.01$. This is being very generous, since the best method of which we are aware, SPKF, achieves only around $\sigma_z = 0.01$ for LMO or NMC cells and $\sigma_z = 0.03$ for LFP cells in practice when Q_{nom} is used instead of Q in the estimator. Other methods that we have used, such as EKF, achieve around $\sigma_z = 0.02$ or higher for LMO cells in practice. A nice advantage of both EKF and SPKF is that they give dynamic estimates of the variance of the state of charge estimate that ensure that the values of σ_{x_i} used in total-capacity estimation are accurate.

We use computer simulation rather than cell testing to validate the algorithms because it allows us to constrain a variety of factors that would be difficult to control in a real-time embedded system. These include:

- The efficacy and accuracy of the SOC estimation algorithms used to provide input to the total-capacity estimation algorithms;
- The accuracy and precision of the raw sensor measurements used as input (including the challenges of bias errors, nonlinear errors, and random errors, for example);
- The repeatability of the experiment; and
- The fact that total capacity of a physical cell fades over time and the associated difficulty or even impossibility of knowing the true value of total capacity against which to compare our results.

Therefore, we choose to use synthetic data to isolate the performance of the total-capacity estimation algorithms themselves, when all other factors are in some sense idealized. The x_i and y_i values are mathematically generated, as described in the individual subsections below.

4.20.1 *HEV application, scenario 1*

The first sets of simulations that we present are for HEV scenarios. From the perspective of total-capacity estimation, these applications are characterized by the narrow window of battery-cell SOC used.

We assume that the vehicle uses a SOC range of 40 % to 60 %. Therefore, each time the total capacity estimate is updated, the true change in SOC can range from -0.2 to $+0.2$. We simulate this by choosing the true value of x_i to be a uniform random number selected between these limits.

The HEV application is also characterized by the fact that the battery pack is never fully charged to a precisely known SOC; therefore, each time the total capacity estimate is updated, two estimates of SOC are required to compute $x_i = \hat{z}^{(i)}_{k_2} - \hat{z}^{(i)}_{k_1}$. This gives an overall $\sigma^2_x = 2\sigma^2_z = 2(0.01)^2$. We simulate this by computing the measured value of x_i to be equal to the true value of x_i added to a zero-mean Gaussian random number having variance σ^2_x.

We compute the true value of y_i to be equal to the nominal capacity of the cell Q_{nom} multiplied by the true value of x_i. Noise on the y_i measurement is assumed to comprise accumulated quantization noises. For y_i computed by summing m_i measurements, taken at a 1 Hz rate, from a sensor having quantizer resolution q, the total noise is $\sigma^2_{y_i} = q^2 m_i/(12 \times 3{,}600^2)$.[24] For HEV scenario 1, we assumed that the maximum range of the current sensor is $\pm 30Q_{\text{nom}}$ and that a 10-bit analog-to-digital converter is used to measure current. This leads to $q = 60Q_{\text{nom}}/1024$. We chose $m_i = 300\,\text{s}$ for every measurement and a nominal capacity of $Q_{\text{nom}} = 10\,\text{Ah}$.

[24] This is the variance of a uniform random variable distributed between $-q/2$ and $q/2$, scaled by $m_i/3{,}600^2$

To implement this scenario, we use the following code:

```
Q0 = 10;            % initial true cell total capacity
slope = 0;          % total capacity does not change with time
maxI = 30*Q0;       % must be able to measure current up to +/- maxI
precisionI = 1024; % 10-bit precision on current sensor
Qnom = 0;           % initialization for recursive variables
xmax = 0.2;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = 300;            % interval length between measurements
socnoise = sqrt(2)*0.01; % sigma_x
gamma = 1;          % fading-memory forgetting factor (no forgetting)
plotTitle = 'HEV Scenario 1'; % for the output plot
runScenario
```

This code calls `runScenario.m`, which we now describe. The first part of this script synthesizes random truth values for x_i and y_i and the corresponding noisy values thereof. Most scenarios have a constant-length interval, corresponding to a scalar value of the `m` variable; however, if `m` is not a scalar, a variable-length interval is required. The details of data generation for a variable-length interval are discussed in Sect. 4.21.2. The second part of the script calls the `xLSalgos.m` function, which has already been described, and plots results:

```
% runScenario.m: run a particular scenario and plot results

% Make up some data
n = 1000;              % number of data points
Q = (Q0+slope*(1:n))'; % true capacity as a function of time
x = ((xmax-xmin)*rand(n,1)+xmin); % true values of "x" measurements
y = Q.*x;              % true values of "y" measurements
```

```
% Add in some noise to both variables.
binsize = 2*maxI/precisionI;
rn1 = ones(n,1);        % ones the size of the "x" measurements
if isscalar(m),         % constant number of measurements per iteration
  rn2 = rn1;            % ones the size of the "y" measurements
  sy = binsize*sqrt(m/12)/3600*rn2; % sigma-y for each measurement
else                    % variable number of measurements per iteration
  mu = log(m(1))+m(2)^2; % input to lognrnd command
  m = 3600*lognrnd(mu,sigma,n,1); % create log-normal rand variables
  sy = binsize*sqrt(m/12)/3600; % sigma-y for each measurement
end
sx = socnoise*rn1;      % sigma-x for each measurement

x = x + sx.*randn(n,1); % noisy measurement of "x"
y = y + sy.*randn(n,1); % noisy measurement of "y"

% Call the xLSalgos.m function. Returned variables have format:
%  - column 1 = WLS - weighted, recursive
%  - column 2 = WTLS - weighted, but not recursive
%  - column 3 = PTLS - recursive weighted, but using SigmaX(1) and
%               SigmaY(1) only (i.e., simplified method using c0 and c1)
%  - column 4 = AWTLS - recursive and weighted
[Qhat,SigmaQ,Fit] = xLSalgos(x,y,sx.^2,sy.^2,gamma,Qnom);
Qrep = repmat(Q,1,size(Qhat,2));

figure(1); clf; plot(Qhat); hold on; box on;
xlabel('Algorithm update index'); ylabel('Capacity estimate (Ah)');
title(sprintf('%s: Capacity Estimates with Error Bounds',...
              plotTitle));
legend('WLS','WTLS','TLS','AWTLS','location','northeast');
plot(Qhat+3*sqrt(SigmaQ),'linewidth',0.5); % error bounds
plot(Qhat-3*sqrt(SigmaQ),'linewidth',0.5);
plot(Qhat);       % overlay true capacity one more time so plots on top
plot(1:n,Q,'k:'); % plot true capacity

figure(2); clf; plot(Fit); hold on; box on;
xlabel('Algorithm update index'); ylabel('Goodness of fit');
title(sprintf('%s: Goodness of fit for each method',plotTitle))
legend('WLS','WTLS','TLS','AWTLS','location','east'); ylim([-0.02 1.02]);
```

The recursive parameters were initialized to zero prior to the first data point being received. Results are presented in Fig. 4.15. The top frame of the figure shows estimates made using the four methods evolving over time as thick lines, and their three-sigma error bounds computed using the Hessian method as thin lines. We see that WTLS, PTLS, and AWTLS give identical estimates and error bounds for this scenario and converge to the neighborhood of the true total capacity. The WLS estimate is biased and its error bounds are (incorrectly) so tight that they are indistinguishable from the estimate itself.

The bottom frame of the figure shows the goodness-of-fit metric as applied to the four methods. Again, WTLS, PTLS, and AWTLS give identical results, quickly converging to a value of 1.0 (that is, these methods are confident that their estimate is reliable). The WLS method returns a vanishingly small value for goodness of fit, re-

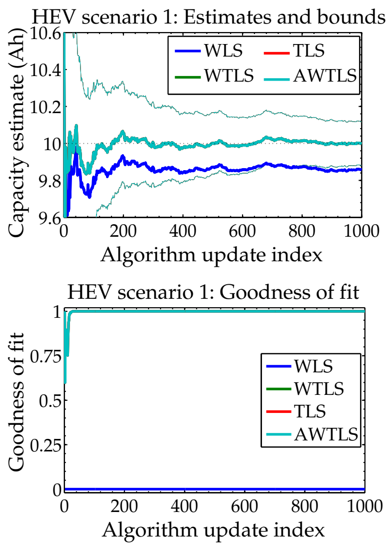


Figure 4.15: Simulation results for HEV scenario 1.
(Reproduced from Fig. 3 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

flecting the fact that the method does not give a good value for its total-capacity estimates and/or the bounds thereon.

4.20.2 *HEV application, scenario 2*

The second HEV scenario is identical to the first except that the recursive methods are initialized with a total capacity estimate before any measurements are received. In this case, the methods were initialized with a nominal capacity estimate of 9.9 Ah (the true total capacity was still 10.0 Ah).

To implement this scenario, we use the following code:

```
Q0 = 10;            % initialize true cell total capacity
slope = 0;          % total capacity does not change with time
maxI = 30*Q0;       % must be able to measure current up to +/- maxI
precisionI = 1024; % 10-bit precision on current sensor
Qnom = 0.99*Q0;     % initialization for recursive variables
xmax = 0.2;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = 300;            % interval length between measurements
socnoise = sqrt(2)*0.01; % sigma_x
gamma = 1;          % fading-memory forgetting factor (no forgetting)
plotTitle = 'HEV Scenario 2'; % for the output plot
runScenario
```

Results from running this code are presented in Fig. 4.16. In this scenario, PTLS and AWTLS give identical results for their estimates, error bounds, and goodness of fit. WTLS cannot be calculated recursively, so its estimate cannot be initialized; subsequently, the WTLS results are the same as for scenario 1. Once again, WLS is inferior to the other methods. PTLS and AWTLS give the most accurate and most confident estimates because the ability to initialize the estimate with a reasonable value has resulted in of tighter error bounds while maintaining a high value for goodness of fit.

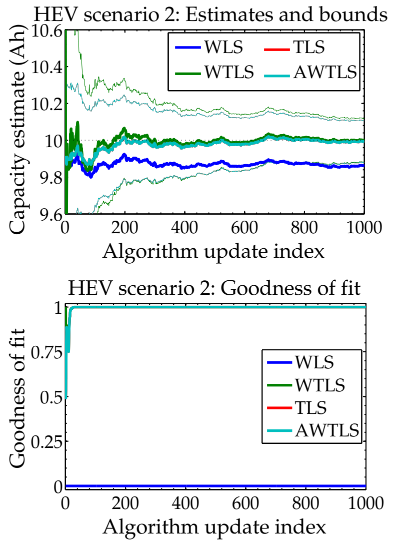


Figure 4.16: Simulation results for HEV scenario 2.
(Reproduced from Fig. 4 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

4.20.3 *HEV application, scenario 3*

In the third HEV scenario, we explore the ability of the algorithms to track a moving value of total capacity. This scenario is identical to HEV scenario 2, except that the true total capacity is changing with a slope of -0.001 Ah per measurement update, and a fading memory forgetting factor of $\gamma = 0.99$ is used for all methods.

To implement this scenario, we use the following code:

```
Q0 = 10;            % initialize true cell total capacity
slope = -0.001;     % true capacity changes this much every iteration
maxI = 30*Q0;       % must be able to measure current up to +/- maxI
precisionI = 1024; % 10-bit precision on current sensor
Qnom = 0.99*Q0;     % initialization of recursive variables
xmax = 0.2;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = 300;            % interval length between measurements
```

```
socnoise = sqrt(2)*0.01; % sigma_x
gamma = 0.99;      % fading-memory forgetting factor (slow forgetting)
plotTitle = 'HEV Scenario 3'; % for the output plot
runScenario
```

Results are presented in Fig. 4.17, where the true total capacity is drawn as a dotted black line. In this example, the WLS method *appears* to give good results, but its goodness of fit value is still vanishingly small because the error bounds are unreasonably tight and almost never surround the true value of total capacity. WTLS, PTLS, and AWTLS are also able to track the moving value of total capacity—PTLS and AWTLS give the best results due to the ability to initialize their recursive parameters with reasonable values, yielding better estimates having narrower error bounds.

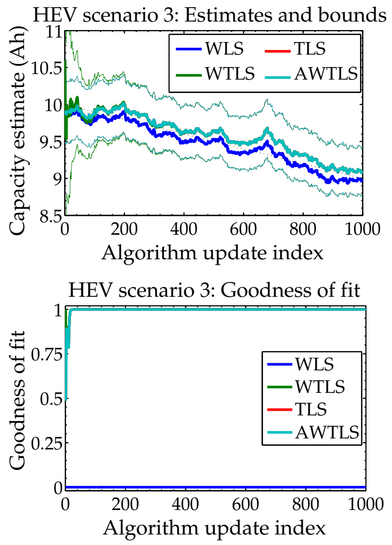


Figure 4.17: Simulation results for HEV scenario 3.
(Reproduced from Fig. 5 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

4.21 *Example EV simulations*

4.21.1 *EV application, scenario 1*

The next scenarios that we consider are typical of EV and PHEV (or E-REV) operation. These are different from the HEV application in several respects: the battery total capacity is larger, the relative rate of energy usage is lower, the range of SOC used by the vehicle is larger, and the EV battery pack is sometimes fully charged to a known setpoint.

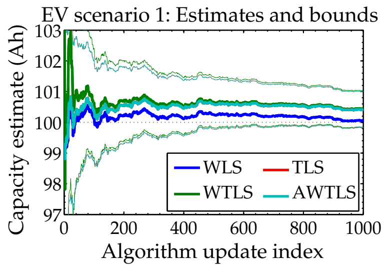
In all cases, we consider a battery pack having total capacity of $Q = 100$ Ah, and a maximum rate of $\pm 5Q$. We again assume a 10-bit current sensor, which gives $q = 10Q_{\text{nom}}/1024$ and $\sigma^2_{y_i} = q^2 m_i/(12 \times 3600^2)$. For the first EV scenario we assume that the total-capacity estimate is updated on a regular basis as the vehicle operates, with $m_i = 7200$ s (i.e., every 2 h or about 120 mi of highway-driven distance). We assume that the battery SOC can change by ± 40 % in that interval, so the true value of x_i is chosen to be a uniform random variable between -0.4 and $+0.4$. Again, noise on x_i is Gaussian with variance $\sigma^2_{x_i} = 2(0.01)^2$. The recursive methods are initialized with an initial total-capacity estimate of 99 Ah.

To implement this scenario, we use the following code:

```
Q0 = 100;           % initialize true cell total capacity
slope = 0;          % true capacity changes this much every iteration
maxI = 5*Q0;        % must be able to measure current up to +/- maxI
precisionI = 1024; % 10-bit precision on current sensor
Qnom = 0.99*Q0;     % initialization of recursive variables
xmax = 0.4;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = 7200;           % interval length between measurements
socnoise = sqrt(2)*0.01; % sigma_x
gamma = 1;          % fading-memory forgetting factor (no forgetting)
plotTitle = 'EV Scenario 1'; % for the output plot
runScenario
```

12:07:34.

Representative results of this scenario are presented in Fig. 4.18. These are very similar in most respects to the HEV scenario 2 results. Again, WLS fails because its error bounds are far too tight, leading to a vanishingly small goodness of fit. WTLS, PTLS, and AWTLS all give good results, with PTLS and AWTLS giving the best results due to lower error bounds because of the possibility of initialization.



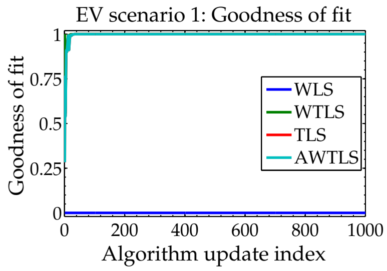


Figure 4.18: Simulation results for EV scenario 1.
(Reproduced from Fig. 6 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

4.21.2 *EV application, scenario 2*

The asymptotic quality of the total-capacity estimates is limited by the noise on the state of charge estimation error. If this noise can be reduced, the total-capacity estimates can become much more accurate. The EV application allows a means to do this: whenever the battery pack is fully charged, we have a precisely known endpoint SOC. Therefore, either $\hat{z}_{k_2}$ or $\hat{z}_{k_1}$ can be known exactly for every total capacity estimate update. This then allows us to use $\sigma_{x_i}^2 = \sigma_z^2 = (0.01)^2$, which is half that used when neither SOC end point is known exactly.

The tradeoff is that we no longer have regular updates. Instead, updates happen randomly, whenever the vehicle is charged. Therefore, m_i becomes a random variable. For the results presented in this section, we treat m_i as a lognormal random variable with mode 0.5 h and standard deviation 0.6 h. This pdf is plotted in Fig. 4.19. Other pdfs could easily be used: this one was chosen to give reasonable-duration drive cycles that encompass a variety of driving behaviors and distances. Also, since a greater fraction of the battery pack would be used for an entire drive cycle than for a regular periodic update, we use an 80 % range of SOC, so the true value of x_i is computed to be a uniform random number from -0.8 to $+0.8$.

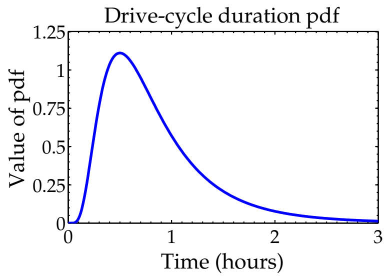


Figure 4.19: Example drive-duration pdf.
(Reproduced from Fig. 7 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

To implement this scenario, we use the following code:

```
Q0 = 100;           % initialize true cell total capacity
slope = 0;          % true capacity changes this much every iteration
maxI = 5*Q0;        % must be able to measure current up to +/- maxI
precisionI = 1024;  % 10-bit precision on current sensor
Qnom = 0.99*Q0;     % initialization of recursive variables
xmax = 0.8;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = [0.5, 0.6];     % mode = 0.5; sigma = 0.6 in pdf
socnoise = 0.01;    % sigma_x
gamma = 1;          % fading-memory forgetting factor (no forgetting)
plotTitle = 'EV Scenario 2'; % for the output plot
runScenario
```

Results from this scenario are presented in Fig. 4.20. WLS fails once again. However, this time PTLS also fails because $\sigma_{x_i} \neq k\sigma_{y_i}$ due to the variable-length drive cycles. The estimate given by PTLS is actually quite reasonable, but the goodness of fit is very small. WTLS gives good results, and AWTLS gives the best results (based on its lower error bounds) because of the ability to initialize the estimate. Note that the asymptotic three-sigma error bounds drop from about $\pm 1\,\%$ for HEV scenario 1 to about $\pm 0.15\,\%$ of the true total capacity in EV scenario 2 due to having a lower value of σ_{x_i} and also due to the wider range in x_i.

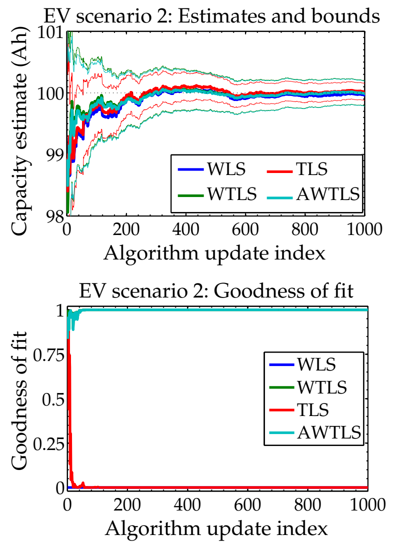


Figure 4.20: Simulation results for EV scenario 2.
(Reproduced from Fig. 8 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

4.21.3 *EV application, scenario 3*

The final scenario that we consider is identical to EV scenario 2, except that we simulate a changing total capacity. The slope of the total capacity curve is chosen to be -0.01 Ah per measurement update, and $\gamma = 0.98$ was used.

To implement this scenario, we use the following code:

```
Q0 = 100;           % initialize true cell total capacity
slope = -0.01;      % true capacity changes this much every iteration
maxI = 5*Q0;        % must be able to measure current up to +/- maxI
precisionI = 1024; % 10-bit precision on current sensor
Qnom = 0.99*Q0;     % initialization of recursive variables
xmax = 0.8;         % maximum change in state of charge per measurement
xmin = -xmax;       % minimum change in state of charge per measurement
m = [0.5, 0.6];     % mode = 0.5; sigma = 0.6 in pdf
socnoise = 0.01;    % sigma_x
gamma = 0.98;       % fading-memory forgetting factor (slow forgetting)
plotTitle = 'EV Scenario 3'; % for the output plot
runScenario
```

Representative results of this scenario are presented in Fig. 4.21. Once again, WLS fails and PTLS is uncertain of its estimate for nearly 100 updates. However, PTLS does recover and do quite well. The AWTLS method gives the best results.

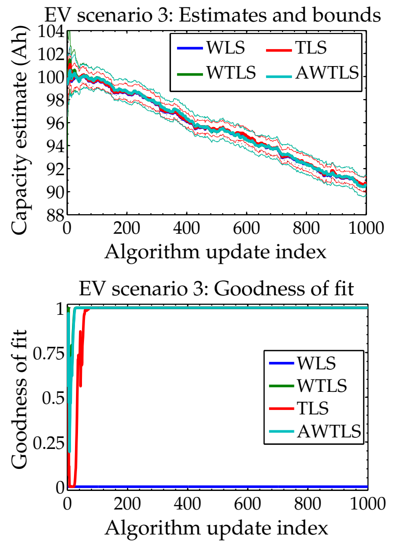


Figure 4.21: Simulation results for EV scenario 3.
(Reproduced from Fig. 9 of Plett, G.L., "Recursive Approximate Weighted Total Least Squares Estimation of Battery Cell Total Capacity," *Journal of Power Sources*, 196(4), 2011, pp. 2,319–31.)

4.22 *Discussion of simulations*

The simulation results have illustrated a few key properties of the four methods we have introduced to estimate total capacity. They confirm that noise on the state of charge estimates used as input to the total-capacity estimator must be considered in order to estimate battery-cell total capacity properly. Least squares, weighted least squares, and other similar methods that do not consider this noise simply fail. They give biased estimates of total capacity and have unreliable error bounds. Methods related to total least squares, where noises on the SOC estimates are explicitly recognized and incorporated in the calculations, are required for reliable total-capacity estimation.

In principle, WTLS always gives the best results. However, we have seen that the PTLS and AWTLS methods can give better results in practice because they can be initialized with a nominal-capacity estimate. Furthermore, since PTLS and AWTLS give nice recursive solutions, one of them should always be used instead of WTLS.

If the measurement update interval m_i is constant and therefore $\sigma_{x_i} = k\sigma_{y_i}$ for all measurements, PTLS and AWTLS give identical results. Therefore, the simpler PTLS method is preferred. However, if $\sigma_{x_i} \neq k\sigma_{y_i}$, the AWTLS method gives better results than PTLS and sometimes PTLS fails. This is particularly important for the EV application where updates to the estimate of total capacity are done when charging the battery: these yield a greatly improved total-capacity estimate because of the reduction in σ_{x_i} due to knowing one SOC value exactly. AWTLS always gives results at least as good as the other methods.

The noises that contribute to σ_{y_i} are assumed to be due to current-sensor errors. In practice, these can include gain errors, bias errors, noise errors, and nonlinear errors. We have considered only noise errors here. Gain errors and nonlinear errors will bias all of the methods; however, we believe that the biased value of the total-capacity estimate will be consistent with the perceived capacity of the battery cell if the same current sensor is used to compute the battery-cell total-capacity estimate and to monitor pack operations. Bias error can be subtracted in a EV setting if we can assume that the coulombic efficiency of the cells is $\eta \approx 1$ by matching the discharged ampere-hours from usage with the charged ampere-hours when charging.

The error bounds on the total-capacity estimate, even with the optimum WTLS estimator, are larger than some might expect. This underscores the need for a method that predicts not only the estimate but also dynamic error bounds on the estimate, as do the ones proposed in this chapter. Without dynamic error bounds, the user of the total-capacity estimate has no idea how good or bad that estimate is. If the estimate is used to compute battery-pack available energy, for example, the energy estimate may be overly optimistic or overly pessimistic, neither of which is acceptable.

4.23 *Where to from here?*

We have now looked at the fundamental state- and parameter-estimation problems that must be solved by a BMS. These estimates can be fed into energy- and power-estimation algorithms. We have already seen some simple examples of this in Chap. 1. We will look at more advanced methods in Chaps. 6 and 7. Next, however, we look at the somewhat simpler but still very important issue of balancing or equalizing the cells in a battery pack.

4.24 *Appendices: Nonlinear Kalman-filter algorithms*

The following pages have summary tables for the nonlinear Kalman-filter based algorithms developed in this chapter.

Appendix: Nonlinear EKF for parameter estimation

Nonlinear state-space model:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \boldsymbol{r}_k,\\ \boldsymbol{d}_k &= \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}_k, \boldsymbol{e}_k).\end{aligned}$$

where $\boldsymbol{r}_k$ and $\boldsymbol{e}_k$ are independent Gaussian noise processes with means zero and $\bar{\boldsymbol{e}}$, respectively, and having covariance matrices $\boldsymbol{\Sigma}_{\tilde{r}}$ and $\boldsymbol{\Sigma}_{\tilde{e}}$, respectively.

Definitions:

$$\hat{\boldsymbol{C}}_k^{\theta} = \left.\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{\theta},\boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k^-} \qquad \hat{\boldsymbol{D}}_k^{\theta} = \left.\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{x}_k,\boldsymbol{u}_k,\boldsymbol{\theta},\boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{e}_k}\right|_{\boldsymbol{e}_k=\bar{\boldsymbol{e}}_k}$$

Caution: Be careful to compute $\hat{\boldsymbol{C}}_k^{\theta}$ using the recursive chain rule described in the chapter!

Initialization: For $k = 0$, set

$$\begin{aligned}\hat{\boldsymbol{\theta}}_0^+ &= \mathbb{E}[\boldsymbol{\theta}_0]\\ \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},0}^+ &= \mathbb{E}\big[(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)^T\big].\\ \frac{\mathrm{d}\boldsymbol{x}_0}{\mathrm{d}\boldsymbol{\theta}} &= 0, \text{ unless side information is available.}\end{aligned}$$

Computation: For $k = 1, 2, \ldots$ compute:

Param.-prediction time update: $\hat{\boldsymbol{\theta}}_k^- = \hat{\boldsymbol{\theta}}_{k-1}^+$

Error-covariance time update: $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k-1}^+ + \boldsymbol{\Sigma}_{\tilde{r}}$

Kalman gain matrix: $\boldsymbol{L}_k^{\theta} = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^-(\hat{\boldsymbol{C}}_k^{\theta})^T[\hat{\boldsymbol{C}}_k^{\theta}\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^-(\hat{\boldsymbol{C}}_k^{\theta})^T + \hat{\boldsymbol{D}}_k^{\theta}\boldsymbol{\Sigma}_{\tilde{e}}(\hat{\boldsymbol{D}}_k^{\theta})^T]^{-1}$

Param.-estimate meas. update: $\hat{\boldsymbol{\theta}}_k^+ = \hat{\boldsymbol{\theta}}_k^- + \boldsymbol{L}_k^{\theta}[\boldsymbol{d}_k - \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{e}}_k)]$

Error-covariance meas. update: $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^+ = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- - \boldsymbol{L}_k^{\theta}\boldsymbol{\Sigma}_{\tilde{d},k}(\boldsymbol{L}_k^{\theta})^T$

12:07:34.

Appendix: Nonlinear SPKF for parameter estimation

Nonlinear state-space model:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{r}_k,$$
$$\boldsymbol{d}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}_k, \boldsymbol{e}_k).$$

where $\boldsymbol{r}_k$ and $\boldsymbol{e}_k$ are independent Gaussian noise processes with means zero and $\bar{\boldsymbol{e}}$, respectively, and having covariance matrices $\boldsymbol{\Sigma}_{\tilde{r}}$ and $\boldsymbol{\Sigma}_{\tilde{e}}$, respectively.

Definitions: Let

$$\boldsymbol{\theta}_k^a = \big[\boldsymbol{\theta}_k^T,\ \boldsymbol{e}_k^T\big]^T, \quad \boldsymbol{\mathcal{W}}_k^a = \big[(\boldsymbol{\mathcal{W}}_k^\theta)^T,\ (\boldsymbol{\mathcal{W}}_k^e)^T\big]^T, \quad p = 2 \times \dim(\boldsymbol{\theta}_k^a).$$

Initialization: For $k = 0$, set

$$\hat{\boldsymbol{\theta}}_0^+ = \mathbb{E}\big[\boldsymbol{\theta}_0\big] \qquad \hat{\boldsymbol{\theta}}_0^{a,+} = \mathbb{E}\big[\boldsymbol{\theta}_0^a\big] = \big[(\hat{\boldsymbol{\theta}}_0^+)^T,\ \bar{\boldsymbol{e}}\big]^T$$
$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},0}^+ = \mathbb{E}\big[(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)^T\big] \qquad \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},0}^{a,+} = \mathbb{E}\big[(\boldsymbol{\theta}_0^a - \hat{\boldsymbol{\theta}}_0^{a,+})(\boldsymbol{\theta}_0^a - \hat{\boldsymbol{\theta}}_0^{a,+})^T\big]$$
$$= \operatorname{diag}\big(\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},0}^+, \boldsymbol{\Sigma}_{\tilde{e}}\big).$$

Computation: For $k = 1, 2, \ldots$ compute:

Parameter-prediction time update: $\hat{\boldsymbol{\theta}}_k^- = \hat{\boldsymbol{\theta}}_{k-1}^+$

Error-covariance time update: $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k-1}^+ + \boldsymbol{\Sigma}_{\tilde{r}}$

Output estimate:
$$\boldsymbol{\mathcal{W}}_k^{a,-} = \Big\{\hat{\boldsymbol{\theta}}_k^{a,-}, \hat{\boldsymbol{\theta}}_k^{a,-} + \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^{a,-}}, \hat{\boldsymbol{\theta}}_k^{a,-} - \gamma\sqrt{\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^{a,-}}\Big\}$$
$$\boldsymbol{\mathcal{D}}_{k,i} = \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\mathcal{W}}_{k,i}^{\theta,-}, \boldsymbol{\mathcal{W}}_{k,i}^{e,-})$$
$$\hat{\boldsymbol{d}}_k = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{m})} \boldsymbol{\mathcal{D}}_{k,i}$$

Estimator gain matrix:
$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}},k} = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big) \times \big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big)^T$$
$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}}\tilde{\boldsymbol{d}},k}^- = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{W}}_{k,i}^{\theta,-} - \hat{\boldsymbol{\theta}}_k^-\big) \times \big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big)^T$$
$$\boldsymbol{L}_k^\theta = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}}\tilde{\boldsymbol{d}},k}^- \boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}},k}^{-1}$$

Parameter-estimate meas. update: $\hat{\boldsymbol{\theta}}_k^+ = \hat{\boldsymbol{\theta}}_k^- + \boldsymbol{L}_k^\theta(\boldsymbol{d}_k - \hat{\boldsymbol{d}}_k)$

Error-covariance meas. update: $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^+ = \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- - \boldsymbol{L}_k^\theta \boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}},k} (\boldsymbol{L}_k^\theta)^T$

12:07:34.

Appendix: Joint EKF for state and parameter estimation

State-space model:

$$\begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} f(x_{k-1}, u_{k-1}, w_{k-1}, \theta_{k-1}) \\ \theta_{k-1} + r_{k-1} \end{bmatrix}$$

$$y_k = h(x_k, u_k, v_k, \theta_k)$$

or

$$\mathbf{X}_k = \mathbf{F}(\mathbf{X}_{k-1}, u_{k-1}, \mathbf{W}_{k-1})$$
$$y_k = h(\mathbf{X}_k, u_k, v_k),$$

where w_k, r_k, and v_k are independent, Gaussian noise processes having means $\bar{w}$, zero, and $\bar{v}$, and covariance matrices $\Sigma_{\widetilde{w}}$, $\Sigma_{\widetilde{r}}$, and $\Sigma_{\widetilde{v}}$, respectively. For brevity, we let $\mathbf{X}_k = \begin{bmatrix} x_k^T, & \theta_k^T \end{bmatrix}^T$, $\mathbf{W}_k = \begin{bmatrix} w_k^T, & r_k^T \end{bmatrix}^T$ and $\Sigma_{\widetilde{\mathbf{W}}} = \mathrm{diag}(\Sigma_{\widetilde{w}}, \Sigma_{\widetilde{r}})$.

Definitions:

$$\hat{A}_k = \frac{\mathrm{d}\mathbf{F}(\mathbf{X}_k, u_k, \mathbf{W}_k)}{\mathrm{d}\mathbf{X}_k}\bigg|_{\mathbf{X}_k = \hat{\mathbf{X}}_k^+} \qquad \hat{B}_k = \frac{\mathrm{d}\mathbf{F}(\mathbf{X}_k, u_k, \mathbf{W}_k)}{\mathrm{d}\mathbf{W}_k}\bigg|_{\mathbf{W}_k = \bar{\mathbf{W}}_k}$$

$$\hat{C}_k = \frac{\mathrm{d}h(\mathbf{X}_k, u_k, v_k)}{\mathrm{d}\mathbf{X}_k}\bigg|_{\mathbf{X}_k = \hat{\mathbf{X}}_k^-} \qquad \hat{D}_k = \frac{\mathrm{d}h(\mathbf{X}_k, u_k, v_k)}{\mathrm{d}v_k}\bigg|_{v_k = \bar{v}_k}.$$

Initialization: For $k = 0$, set

$$\hat{\mathbf{X}}_0^+ = \mathbb{E}[\mathbf{X}_0]$$
$$\Sigma_{\widetilde{\mathbf{X}},0}^+ = \mathbb{E}[(\mathbf{X}_0 - \hat{\mathbf{X}}_0^+)(\mathbf{X}_0 - \hat{\mathbf{X}}_0^+)^T]$$

Computation: For $k = 1, 2, \ldots$ compute:

State estimate time update: $\hat{\mathbf{X}}_k^- = \mathbf{F}(\hat{\mathbf{X}}_{k-1}^+, u_{k-1}, \bar{\mathbf{W}}_{k-1})$

Error covariance time update: $\Sigma_{\widetilde{\mathbf{X}},k}^- = \hat{A}_{k-1}\Sigma_{\widetilde{\mathbf{X}},k-1}^+\hat{A}_{k-1}^T + \hat{B}_{k-1}\Sigma_{\widetilde{\mathbf{W}}}\hat{B}_{k-1}^T$

Output estimate: $\hat{y}_k = h(\hat{\mathbf{X}}_k^-, u_k, \bar{v}_k)$

Estimator gain matrix: $L_k = \Sigma_{\widetilde{\mathbf{X}},k}^-\hat{C}_k^T[\hat{C}_k\Sigma_{\widetilde{\mathbf{X}},k}^-\hat{C}_k^T + \hat{D}_k\Sigma_{\widetilde{v}}\hat{D}_k^T]^{-1}$

State estimate meas. update: $\hat{\mathbf{X}}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$

Error covariance meas. update: $\Sigma_{\widetilde{\mathbf{X}},k}^+ = \Sigma_{\widetilde{\mathbf{X}},k}^- - L_k\Sigma_{\widetilde{y},k}L_k^T$

Appendix: Dual EKF for state and parameter estimation

Nonlinear state-space models:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}_k, \boldsymbol{w}_k) \\ \boldsymbol{y}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}_k, \boldsymbol{v}_k) \end{aligned} \quad \text{and} \quad \begin{aligned} \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \boldsymbol{r}_k, \\ \boldsymbol{d}_k &= \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\theta}_k, \boldsymbol{e}_k). \end{aligned}$$

where $\boldsymbol{w}_k$, $\boldsymbol{v}_k$, $\boldsymbol{r}_k$ and $\boldsymbol{e}_k$ are independent Gaussian noise processes with means $\bar{\boldsymbol{w}}$, $\bar{\boldsymbol{v}}$, zero, and $\bar{\boldsymbol{e}}$ and covariance matrices $\boldsymbol{\Sigma}_{\widetilde{w}}$, $\boldsymbol{\Sigma}_{\tilde{v}}$, $\boldsymbol{\Sigma}_{\tilde{r}}$ and $\boldsymbol{\Sigma}_{\tilde{e}}$, respectively.

Definitions:

$$\hat{\boldsymbol{A}}_k = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_k,\boldsymbol{u}_k,\hat{\boldsymbol{\theta}}_k^-,\boldsymbol{w}_k)}{\mathrm{d}\boldsymbol{x}_k}\bigg|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_k^+} \qquad \hat{\boldsymbol{B}}_k = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x}_k,\boldsymbol{u}_k,\hat{\boldsymbol{\theta}}_k^-,\boldsymbol{w}_k)}{\mathrm{d}\boldsymbol{w}_k}\bigg|_{\boldsymbol{w}_k=\bar{\boldsymbol{w}}}$$

$$\hat{\boldsymbol{C}}_k^x = \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k,\boldsymbol{u}_k,\hat{\boldsymbol{\theta}}_k^-,\boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{x}_k}\bigg|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_k^-} \qquad \hat{\boldsymbol{D}}_k^x = \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}_k,\boldsymbol{u}_k,\hat{\boldsymbol{\theta}}_k^-,\boldsymbol{v}_k)}{\mathrm{d}\boldsymbol{v}_k}\bigg|_{\boldsymbol{v}_k=\bar{\boldsymbol{v}}}$$

$$\hat{\boldsymbol{C}}_k^\theta = \frac{\mathrm{d}\boldsymbol{g}(\hat{\boldsymbol{x}}_k^-,\boldsymbol{u}_k,\boldsymbol{\theta},\boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_k^-} \qquad \hat{\boldsymbol{D}}_k^\theta = \frac{\mathrm{d}\boldsymbol{g}(\hat{\boldsymbol{x}}_k^-,\boldsymbol{u}_k,\boldsymbol{\theta},\boldsymbol{e}_k)}{\mathrm{d}\boldsymbol{e}_k}\bigg|_{\boldsymbol{e}_k=\bar{\boldsymbol{e}}}.$$

Initialization: For $k = 0$, set

$$\begin{aligned} \hat{\boldsymbol{\theta}}_0^+ &= \mathbb{E}[\boldsymbol{\theta}_0], & \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},0}^+ &= \mathbb{E}\big[(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\theta}}_0^+)^T\big], \\ \hat{\boldsymbol{x}}_0^+ &= \mathbb{E}[\boldsymbol{x}_0], & \boldsymbol{\Sigma}_{\tilde{x},0}^+ &= \mathbb{E}\big[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)^T\big]. \end{aligned}$$

Computation: For $k = 1, 2, \ldots$ compute:

Param.-pred. time update:
$$\begin{aligned} \hat{\boldsymbol{\theta}}_k^- &= \hat{\boldsymbol{\theta}}_{k-1}^+ \\ \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- &= \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k-1}^+ + \boldsymbol{\Sigma}_{\tilde{r}} \end{aligned}$$

State-pred. time update:
$$\begin{aligned} \hat{\boldsymbol{x}}_k^- &= \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}^+, \boldsymbol{u}_{k-1}, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{w}}) \\ \boldsymbol{\Sigma}_{\tilde{x},k}^- &= \hat{\boldsymbol{A}}_{k-1}\boldsymbol{\Sigma}_{\tilde{x},k-1}^+\hat{\boldsymbol{A}}_{k-1}^T \\ &\quad + \hat{\boldsymbol{B}}_{k-1}\boldsymbol{\Sigma}_{\widetilde{w}}\hat{\boldsymbol{B}}_{k-1}^T \end{aligned}$$

State filter meas. update:
$$\begin{aligned} \boldsymbol{L}_k^x &= \boldsymbol{\Sigma}_{\tilde{x},k}^-(\hat{\boldsymbol{C}}_k^x)^T\big[\hat{\boldsymbol{C}}_k^x\boldsymbol{\Sigma}_{\tilde{x},k}^-(\hat{\boldsymbol{C}}_k^x)^T + \\ &\qquad \hat{\boldsymbol{D}}_k^x\boldsymbol{\Sigma}_{\tilde{v}}(\hat{\boldsymbol{D}}_k^x)^T\big]^{-1} \\ \hat{\boldsymbol{x}}_k^+ &= \hat{\boldsymbol{x}}_k^- + \boldsymbol{L}_k^x\big[\boldsymbol{y}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{v}})\big] \\ \boldsymbol{\Sigma}_{\tilde{x},k}^+ &= \boldsymbol{\Sigma}_{\tilde{x},k}^- - \boldsymbol{L}_k^x\boldsymbol{\Sigma}_{\tilde{y},k}(\boldsymbol{L}_k^x)^T \end{aligned}$$

Param.-est. meas. update:
$$\begin{aligned} \boldsymbol{L}_k^\theta &= \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^-(\hat{\boldsymbol{C}}_k^\theta)^T\big[\hat{\boldsymbol{C}}_k^\theta\boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^-(\hat{\boldsymbol{C}}_k^\theta)^T + \\ &\qquad \hat{\boldsymbol{D}}_k^\theta\boldsymbol{\Sigma}_{\tilde{e}}(\hat{\boldsymbol{D}}_k^\theta)^T\big]^{-1} \\ \hat{\boldsymbol{\theta}}_k^+ &= \hat{\boldsymbol{\theta}}_k^- + \boldsymbol{L}_k^\theta\big[\boldsymbol{y}_k - \boldsymbol{g}(\hat{\boldsymbol{x}}_k^-, \boldsymbol{u}_k, \hat{\boldsymbol{\theta}}_k^-, \bar{\boldsymbol{e}})\big] \\ \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^+ &= \boldsymbol{\Sigma}_{\tilde{\boldsymbol{\theta}},k}^- - \boldsymbol{L}_k^\theta\boldsymbol{\Sigma}_{\tilde{d},k}(\boldsymbol{L}_k^\theta)^T \end{aligned}$$

12:07:34.

Appendix: Joint SPKF for state and parameter estimation

State-space model:

$$\begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} f(x_{k-1}, u_{k-1}, w_{k-1}, \theta_{k-1}) \\ \theta_{k-1} + r_{k-1} \end{bmatrix}$$

$$y_k = h(x_k, u_k, v_k, \theta_k)$$

or

$$\mathbf{X}_k = \mathbf{F}(\mathbf{X}_{k-1}, u_{k-1}, \mathbf{W}_{k-1})$$

$$y_k = h(\mathbf{X}_k, u_k, v_k),$$

where w_k, r_k, and v_k are independent, Gaussian noise processes with means $\bar{w}$, zero, and $\bar{v}$, and covariance matrices $\Sigma_{\widetilde{w}}$, $\Sigma_{\tilde{r}}$, and $\Sigma_{\tilde{v}}$, respectively. For brevity, we let $\mathbf{X}_k = \left[x_k^T,\ \theta_k^T\right]^T$, $\mathbf{W}_k = \left[w_k^T,\ r_k^T\right]^T$, and $\Sigma_{\widetilde{\mathbf{W}}} = \operatorname{diag}(\Sigma_{\widetilde{w}}, \Sigma_{\tilde{r}})$.

Definitions: Let

$$\mathbf{X}_k^a = \left[\mathbf{X}_k^T,\ \mathbf{W}_k^T,\ v_k^T\right]^T, \quad \mathcal{X}_k^a = \left[(\mathcal{X}_k^{\mathbf{X}})^T,\ (\mathcal{X}_k^{\mathbf{W}})^T,\ (\mathcal{X}_k^{v})^T\right]^T,$$

$$p = 2 \times \dim(\mathbf{X}_k^a).$$

Initialization: For $k = 0$, set

$$\hat{\mathbf{X}}_0^+ = \mathbb{E}[\mathbf{X}_0] \qquad \hat{\mathbf{X}}_0^{a,+} = \mathbb{E}[\mathbf{X}_0^a] = \left[(\hat{\mathbf{X}}_0^+)^T,\ \bar{\mathbf{W}},\ \bar{v}\right]^T$$

$$\Sigma_{\tilde{\mathbf{X}},0}^+ = \mathbb{E}\left[(\mathbf{X}_0 - \hat{\mathbf{X}}_0^+)(\mathbf{X}_0 - \hat{\mathbf{X}}_0^+)^T\right] \qquad \Sigma_{\tilde{\mathbf{X}},0}^{a,+} = \mathbb{E}\left[(\mathbf{X}_0^a - \hat{\mathbf{X}}_0^{a,+})(\mathbf{X}_0^a - \hat{\mathbf{X}}_0^{a,+})^T\right]$$

$$= \operatorname{diag}\left(\Sigma_{\tilde{\mathbf{X}},0}^+, \Sigma_{\widetilde{\mathbf{W}}}, \Sigma_{\tilde{v}}\right).$$

Computation: For $k = 1, 2, \ldots$ compute:

State pred. time update:
$$\mathcal{X}_{k-1}^{a,+} = \left\{\hat{\mathbf{X}}_{k-1}^{a,+}, \hat{\mathbf{X}}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\tilde{\mathbf{X}},k-1}^{a,+}}, \hat{\mathbf{X}}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\tilde{\mathbf{X}},k-1}^{a,+}}\right\}$$

$$\mathcal{X}_{k,i}^{\mathbf{X},-} = \mathbf{F}(\mathcal{X}_{k-1,i}^{\mathbf{X},+}, u_{k-1}, \mathcal{X}_{k-1,i}^{\mathbf{W},+})$$

$$\hat{\mathbf{X}}_k^- = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{m})} \mathcal{X}_{k,i}^{\mathbf{X},-}$$

Error covar. time update:
$$\Sigma_{\tilde{\mathbf{X}},k}^- = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{c})} \left(\mathcal{X}_{k,i}^{\mathbf{X},-} - \hat{\mathbf{X}}_k^-\right)\left(\mathcal{X}_{k,i}^{\mathbf{X},-} - \hat{\mathbf{X}}_k^-\right)^T$$

Output estimate:
$$\mathcal{Y}_{k,i} = h(\mathcal{X}_{k,i}^{\mathbf{X},-}, u_k, \mathcal{X}_{k-1,i}^{v,+})$$

$$\hat{y}_k = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{m})} \mathcal{Y}_{k,i}$$

Estimator gain matrix:
$$\Sigma_{\tilde{y},k} = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{c})} \left(\mathcal{Y}_{k,i} - \hat{y}_k\right)\left(\mathcal{Y}_{k,i} - \hat{y}_k\right)^T$$

$$\Sigma_{\tilde{\mathbf{X}}\tilde{y},k}^- = \textstyle\sum_{i=0}^p \alpha_i^{(\mathrm{c})} \left(\mathcal{X}_{k,i}^{\mathbf{X},-} - \hat{\mathbf{X}}_k^-\right)\left(\mathcal{Y}_{k,i} - \hat{y}_k\right)^T$$

$$L_k = \Sigma_{\tilde{\mathbf{X}}\tilde{y},k}^- \Sigma_{\tilde{y},k}^{-1}$$

State est. meas. update:
$$\hat{\mathbf{X}}_k^+ = \hat{\mathbf{X}}_k^- + L_k\left(y_k - \hat{y}_k\right)$$

Error covar. meas. update:
$$\Sigma_{\tilde{\mathbf{X}},k}^+ = \Sigma_{\tilde{\mathbf{X}},k}^- - L_k \Sigma_{\tilde{y},k} L_k^T$$

Appendix: Dual SPKF for state and parameter estimation

Nonlinear state-space models:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}, \theta_{k-1})$$
$$y_k = h(x_k, u_k, v_k, \theta_k)$$

and

$$\theta_k = \theta_{k-1} + r_{k-1},$$
$$d_k = h(f(x_{k-1}, u_{k-1}, \bar{w}_{k-1}, \theta_{k-1}), u_k, \bar{v}_k, \theta_{k-1}, e_k).$$

where w_k, v_k, r_k and e_k are independent, Gaussian noise processes with means $\bar{w}$, $\bar{v}$, zero, and $\bar{e}$, and covariance matrices $\Sigma_{\tilde{w}}$, $\Sigma_{\tilde{v}}$, $\Sigma_{\tilde{r}}$ and $\Sigma_{\tilde{e}}$, respectively.

Definitions:

$$x_k^a = \left[x_k^T,\ w_k^T,\ v_k^T\right]^T, \quad \mathcal{X}_k^a = \left[(\mathcal{X}_k^x)^T,\ (\mathcal{X}_k^w)^T,\ (\mathcal{X}_k^v)^T\right]^T,$$
$$p = 2 \times \dim(x_k^a).$$

Initialization: For $k = 0$, set

$$\hat{\theta}_0^+ = \mathbb{E}[\theta_0], \qquad \Sigma_{\tilde{\theta},0}^+ = \mathbb{E}\left[(\theta_0 - \hat{\theta}_0^+)(\theta_0 - \hat{\theta}_0^+)^T\right]$$
$$\hat{x}_0^+ = \mathbb{E}[x_0], \qquad \hat{x}_0^{a,+} = \mathbb{E}\left[x_0^a\right] = \left[(\hat{x}_0^+)^T,\ \bar{w},\ \bar{v}\right]^T$$
$$\Sigma_{\tilde{x},0}^+ = \mathbb{E}\left[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\right] \qquad \Sigma_{\tilde{x},0}^{a,+} = \mathbb{E}\left[(x_0^a - \hat{x}_0^{a,+})(x_0^a - \hat{x}_0^{a,+})^T\right]$$
$$= \operatorname{diag}\left(\Sigma_{\tilde{x},0}^+, \Sigma_{\tilde{w}}, \Sigma_{\tilde{v}}\right).$$

Computation: For $k = 1, 2, \ldots$ compute:

Param.-pred. time update: $\hat{\theta}_k^- = \hat{\theta}_{k-1}^+$

Param.-covar. time update: $\Sigma_{\tilde{\theta},k}^- = \Sigma_{\tilde{\theta},k-1}^+ + \Sigma_{\tilde{r}}$

State-estimate time update:
$$\mathcal{X}_{k-1}^{a,+} = \left\{\hat{x}_{k-1}^{a,+}, \hat{x}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}}, \hat{x}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}}\right\}$$
$$\mathcal{X}_{k,i}^{x,-} = f(\mathcal{X}_{k-1,i}^{x,+}, u_{k-1}, \mathcal{X}_{k-1,i}^{w,+}, \hat{\theta}_k^-)$$
$$\hat{x}_k^- = \textstyle\sum_{i=0}^p \alpha_i^{(m)} \mathcal{X}_{k,i}^{x,-}$$

State-covar. time update:
$$\Sigma_{\tilde{x},k}^- = \textstyle\sum_{i=0}^p \alpha_i^{(c)} \left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-\right)\left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-\right)^T$$

Output est., param. filter:
$$\mathcal{W}_k = \left\{\hat{\theta}_k^-, \hat{\theta}_k^- + \gamma\sqrt{\Sigma_{\tilde{\theta},k}^-}, \hat{\theta}_k^- - \gamma\sqrt{\Sigma_{\tilde{\theta},k}^-}\right\}$$
$$\mathcal{D}_{k,i} = h(f(\hat{x}_{k-1}^+, u_{k-1}, \bar{w}_{k-1}, \mathcal{W}_{k,i}), u_k, \bar{v}_k, \mathcal{W}_{k,i})$$
$$\hat{d}_k = \textstyle\sum_{i=0}^p \alpha_i^{(m)} \mathcal{D}_{k,i}$$

Output est., state filter:
$$\mathcal{Y}_{k,i} = h(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+}, \hat{\theta}_k^-)$$
$$\hat{y}_k = \textstyle\sum_{i=0}^p \alpha_i^{(m)} \mathcal{Y}_{k,i}$$

(continued on next page...)

Computation (continued from prior page): For $k = 1, 2, \ldots$ compute:

State filter gain matrix:
$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}},k} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)\big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)^T$$
$$\boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{z}},k} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{X}}^{x,-}_{k,i} - \hat{\boldsymbol{x}}_k^{-}\big)\big(\boldsymbol{\mathcal{Y}}_{k,i} - \hat{\boldsymbol{y}}_k\big)^T$$
$$\boldsymbol{L}_k^{x} = \boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{y}},k}\boldsymbol{\Sigma}^{-1}_{\tilde{\boldsymbol{y}},k}$$

Param. filter gain matrix:
$$\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}},k} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big)\big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big)^T$$
$$\boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{\theta}}\tilde{\boldsymbol{d}},k} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \big(\boldsymbol{\mathcal{W}}_{k,i} - \hat{\boldsymbol{\theta}}_k^{-}\big)\big(\boldsymbol{\mathcal{D}}_{k,i} - \hat{\boldsymbol{d}}_k\big)^T$$
$$\boldsymbol{L}_k^{\theta} = \boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{\theta}}\tilde{\boldsymbol{d}},k}\boldsymbol{\Sigma}^{-1}_{\tilde{\boldsymbol{d}},k}$$

State-est. meas. update: $\hat{\boldsymbol{x}}_k^{+} = \hat{\boldsymbol{x}}_k^{-} + \boldsymbol{L}_k^{x}\big(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\big)$

State-covar. meas. update: $\boldsymbol{\Sigma}^{+}_{\tilde{\boldsymbol{x}},k} = \boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{x}},k} - \boldsymbol{L}_k^{x}\boldsymbol{\Sigma}_{\tilde{\boldsymbol{y}},k}(\boldsymbol{L}_k^{x})^T$

Param.-est. meas. update: $\hat{\boldsymbol{\theta}}_k^{+} = \hat{\boldsymbol{\theta}}_k^{-} + \boldsymbol{L}_k^{\theta}\big(\boldsymbol{y}_k - \hat{\boldsymbol{d}}_k\big)$

Param.-covar. meas. update: $\boldsymbol{\Sigma}^{+}_{\tilde{\boldsymbol{\theta}},k} = \boldsymbol{\Sigma}^{-}_{\tilde{\boldsymbol{\theta}},k} - \boldsymbol{L}_k^{\theta}\boldsymbol{\Sigma}_{\tilde{\boldsymbol{d}},k}(\boldsymbol{L}_k^{\theta})^T$

12:07:34.

5
Cell Balancing

Referring back to the roadmap Fig. 3.1 for the principal topics covered in this book, we recognize that we have now explored the basic *estimation* tasks performed by a BMS. That is, we have seen how to estimate cell internal state (including SOC), and cell-model parameter values (including those indicative of SOH).

We now begin to study the *control* tasks required by a BMS. This chapter focuses on the battery-pack task known by either of the terms *balancing* or *equalization*, which has to do with modifying the level of charge in battery-pack cells on a cell by cell basis to bring the battery pack into balance.

For the time being, we will adopt the working definition, "A balanced battery pack is one in which, at some point in its cycle, all the cells are at exactly the same SOC."[1] There are other ways of thinking about what the distribution of SOCs should be in a balanced battery pack at any point of time, but this definition will be sufficient for now.

[1] Andrea, D., *Battery Management Systems for Large Lithium-Ion Battery Packs*, Artech House, 2010, p. 23.

If this balancing condition is not met, then one or more cells has SOC that is too high and one or more cells has SOC that is too low with respect to the balancing condition. We must somehow modify the SOCs of specific cells individually to bring the battery pack into balance. There are two basic approaches to doing so.

Dissipative balancing works by draining charge from cells having too much charge with respect to the balancing condition and dissipates the drained energy as heat. This type of balancing is very commonly called *passive balancing* because, historically, only passive circuit elements were used to equalize the cells. However, contemporary dissipative balancing circuits use active components and controls (i.e., they use transistor switching circuits), so we prefer the term *dissipative balancing* to *passive balancing*.

Nondissipative balancing works by moving charge from cells having too much charge either to cells having too little charge or to an auxiliary load circuit. While dissipative balancing wastes the unusable electrochemical energy stored in some cells by converting it to heat, nondissipative balancing attempts to conserve energy by redistributing charge among cells so that more energy is available to the load, or by directly putting it to some immediately useful purpose. This type of balancing is commonly known as *active balancing*; however, as we've just noted, dissipative balancing systems can use active components. For this reason, to minimize the possibility of confusion, we prefer the term *nondissipative balancing* to *active balancing*.

We will investigate the design of balancing methods and look at balancing circuits later, but first we discuss why balancing is important. Consider again the trivial battery pack drawn in Fig. 5.1. One cell has 0 % SOC, and the other has 100 % SOC. Because current passing through the battery pack to or from the load circuit will cause SOCs for both of these cells to move in the same direction, we cannot discharge the pack without overdischarging one cell; nor can we charge the pack without overcharging the other. We need somehow to augment the main power pathway of the battery pack with auxiliary pathways that allow for a degree of individual cell-level control to bring these cells back into balance.

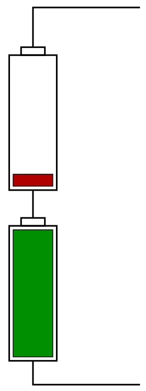


Figure 5.1: An out-of-balance series-connected battery pack (duplicated from Figs. 1.20 and 3.31).

We have seen this figure before as motivation for the need to estimate the individual SOCs of all cells in a battery pack, without considering at the time how the cells could have become imbalanced in the first place. In this chapter, we will look at some causes for imbalance and some additional factors that one might initially think would lead to imbalance but actually do not. Then, we discuss some engineering choices that need to be made when designing a balancing system; we examine a number of circuit topologies that can be used to balance cells; and we look at a method for determining how quickly the balancing system must operate to equalize charge.

5.1 *Causes of imbalance*

Fig. 5.2 sketches profiles of SOC versus time for an example scenario where a two-cell battery pack is being repeatedly discharged and charged. The scenario begins with both cells balanced at a top design SOC. However, by the time that the pack has been discharged until the lesser of the two SOCs reaches a lower design limit, we see that the SOCs of the two cells have begun to diverge. When the pack is subsequently charged, this divergence is not corrected, and over time the imbalance grows.

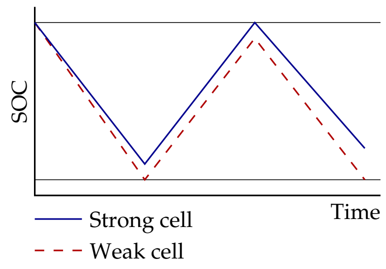


Figure 5.2: Battery-pack imbalance growing over time.

Whenever the lower SOC design limit is reached, in this example, the cell depicted by the dashed red line has no more charge available to deliver to the load circuit but the cell depicted by the solid blue line still holds charge. Therefore, we refer to the first cell as being

12:07:34.

"weak" and to the second cell as being "strong" in the sense that a weak cell limits battery-pack performance. Since the strong cell cannot actually deliver its charge to the load without overdischarging the weak cell, there is energy stored in the battery pack that is unavailable for use. Over time, the imbalance will grow and the weak cell will ultimately render the pack useless unless cells are balanced.

Imbalance is caused by anything that can make one cell's SOC diverge from another's. To see what these causes might be, we consider the SOC relationship:

$$z_k = z_0 - \frac{\Delta t}{Q} \sum_{i=0}^{k-1} \eta_i i_{\text{net},i}. \qquad (5.1)$$

One cause of imbalance is when cells have different coulombic efficiencies. Cells may start with the same initial SOC z_0, have the same total capacity Q, and receive the same net current $i_{\text{net},i}$. However, if they have different coulombic efficiencies η_i, then their SOCs will diverge while charging the battery pack. This is illustrated in Fig. 5.3. The strong cell has high coulombic efficiency and so the majority of the charge current $i_{\text{net},i}$ is converted to a change in cell SOC. The weak cell has low coulombic efficiency and so a smaller fraction of the charge current converts to a change in the cell's SOC. During discharge, coulombic efficiency is assumed to be perfect for all cells, and so the divergence caused during charging will not be neutralized. Therefore, during every charge cycle, a difference in coulombic efficiency will cause increased divergence that is not corrected later during a discharge cycle. Over time, the battery pack can arrive at the extreme state depicted in Fig. 5.1.

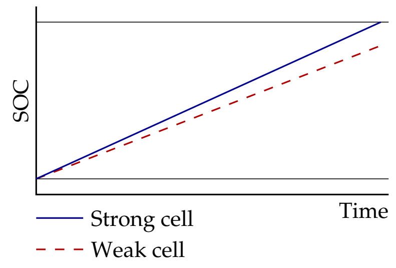


Figure 5.3: Imbalance growing during charge.

Another cause of imbalance is when cells experience different net current from one other. That is, we need to consider carefully the constituent parts of net current for cell i:

$$i_{\text{net},i} = i_{\text{app}} + i_{\text{self-discharge},i} + i_{\text{leakage},i},$$

where i_{app} is the battery-pack load current, $i_{\text{self-discharge},i}$ is the rate of self-discharge for cell i, and $i_{\text{leakage},i}$ is the current drawn from cell i that powers the attached BMS electronic circuitry. While i_{app} is the same for all cells, the self-discharge rates and leakage currents of individual cells can be different, leading to different $i_{\text{net},i}$. This is illustrated during a discharge profile in Fig. 5.4. The strong cell has low self-discharge and/or low leakage current. The weak cell has higher self-discharge and/or leakage current. While the two cells start the profile in a balanced state, the larger net discharge current experienced by the weak cell causes its SOC to decrease more rapidly than that of the strong cell. When the pack is subsequently charged,

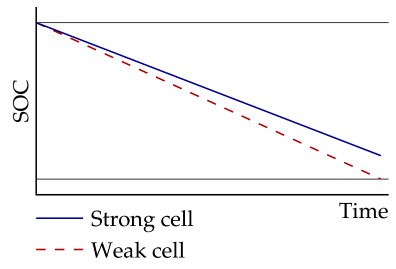


Figure 5.4: Imbalance growing during discharge.

the strong cell will charge more quickly than the weak cell because its net current is not drawn down as much by self discharge and leakage, and so the imbalance will continue to grow. Over time, the cells will diverge completely.

Temperature is not a direct cause of imbalance; but, a gradient of temperature across a battery pack can be a contributing factor to imbalance. Cell parameter values are temperature-dependent, so self-discharge rates, electronics' performance, and coulombic efficiency will be different for cells having different internal temperatures. This will cause imbalance. Also, high temperatures tend to accelerate degradation, and so a long-term temperature gradient will lead different rates of degradation in different cells in the battery pack, which will lead to to accelerated imbalance because of the resulting different self-discharge rates and coulombic efficiencies among cells. Maintaining uniform temperatures across the battery pack will help to prolong the battery-pack life but, in general, balancing will still be needed.

In summary, we stress that it is *differences* in the coulombic efficiencies, self-discharge rates, or leakage currents among the cells of a battery pack that lead to imbalance, not the absolute quantities themselves. If all cells are equally strong or equally weak, there will be no growth in imbalance as the pack operates.

5.2 *Not causes of imbalance*

Considering the SOC Eq. (5.1), we see that cell total capacity Q plays a role. So, it is natural to assume that cells in a battery pack having different total capacities will cause the pack to become progressively imbalanced. It turns out that this is not the case; instead, different cell total capacities cause only a temporary imbalance that is corrected automatically when any cell returns to its original SOC.[2]

[2] The following argument assumes that cell capacities do not change appreciably in a singe cycle. If one or more cells in a battery pack experience rapid capacity loss between the discharge and charge portion of a cycle (due, perhaps, to overdischarge), then the pack will become imbalanced. However, this will not happen if the BMS is working as intended.

This is illustrated in Fig. 5.5. Suppose that the strong cell has a total capacity of 6 Ah and that the weak cell has a total capacity of 5 Ah. The scenario begins with both cells having equal SOC of 100 %. Then, suppose that 5 Ah is discharged from the battery pack. The weak cell now has zero remaining available charge but the strong cell has residual capacity of 1 Ah. We might suppose that the the pack has become imbalanced, but we need to continue the example before we can make that conclusion.

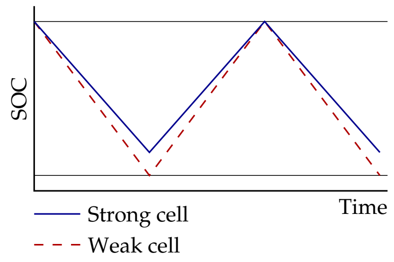


Figure 5.5: Difference in total capacities does not cause long-term growth in imbalance.

We now charge the battery pack, adding $5/\eta$ Ah, where we assume that the coulombic efficiencies η of both cells are equal. Then, the 5 Ah cell will contain 5 Ah of charge and the 6 Ah cell will contain 6 Ah of charge. That is, both cells will be balanced with SOC equal to 100 %. Recalling our working definition of a balanced battery

12:07:34.

pack from footnote 1, we see that this battery pack is balanced—even though cell total capacities are different—because all cells have equal SOC at one point in the discharge/charge cycle (i.e., when all cells have 100 % SOC, in this example).

But, while we can claim that the pack is balanced, technically, we still observe that the weak cell limits its performance. When the weak cell is fully discharged, there is still 1 Ah of unused charge stored in the strong cell that is not available to power the load circuit without over-discharging the weak cell. If we were able to move charge continuously from the strong cell into the weak cell while discharging (and vice versa while charging), we could keep both cells' SOCs equal at all points in time. Nondissipative balancing circuits can be used to do this: if these circuits operate with perfect efficiency, then we could draw 5.5 Ah from the example battery pack instead of only 5 Ah without balancing circuitry, which is a significant improvement. The weak cell no longer limits the battery-pack performance; rather, it is the average over all cells that does so.

We might imagine that cells having different resistance could also cause battery-pack imbalance. However, examining Eq. (5.1), we see that this is not the case, because resistance is not a factor in the SOC equation. Different cell resistances will cause cell loaded terminal voltages to be different, but not their SOCs.

This does mean that a cell with high resistance will tend to reach an upper or lower design voltage limit before other cells in the battery pack. While this does not cause imbalance, it does limit the available battery-pack power. This also limits the *available energy* which translates directly to a limited vehicle range in xEV application.[3]

[3] *Available energy* is the amount of energy that can be removed from the battery pack before a discharge power limitation is reached. This is different from *total energy*, which was described in Sect. 1.14. Available energy is always less than total energy unless the discharge power specification is 0 W.

Therefore, just as different total capacities did not cause imbalance but did limit performance, so too a cell having higher resistance than others in the pack will limit performance. It is possible to mitigate this limitation by using nondissipative balancing circuitry to intentionally bring the battery pack to an out-of-balance condition to equalize the power that can be sourced/sunk from cells in the pack. That is, when the battery pack is being discharged, high-resistance cells are kept at higher SOC than low-resistance cells so that the discharge power that can be sunk by all cells is equal. To do so, we solve Eq. (1.7) for a desired set of cell SOCs to enforce equal discharge powers despite unequal discharge resistances. Similarly, when the pack is being charged, high-resistance cells are kept at lower SOCs than low-resistance cells so that the charge power that is available from all cells is equal. We solve Eq. (1.8) for a desired set of cell SOCs to enforce equal charge powers despite unequal charge resistances.

5.3 *Balancer design choices*

When designing a balancing system, certain engineering decisions must be made. The following subsections discuss the design questions that must be addressed but are not prescriptive. The answers to these questions will tend to be different for distinct application scenarios, leading to different designs.

5.3.1 *What balance setpoint?*

According to our working definition, all cells in a balanced battery pack must be at the same SOC at some point in the battery-pack cycle. We have seen an exception to this, where (ironically) fast nondissipative balancing circuits are used when cell total capacities and resistances differ to cause intentional imbalance to improve battery-pack available power and energy. However, disregarding this special case, we need to consider what that balance-point SOC should be.

Several different alternatives are illustrated in Fig. 5.6. The top frame considers a balance setpoint at the maximum permitted cell SOC. The example shows a four-cell battery pack, where the "strong" cell has highest total capacity and the "weak" cell has lowest total capacity. While the differences in capacity are not the cause of imbalance, they do lead to different SOC ranges being utilized by each cell as the battery pack is discharged and charged, as illustrated by the extent of the corresponding shaded boxes.

The weak cell, having lowest total capacity, uses its entire SOC range as illustrated by the magenta shaded box. The strong cell, having highest total capacity, uses the smallest SOC range as illustrated by the blue shaded box. The other two cells use SOC ranges inbetween these extremes, as illustrated by the yellow and green shaded boxes. Plotted versus time, the strong- and weak-cell SOCs would evolve as is shown in Fig. 5.5. The time element is removed in Fig. 5.6 to show only the range of SOC utilization.

When the balance setpoint is set to the maximum permitted SOC, the energy that is stored by the battery pack is maximized for a given amount of available ampere-hours. (This is because cells having higher SOCs produce higher voltages than cells having lower SOCs.) The higher level of energy is good for electric vehicles and similar applications. However, some cell aging mechanisms are accelerated at high SOCs, which could be a disadvantage to this approach because all cells spend time during a discharge/charge cycle at the maximum design SOC.

We could also set the balance point to an intermediate SOC, as is illustrated in the second frame of the figure. This decreases the total

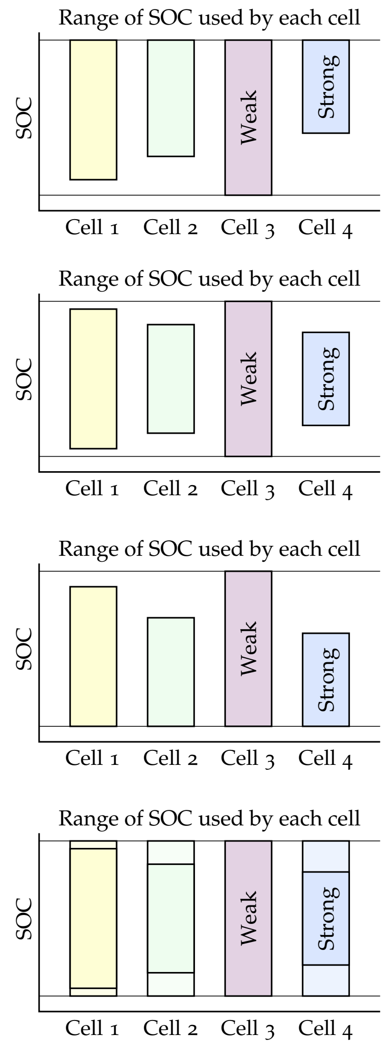


Figure 5.6: Several strategies for choosing a balancing setpoint.

available energy slightly, but it also maximizes a pack's ability to accept or deliver power, because most cells are kept away from extreme SOC operating points. This setpoint strategy is good for HEV and similar applications where available energy is not as important as available power.

We might also consider setting the balance point to the minimum permitted SOC, as is shown in the third frame of the figure. However, there is little to recommend doing so. It stores the least energy of the three approaches and all cells encounter discharge-power limits at roughly the same time (depending on their resistances), so that any cell could potentially limit discharge power when the pack is close to empty. Further, while aging based on high SOC is slowed down for the strongest cells because they spend less time at high SOCs, it is accelerated for the weakest cell because it does spend some time during a discharge/charge cycle at high SOC. This means that "the strong will stay strongest and the weak will get weaker." This kind of divergent positive feedback will tend to cause the battery pack to age much more quickly than it otherwise would, because battery-pack life is limited by the longevity of its weakest cell.

If nondissipative balancing were implemented, then the balance setpoint can be dynamic such that the entire range of every cell is used. This is illustrated in the bottom frame of the figure. The stronger cells experience greater total load than the weaker cells as they differentially contribute more power to the load circuit. The stronger cells then tend to age more quickly than the weaker cells due to the greater stress levels placed on them. This self-regulating negative feedback tends to cause the limiting weak cells in the battery pack to age less quickly than with dissipative balancing, and will tend to bring the battery-pack cells to a homogeneous end-of-life condition that is based on the average cell rather than the weakest cell.

Note again that the balance setpoint is a *design choice*. There isn't a one-size-fits-all answer. The controls engineer will need to consider the benefits and costs corresponding to each alternative for every new application.

5.3.2 When to balance?

Once we have determined a design balance setpoint, we must decide when we will consider "turning on" balancing circuitry during the battery usage cycle. There are three basic options.

If the battery pack is ever plugged in to the utilities grid or has some other special charging mode, then we might consider balancing only when the battery pack is being charged. This approach can be used for EVs, PHEVs, and E-REVs, for example. Since balancing is performed only when the battery pack is connected to an external power source, any energy dissipated by balancer circuitry can be replenished immediately from the external source, topping up the level of charge in the pack. This maximizes the energy stored by the battery pack that will be available for use when the pack is disconnected from the charger. In a vehicle application, this maximizes vehicle range.

Alternately, we might balance continuously. That is, at every point in time that the battery pack is being used, we calculate which cells need to be balanced in order to meet the overall balancing objective and we activate or deactivate the appropriate balancing circuits. This is necessary for applications such as hybrid-electric vehicles where the battery pack operates only in a single mode, and so there is no additional special mode (such as plug-in charging) during which we might consider balancing. Similarly, if we are using nondissipative balancing with dynamic SOC setpoints for each cell, we will need to recompute the moving setpoints and activate the appropriate balancing circuitry continuously.

Either one of these two approaches could use a myopic or a predictive perspective. Myopically, we could look only at the present state of the cells; predictively, we could project cell states into the future. A myopic approach to balancing while charging would be to charge the battery pack first, and then balance it once the first cell has hit an upper voltage or SOC limit. A predictive approach would instead project future states of the cells and balance while the battery pack is otherwise in use. Such an approach to balancing while charging would be to predict the end state of all cells when the charge is complete, and to balance and charge simultaneously so that all cells reach the same desired end state at the same time, even if it is not otherwise obvious based on the present cell states that balancing is needed. This can shorten charging times required by the battery pack.

5.3.3 *How to balance?*

By this point, we have decided on a balance setpoint and when, in general, we will consider balancing during a battery-pack usage cycle. It remains to determine how we will decide, in real time during a balancing interval, which balancing circuits to activate and how to know when to stop balancing.

Taking a myopic approach, we might choose to balance based on present cell SOC estimates. We compute estimates of SOC for all cells, then activate balancing circuits to remove charge from cells having SOC too high and possibly add charge to cells having SOC too low. This is generally what we wish to do, especially in the neighborhood of the balance setpoint. However, if cell SOC estimates are poor, we can sometimes balance the "wrong" cells. That is, noise in a SOC estimate might cause us to think that the cell's SOC is either much higher or much lower than it actually is, and we would activate balancing circuitry that actually causes greater imbalance. So, it is best to stop balancing when the difference between the maximum and minimum SOC estimates fall below some SOC dispersion threshold. This threshold is chosen relative to the expected confidence bounds on the SOC estimates in order to avoid wasting significant amounts of time and energy balancing the wrong cells.

Alternately, we could choose to balance based on cell voltage measurements and to continue balancing until the total dispersion in voltage measurements among all cells falls below some voltage design threshold. This method is simpler because a SOC estimator is not needed in an implementation, but is often wasteful since voltage is a poor indicator of cell SOC, and we will often balance the wrong cells. It is most useful in a "charge-first then balance" strategy where the balancing currents are the only currents experienced by the cells during the balancing period, and so the cells are often close to their equilibrium voltages. Since a SOC estimator is required for other BMS tasks, voltage-based balancing does not simplify overall BMS design and rarely makes sense.

In a more predictive sense, we could even balance based on total available energy computed for each cell.[4] To do so, we estimate how much energy could be removed from every cell in the pack at some specified constant-current or constant-power rate before a voltage cutoff limit is reached. Then, charge is removed from cells having excess available energy and possibly added to cells having lower available energy. This process continues until the difference between maximum and minimum available energy falls within some energy design threshold. This can improve total available energy in a nondissipative balancing system by moving charge from low-resistance cells to high-resistance cells to bolster their SOCs, and hence their voltages. However, it also needs an accurate cell model and state estimate for every cell to be done well.

[4] SOCs are never exactly equal at any point when balancing for this criterion.

5.4 *Circuits for balancing*

There are a variety of generic electronics architectures that may be used in a cell-balancing system.[5] A taxonomy of the most common topologies is drawn in Fig. 5.7, and we look at each in the following subsections. The simplest architectures do not allow any control by the BMS software, so are introduced as reference only and are not recommended for most applications. Others may be turned on

[5] For excellent overviews, see Moore, S.W. and Schneider, P.J. "A Review of Cell Equalization Methods for Lithium Ion and Lithium Polymer Battery Systems," No. 2001-01-0959, SAE Technical Paper, 2001 and Daowd, M., Omar, N., Van Den Bossche, P., Van Mierlo, J., "Passive and Active Battery Balancing Comparison based on MATLAB Simulation," *IEEE Vehicle Power and Propulsion Conference (VPPC)*, 2011.

or off globally but do not allow individual cell-level control. They operate autonomously to balance the voltages of all cells. As mentioned in the prior section, this is not the ideal scenario but might be acceptable if performed at the end of a battery-pack charge cycle (for a balance-at-top design strategy) when the pack is otherwise resting and therefore when voltage is a reasonable indicator of SOC. The remaining architectures allow independent activation or deactivation of balancers for individual cells, so can be used for more general dynamic SOC balancing.

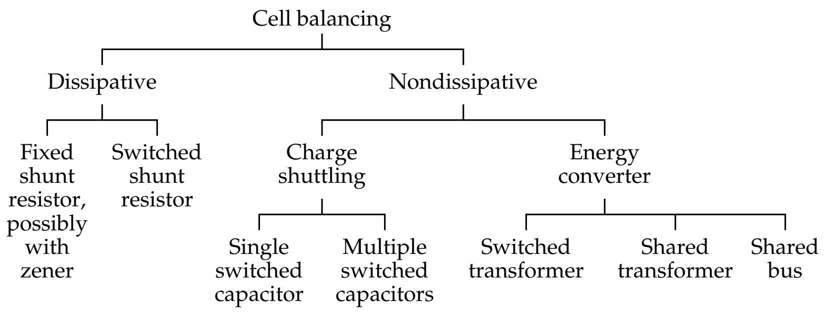


Figure 5.7: Taxonomy of cell-balancing circuits.

5.4.1 *Dissipative: Fixed shunt resistor*

The simplest electronics designs are for dissipative balancing systems, which place a resistor in parallel with every cell. This resistor is used to drain excess charge from the cell and the energy that is removed from the cell is dissipated as heat.

The most basic design of all is the "fixed shunt resistor design," illustrated in Fig. 5.8. To the left of the diagram, we see the stack of battery cells. To the right, a fixed resistor is placed in parallel with every cell, where all resistors are chosen to have equal value. Cells having higher SOC will generally have higher terminal voltage and so the discharge current passing from a cell through its connected resistor will also be greater than the discharge currents passing from other cells through their connected resistors. Therefore, cells having higher SOCs will discharge more quickly than the others, thereby bringing all cells into balance.

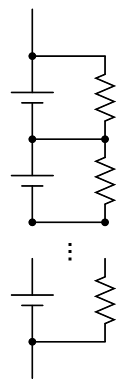


Figure 5.8: Fixed shunt resistor design.

The genius of this design is its simplicity. No voltage monitoring, SOC estimation, or active controls are needed. It operates entirely autonomously. However, also note that the circuit continuously dissipates charge, even when the pack is perfectly balanced. Therefore, the resistances should be chosen with very large value to minimize this energy loss.

Overall, a fixed shunt resistor design does not simplify a BMS design as much as it would seem at first. If lithium-ion battery cells are used, we must still monitor every cell's voltage for safety purposes and we must still estimate every cell's SOC for energy and power calculations. This circuit is probably best suited for balancing lead-acid or nickel-metal-hydride battery packs, where some overcharge is acceptable and so individual cell voltage monitoring is not needed, and for applications where the battery pack is almost always in a fully charged state such as for uninterruptible power supplies. If the battery-pack cells are not continuously "topped up" from an external source, then they will all discharge down to 0 V in a relatively short period of time.[6]

[6] However, this circuit can work well for balancing supercapacitor banks, since a balancing setpoint of 0 V can make sense in that application.

A variation on the fixed shunt resistor design uses zener diodes to turn balancing off automatically when cell voltage drops below some point. This architecture is illustrated in Fig. 5.9. The zener diodes must be sized to allow passage of full battery-pack charging current, and their cutoff voltages are chosen to correspond to a "100 % SOC" setpoint (e.g., about 2.2 V for a lead-acid cell). When a cell's voltage is above the zener setpoint, the resistor path is activated and that particular cell's charge is depleted until its voltage drops below the zener setpoint. This design also works best for chemistries where overcharge is tolerable and the cell can "float." This includes lead-acid and some nickel-based chemistries, but *not* lithium-ion chemistries. It is still limited by the fact that voltage alone is a poor indicator of SOC when the cell is not resting, even for lead-acid and nickel-based cells.

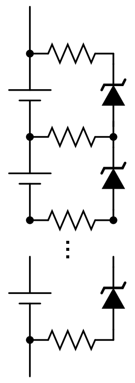


Figure 5.9: Fixed shunt resistor design, with zener modification.

5.4.2 *Dissipative: Switched shunt resistor*

A variation on the above idea, which works for lithium-ion chemistries as well, is to replace each zener diode with a transistor that is controlled by the BMS to enable or disable balancing of individual cells. This design is illustrated in Fig. 5.10. Transistors are drawn simply as switches; in an actual design, the type and rating of each transistor would need to be specified, and appropriate gate biasing circuitry would need to be added.

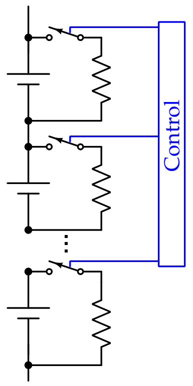


Figure 5.10: Switched shunt resistor.

The electronics required to control the transistors make this approach more complicated than either of the fixed shunt resistor designs; however, it allows for much greater flexibility in balancing strategy. The BMS simply closes switches on cells that are determined to have too much charge, allowing them to drain relative to the other cells whose balancing switches are open.

Note that the added complexity is not as big a concern as it used to be. Modern battery-stack monitoring chips often have built-in circuitry to control either an internal transistor switch (for slow balancing) or an external transistor switch (for faster balancing). In the former case, all that needs to be added by the designer is an external

12:07:34.

resistor for every cell; the latter case additionally requires an external transistor circuit.

THE PRIMARY ADVANTAGE OF any of these dissipative balancing approaches, as compared with nondissipative designs, is the simplicity (and hence, lower cost) of the circuitry involved. The drawbacks are:

- Energy that might otherwise be used productively is wasted as heat.
- In a balance-at-top design, energy remains in some cells when the weakest cell is completely discharged that could be utilized by the load if a nondissipative balancing system were used instead.
- Heat is generated. The power dissipated as heat is $p \approx v_{\text{nom}} \times i_{\text{balance}}$. For fast dissipative balancing, more heat is generated than for slower balancing. This generally imposes a high-wattage requirement on the balancing resistors. The quantity of heat generated by balancing can be similar to the heat generated by normal cell operation. Therefore, dissipative balancing may increase the cooling requirements for the battery-pack thermal-management system, which is a significant expense.
- Battery-pack life could be shorter than that of a pack with a nondissipative balancing design. With dissipative balancing, pack life is determined by the weakest cell in the pack. With nondissipative balancing, pack life is determined by the average cell. Further, nondissipative balancing can use strong cells to support weak cells, bringing the pack to a uniform end-of-life configuration. Dissipative balancing does not have this ability.

5.4.3 *Nondissipative: Multiple switched capacitors*

Referring back to Fig. 5.7, nondissipative balancing circuits break down into two general categories: some are based on the principle of moving charge from cell-to-cell via intermediary switched capacitors; others use switched-transformer or inductor-based designs and can be thought of as energy converters. We look first at capacitor-based balancing circuits.

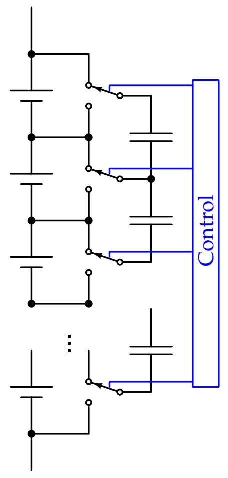


Figure 5.11: Balancing via a bank of capacitors.

Consider the circuit drawn in Fig. 5.11, where there is one fewer capacitor than there are battery-pack cells. The single-pole double-throw transistor-based switches repeatedly move back and forth synchronously. That is, all switches are placed in the top position for a period of time, then all are placed in the bottom position for a period of time, and then the cycle repeats. There is no intelligence behind the switching.

As the circuit operates, cell voltages are balanced. Consider two neighboring cells. The higher-voltage cell charges the capacitor to

12:07:34.

its voltage level when the switches are aligned to place the capacitor across its terminals. Then, when the switches are aligned across the lower-voltage cell, the capacitor discharges down to this lower voltage. In the process, charge moves from the higher-voltage cell to the lower-voltage cell via the capacitor.

Over the course of time, the entire battery pack can be equalized. However, if the highest-voltage cell is on the opposite end of the battery pack from the lowest-voltage cell, charge has to propagate through all intermediate cells (and capacitors) before overall balance is achieved. This can take a considerable amount of time.

5.4.4 *Nondissipative: One switched capacitor*

An alternate design uses a single switched capacitor with intelligent control, such as is drawn in Fig. 5.12. The transistor switches can be configured to place the capacitor across the terminals of any cell in the battery pack, allowing for more direct movement of charge from a high-voltage (or, high SOC) to a low-voltage (or, low SOC) cell.

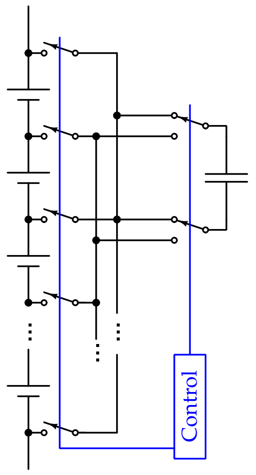


Figure 5.12: Balancing via a single capacitor.

However, a serious drawback of *all* capacitor-based designs is that they rely on a voltage difference between cells in order to work. Most types of lithium-ion cells have little variation in terminal voltage even if their SOCs are significantly different. This tends to make balancing using capacitor circuits very slow.

To see this, consider a first-order approximate analysis. The maximum energy that can be transferred from one cell to another in a single switching operation can be calculated via the capacitor energy equation based on the different cell voltages:

$$e = \frac{1}{2}C(v_{\text{high}}^2 - v_{\text{low}}^2).$$

This change in energy can be related to a change in SOC via the battery-cell energy Eq. (1.6):

$$e \approx (\Delta z)Qv_{\text{nom}}.$$

Now, suppose that $v_{\text{nom}} \approx (v_{\text{high}} + v_{\text{low}})/2$. Then, equating the two energies gives:

$$\begin{aligned}(\Delta z)Q\frac{v_{\text{high}} + v_{\text{low}}}{2} &\approx \frac{1}{2}C(v_{\text{high}} + v_{\text{low}})(v_{\text{high}} - v_{\text{low}})\\ \Delta z &\approx \frac{C}{Q}\Delta v,\end{aligned}$$

where Q must be measured in coulombs for the units to agree.[7]

[7] Note that one cell's SOC decreases by this amount and the other cell's SOC increases by this amount. Therefore, the net dispersion between cells closes at twice this rate.

As an example of how to apply this result, consider a 10 Ah cell (36,000 C). We would like to compute the change in SOC Δz for a single switching operation when $\Delta v = 0.1$ V. We are free to select

12:07:34.

the capacitance value, but note that high-valued capacitors tend to have high resistance and so will charge slowly, a fact we have not taken into account in our simple approximations. Even if we select an unrealistically large value of $C = 1\,\text{F}$, we get

$$\begin{aligned}\Delta z &\approx \frac{1}{36{,}000} \times 0.1\\ &\approx 3 \times 10^{-6}.\end{aligned}$$

That is, every time a capacitor is charged from one cell and discharged into a cell having 0.1 V lower voltage, the change in the SOC for each of the two cells is on the order of 0.0003 %. At this rate, it will take forever to equalize! Furthermore, as the cells become closer to balance, the value of Δv will approach zero, and balancing will slow down even further.

5.4.5 *Nondissipative: Switched transformer*

An alternative approach, which can move charge at a much greater speed, is to use a transformer, as is illustrated in Fig. 5.13. The primary side of the transformer is connected to the overall module or battery-pack voltage. The secondary of the transformer can be switched to connect across any of the cells in the module.

Rapidly switching the input to the primary creates an approximate ac waveform that is reproduced at the secondary. The primary is connected across n cells, and the transformer is wound with an $n : 1$ ratio. This causes the output voltage of the transformer to be decreased by factor of n, but also causes the output current to be increased by a factor of n. The diode at the secondary side ensures that charge is only ever added and not removed from cells on the secondary side.

This method is much more efficient than dissipative balancing and is much faster than the capacitive methods, but may be more expensive due to transformer and electronics costs. Presently, a number of silicon vendors are working to create automated control chips that will make this design feasible for future mass-produced products.

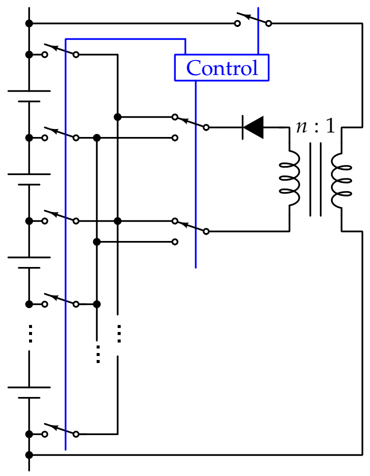


Figure 5.13: Switched transformer design.

5.4.6 *Nondissipative: Shared transformer*

Fig. 5.14 shows a simplified version of Fig. 5.13. The transformer in this circuit uses a custom winding such that there are n turns of the primary for each turn of all the secondaries. That is, the overall battery-pack or module voltage is transformed down to a single-cell level, and diodes ensure that energy is dumped only into cells having too-low voltage. The control rapidly switches the primary; diodes route the current. Balancing is automatic without sophisticated algorithms.

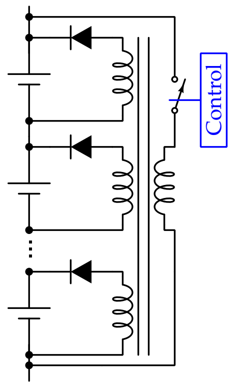


Figure 5.14: Shared transformer design.

12:07:34.

5.4.7 *Nondissipative: Shared bus*

There are significant benefits that can be accrued from nondissipative balancing versus dissipative balancing. Nondissipative balancing is more energy efficient; it can be much faster; individual cells can be both charged and discharged on demand in some designs (meaning that a single low or high cell will not seriously slow down balancing); and time-varying balancing setpoints can be used to enable use of all energy stored in the battery pack.

Yet, despite all of these advantages, dissipative balancers are used in the vast majority of present fielded BMS. The reason is cost. Switched-resistor balancers are used almost exclusively because they are very inexpensive. Nondissipative balancers add electronics cost for every cell, and transformer-based methods add expense for the transformers themselves.

The final balancing architecture that we look at is relatively new but has the potential to make nondissipative balancing cost-neutral with respect to dissipative balancing in some applications. The circuit topology is drawn in Fig. 5.15.[8] The approach uses one small isolated dc–dc converter per cell, where the input to each converter is connected directly to the cell and the outputs from all converters are connected to a shared capacitive low-voltage bus.

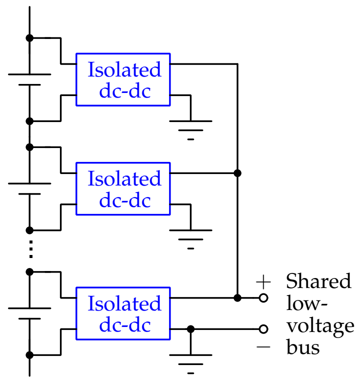


Figure 5.15: Shared-bus balancing topology.

[8] Ur Rehman, M.M., Evzelman, M., Hathaway, K., Zane, R., Plett, G.L., Smith, K., Wood, E., and Maksimović, D., "Modular Approach for Continuous Cell-level Balancing to Improve Performance of Large Battery Packs," in *Proc. IEEE Energy Conversion Conference and Expo* (ECCE), Pittsburg, 2014.

This approach sounds more complex than a switched-resistor dissipative system. It is. However, that does not necessarily make it more expensive. Relative costs are difficult to evaluate fairly and are subject to change. Moreover, and very importantly, the shared low-voltage bus can displace other expensive system components so that the overall architecture is financially competitive.

To see how this can be the case, we consider what the low-voltage bus might be used for. The high-voltage bus is connected directly to the high-voltage load. However, the low-voltage bus can be connected to an auxiliary load. For an example, in xEV applications the high-voltage battery pack and high-voltage vehicle systems are always complemented by a 12 V battery and 12 V systems. This is in part because of the need to power legacy systems, but also due to some subtle safety concerns relating to a requirement to be able to distribute low power levels to parts of the vehicle without closing the high-voltage contactors.

In present xEV, this 12 V system is run by a 12 V lead-acid battery that is charged by a high-voltage to 12 V dc–dc converter. This high-voltage dc–dc converter plays the same role that an alternator plays in a standard gasoline-powered vehicle, keeping the lead-acid battery fully charged using engine power.

The low-voltage bus in a shared-bus balancing system can be

12:07:34.

designed to operate in any reasonable voltage range. If we specify the low-voltage bus to have nominal voltage of 12 V, then we can entirely replace the high-voltage to 12 V dc–dc converter in present xEV designs and we can either eliminate or significantly shrink the 12 V lead-acid battery. Essentially, a single expensive high-voltage to 12 V dc–dc converter is replaced by a lot of small dc–dc converters: one per cell.

The savings accrued by displacing the high-voltage dc–dc converter and by shrinking the lead-acid battery causes the shared-bus balancing system to become approximately cost-neutral with respect to switched-resistor dissipative balancing systems in xEV. However, the shared-bus method can additionally attain all the benefits of nondissipative balancing, which also have significant long-term monetary value (that can be difficult to quantify). It is a very promising technique, even outside of xEV, whenever a low-voltage auxiliary load must be powered in addition to the high-voltage load.[9]

[9] For example, in a grid-storage, grid-backup, or frequency-regulation application the shared bus can power a thermal-management system, etc.

Balancing is performed by using the voltage of the shared bus as an indicator of the battery-pack average of some battery-cell quantity. That is, we don't fix the bus voltage at exactly 12 V, but we allow it to vary somewhat. For example, we might allow it to vary within the standard lead-acid battery voltage range of about 11 V to 13.5 V. Whenever charge is moved from a cell into the bus, the voltage of the bus will tend to increase; whenever charge moves from the bus into a cell, the voltage of the bus will tend to decrease.

We can use different mappings between some cell quantity and bus voltage to achieve different desired tasks, including maximizing life, maximizing the energy that can be removed from the battery pack, and so forth. For now, consider the straightforward case where we simply want to balance the SOCs of all cells at every point in time. Then, define the metric function

$$v_k = 11 + 1.5z_k.$$

Every cell computes its own value for v_k, between 11 and 13.5. It then compares its own value to the voltage of the shared bus. If this cell's value of v_k is below the shared-bus voltage, then its SOC is below the battery-pack-average SOC. It draws energy from the bus to charge the cell, raising its own SOC and hence its own value of v_k; simultaneously, the shared-bus voltage decreases because of the charge that has been removed. Alternately, if this cell's value of v_k is above the shared-bus voltage, the opposite process takes place. In both cases, the cell SOC becomes closer to the battery-pack-average SOC and the shared-bus voltage automatically changes to reflect this fact.

No communication among cells is necessary. The analog shared-bus voltage is sufficient to coordinate all balancing tasks. Also, the power requirements imposed on the dc–dc converters for cell balancing are relatively small. They do not need to process the entire power of the battery pack as with some other designs. Instead, they need to process only the relatively small mismatch power between cells.

As BATTERY PACKS GROW IN SIZE, dissipative balancing techniques make less and less sense. The amount of power that must be dissipated in the form of heat leads to very high wattage resistors and to significant load on the battery-pack thermal-management system. As costs of nondissipative methods come down, they are very likely to overtake dissipative methods in fielded applications.

Before continuing, we should also mention safety concerns when balancing. As long as the balancing circuitry is working correctly, the methods described throughout this book can be used to monitor cell states and ensure safety. However, it is also important to consider what would happen if the balancing circuitry fails. Failures in dissipative balancing circuits tend to be relatively benign. One or more cells may overdischarge, which will damage the cells but not cause a safety hazard. Failures in nondissipative balancing circuits can be more serious. If a nondissipative balancer that is charging a cell gets "stuck on," then that cell may become overcharged, leading to risks of thermal runaway. The circuitry should be designed with these concerns in mind to provide redundant shutdown paths that can maintain safety even if part of the circuit fails.

5.5 *How quickly must I balance?*

When designing a balancing system, a final decision that must be made has to do with how quickly the balancing circuits are required to move or remove charge. In a switched-resistor dissipative system, this requirement enables sizing the balancing resistor that is placed in parallel with each cell; in a nondissipative system, this will enable design of the corresponding circuit component values.

If our primary concern is setpoint balancing for long-term balancing needs, then we must be able to balance the battery pack at least as quickly as it can become imbalanced. If, instead, we are interested in dynamic setpoint balancing using fast nondissipative balancing, then we will need to be able to move charge much more quickly to ensure that the dynamic setpoint can always be tracked.

Either way, battery-pack simulation is an excellent tool to evaluate how quickly a battery pack can become imbalanced. The simulator program will need to:

1. Create a virtual battery pack by drawing random numbers for the

parameter values of all of its cells, such that the randomness of the parameters is typical of the types and magnitudes of variation expected in real cells;

2. Simulate this virtual battery pack over many repeated realistic usage scenarios;
3. Gather statistics on how quickly cells become imbalanced and how much imbalance there is as a function of time;
4. Save these statistics plus the parameter values for this random battery pack for later use and analysis;
5. Repeat this process for numerous battery packs—enough that the results are statistically significant.

Step 1 of this process emulates the creation of a battery pack by assembling cells as they are produced by a manufacturing line. Steps 2 and 3 simulate the operation of this virtual battery pack in its application and record results. Step 4 is careful to save the configuration of this battery pack, so that future simulations that consider balancing its cells don't start fresh with new random cells but use exactly the same cell characteristics that were used to simulate the pack in Step 2 and produce the results of Step 3.

The output of this process can tell us quite a few things, including which cell and battery-pack characteristics are the greatest contributors to imbalance. It can be very helpful for gaining insight into the causes and extent of imbalance in a battery pack.

Once we have simulated the battery pack without balancing, we can load these saved data and subsequently simulate the balancing process of the battery pack. For fixed-setpoint balancing, the amount of time spent balancing must be less than the time taken to become imbalanced to have a stable steady-state solution. For dynamic setpoint balancing, we must be able to move charge more quickly than this and we would need to be able to cosimulate the imbalancing and the balancing processes to ensure that the balancing methods can keep up.

In the remainder of this section, we examine MATLAB code `simRandPack.m` that is designed to simulate packs having random cell characteristics, and then look at some sample results. As is our practice by now, we will consider this script in sections that would need to be reassembled in order to create an executable MATLAB program.

The first code segment gives help information and sets up the battery-pack cells' random parameters and other simulation variables. A battery pack comprising `Ns` cells is created and will be simulated for `Nc` discharge/charge cycles. Each discharge cycle comprises repetitions of the power versus time profile in `cycleFile`. The ESC-format cell model is passed as `model`. The `randOptions` vector has entries that are set to "0" for a simulation using standard parameter values from the `model` structure or "1" for random values, as unpacked below the function header. These entries will be described in more detail in the sections where they are first used.

```
% ------------------------------------------------------------------
% simRandPack: Simulate battery pack having Ns cells in series for Nc
% discharge/charge cycles, where all cells in pack can have random
% parameter values (e.g., total capacity, resistance, etc.)
%
% Assumes no hysteresis in the cell model (this could be changed
% fairly easily; hysteresis makes results more difficult to interpret,
% so this assumption is okay for a first analysis, at least).
% ------------------------------------------------------------------
function packData = simRandPack(Ns,Nc,cycleFile,model,randOptions)

  tOpt  = randOptions(1); qOpt = randOptions(2); rOpt = randOptions(3);
  sdOpt = randOptions(4); cOpt = randOptions(5); lOpt = randOptions(6);
  profile = load(cycleFile); % e.g., 'uddsPower.txt'

  % Create storage for all cell states after completion of each cycle
  packData.storez = zeros([Ns Nc]);  % create storage for final SOC
  packData.storeirc = zeros([Ns Nc]);

  % Initialize default states for ESC cell model
  maxSOC = 0.95; % cell SOC when pack is "fully charged"
  minSOC = 0.1;  % cell SOC when pack is "fully discharged"
  z   = maxSOC*ones(Ns,1); % start fully charged
  irc = zeros(Ns,1);       % at rest
  ik  = zeros([Ns 1]);     % current experienced by each cell
```

The output of this function is `packData`, which stores all cells' random parameters and both SOC and diffusion-resistor states after each discharge/charge cycle.

The next section populates random variables for this battery pack. The default temperature `T` for every cell is 25 °C but, if `tOpt` is set, then every cell is assigned a random temperature uniformly distributed between 22.5 °C and 27.5 °C. An auxiliary temperature `Tsd`, which will be used later for cell self-discharge calculations, is set to a random value uniformly distributed in the range `T` plus or minus 5 C°. The default leakage current is set to 10 mA, but if `lOpt` is set, then every cell is assigned a random leakage current uniformly distributed between 10 mA and 12 mA:

```
  % Set cell temperatures based on tOpt
  if tOpt, % set to "1," to execute, or "0," to skip this code
    T = 22.5 + 5*rand([Ns 1]);
  else
    T = 25*ones([Ns 1]);
  end
  % Set self-discharge "cell temperature"
  Tsd  = T - 5 + 10*rand([Ns 1]);

  % Set cell module leakage current based on lOpt
  if lOpt,
    leak = 0.01 + 0.002*rand([Ns 1]);
  else
    leak = 0.01*ones([Ns 1]);
  end
```

The main cell simulation parameters are now retrieved from the `model` data structure for each individual cell at its temperature `T`. In this code, the default coulombic efficiency is set to one, and interconnect resistance is set to twice the cell tab resistance, or 250 $\mu\Omega$ per cell:

```
% Default initialization for cells within the pack
% Note that since T has Ns elements, there is one parameter value
% per cell (even if all turn out to be identical)
q   = getParamESC('QParam',T,model);
rc  = exp(-1./abs(getParamESC('RCParam',T,model)));
r   = (getParamESC('RParam',T,model)).*(1-rc);
r0  = getParamESC('R0Param',T,model);
rt  = 2*0.000125; % 125 microOhm resistance for each tab
eta = ones([Ns 1]);
```

If certain options in `randOptions` are set, then these default cell parameters can be overwritten. In the next segment of code, cell total capacity can be randomized uniformly between its default value plus or minus 0.25 Ah. Cell resistance can be randomized uniformly between its default value minus 0.5 mΩ and its default value plus 1.5 mΩ. Coulombic efficiency can be randomized uniformly between 0.997 and 0.999:

```
% Modified initialization for cell variability:
% Set individual random cell-capacity values
if qOpt, % set to "1," to execute, or "0," to skip this code
  q=q-0.25+0.5*rand([Ns 1]);      % random capacity for ea. cell
end

% Set individual random cell-resistance values
if rOpt, % set to "1," to execute, or "0," to skip this code
  r0 = r0-0.0005+0.0015*rand(Ns,1);
end
r0 = r0 + rt;  % add tab resistance to cell resistance
R = sum(r0,1); % overall total ohmic resistance of battery pack

% Set individual random cell-coulombic-efficiency values
if cOpt, % set to "1," to execute, or "0," to skip this code
  eta = eta - 0.001 - 0.002*rand([Ns 1]);
end
```

The next code segment sets maximum and minimum allowed voltages, initializes some simulation variables, and starts the simulation:

```
% Now, simulate pack performance using ESC cell model.
maxVlim = min(OCVfromSOCtemp(maxSOC,T,model));
minVlim = max(OCVfromSOCtemp(minSOC,T,model));
theCycle = 1; theState = 'discharge';
disCnt = 0; % start at beginning of profile
fprintf('  Cycle = 1, discharging... ');
while theCycle <= Nc,
  v  = OCVfromSOCtemp(z,T,model); % get OCV for each cell
```

```
    v  = v - r.*irc;         % add in diffusion voltages
    V  = sum(v);             % total "fixed" voltage excluding i*R
    vt = v-ik.*r0;           % cell terminal voltages
```

The simulation is organized as a two-state state machine, which can be in either a "discharge" or a "charge" state at any point in time. The discharge state simulates a usage profile of power versus time such as a drive-cycle profile in a vehicle. The charge state simulates a plug-in charging operation. In both cases, we compute required battery-pack current every iteration, modify it by cell characteristics, and update every cell's state.

While in the discharge state, we execute the profile from `cycleFile` repeatedly until a lower voltage or lower SOC limit is reached. Each time step, we retrieve the next value of power as a function of time from the `profile` variable, wrapping from the end of the profile back around to its beginning whenever we exceed the profile length. We then compute battery-pack current from this demanded power and modify it by the cell's coulombic efficiency if the required current is negative. The values in the vector `ik` comprise the cell currents to be used to update the cell states before the next time step. However, if we discover that we have hit a lower voltage or SOC limit, we abandon the computed values in `ik` and switch the simulation state machine from the discharge state to the charge state:

```
    switch( theState )
      case 'discharge';
        % Get instantaneous demanded pack power, repeating profile
        P = profile(rem(disCnt,length(profile))+1);
        % Compute demanded pack current based on unloaded voltage
        I = V/(2*R) - sqrt(V^2/R^2 - 4*P/R)/2;
        % Default cell current = pack current
        ik = I*ones(Ns,1);
        if I < 0, % If we happen to be charging this momement
          ik = ik.*eta;
        end
        if min(z) <= minSOC || min(vt) < minVlim,   % stop discharging
          theState = 'charge';
          chargeFactor = 1;
          ik = 0*ik;
          fprintf('charging... ');
        end
        disCnt = disCnt + 1;
```

In the charge state, we begin by charging the battery pack at a 6.6 kW level, representative of a domestic "level 2" xEV charging system. We compute battery-pack current and then cell current from the demanded power in much the same way as we did while in the discharge state of the state machine. When we reach an upper voltage limit, we divide the present charge power in half and continue charging. We repeat this process until charge power has been reduced beyond a 6.6/32 kW rate. After that, we consider the pack to be fully charged, store results for this completed discharge/charge cycle, and switch the simulation state machine back from the charge state to the discharge state to begin the next cycle:

```
      case 'charge';
        % start charging @ 6.6kW, then taper
        P = -6600/chargeFactor;
        I = V/(2*R) - sqrt(V^2/R^2 - 4*P/R)/2;
        I = max(-min(q),I); % limit to 1C charge rate max
        ik = I*eta;                   % Charge coulombic eff.
        if max(vt)>=maxVlim,
          if chargeFactor > 32, % bail after 6.6kW/32 charge
            packData.storez(:,theCycle) = z;
            packData.storeirc(:,theCycle) = irc;
            theState = 'discharge';
            disCnt = 0;
            ik = 0*ik;
            theCycle = theCycle + 1;
            if theCycle <= Nc,
              fprintf('\n  Cycle = %d, discharging... ',theCycle);
            end
          end
          chargeFactor = chargeFactor*2;
        end
      otherwise
        error('charge/discharge state has been corrupted')
    end
```

By this point in the main loop, we have computed the cell currents `ik` demanded by the load for this time step. It remains to add the self-discharge and leakage currents. In the following, we model temperature-dependent self-discharge as current flowing through a fictitious resistance in parallel with the cell, where we use the self-discharge temperature `Tsd` computed earlier as the basis for cell-to-cell variation in this resistance. We add the leakage currents, compute new SOC and diffusion-current states, and repeat:

```
    % Simulate self discharge via variable resistor in parallel
    if sdOpt,
      rsd = ((-20+0.4*Tsd).*z + (35-0.5*Tsd))*1e3;
      ik  = ik + vt./rsd;
    end

    % Simulate leakage current
    ik = ik + leak;

    z = z - (1/3600)*ik./q;             % Update each cell SOC
    irc = rc.*irc + (1-rc).*ik;         % Update resistor currents
  end % end while
  fprintf('\n');
  packData.q = q; packData.rc = rc; packData.eta = eta;
  packData.r = r; packData.r0 = r0; packData.Tsd = Tsd;
  packData.T = T; packData.leak = leak;
end
```

When all `Nc` discharge/charge cycles have been simulated, parameters and results are stored to the `packData` structure, and the function returns.

This function would generally be invoked by a wrapper script that simulates many random packs (to gather statistics). Any and all combination(s) of random options can be selected to explore sensitivity of imbalance to each cause. The `packData` structures can then be stored for later analysis using balancing algorithms to see how quickly balancing must occur.

5.6 *Balancing simulation results*

To illustrate some of the things we can learn from this code, we consider seven different scenarios. Each scenario simulates 30 discharge/charge cycles for each of 100 battery packs, where every battery pack has 100 cells and 7.7 Ah nominal capacity. The first scenario considered random cell temperatures with `tOpt=1`, but default capacity, resistance, self-discharge, coulombic efficiency, and leakage current. The second scenario set `tOpt=0`, but considered random capacities with `qOpt=1` and default values for all other simulation parameters. The third through sixth scenarios considered `rOpt=1`, `sdOpt=1`, `cOpt=1`, and `lOpt=1`, respectively. Finally, the seventh scenario had all randomizations turned on. Thescenario definitions are summarized in Table 5.1.

	Temperature	Capacity	Resistance	Self-discharge	Coulombic efficiency	Leakage current
Scenario 1	**random**	standard	standard	off	ideal	uniform
Scenario 2	uniform	**random**	standard	off	ideal	uniform
Scenario 3	uniform	standard	**random**	off	ideal	uniform
Scenario 4	uniform	standard	standard	**on**	ideal	uniform
Scenario 5	uniform	standard	standard	off	**random**	uniform
Scenario 6	uniform	standard	standard	off	ideal	**random**
Scenario 7	**random**	**random**	**random**	**on**	**random**	**random**

Table 5.1: Simulation scenarios

Fig. 5.16 shows histograms for cell SOCs at the end of 30 discharge/charge cycles for all cells in all battery packs. As expected, temperature variation by itself did not cause SOC dispersion. Capacity variation *appears* to have caused a small amount of dispersion, but this is misleading. The simulation is initialized with all cells having 95 % SOC. However, recharge ends when a voltage limit is reached and this occurs before any cell is charged exactly back to 95 % SOC. If the voltage limit were removed, then all cells would reach 95 % SOC at the same time and the pack would be perfectly balanced.

Resistance variation did not cause dispersion in SOC and self-discharge variation caused relatively little dispersion during this simulated period of time. However, self-discharge is a time-duration

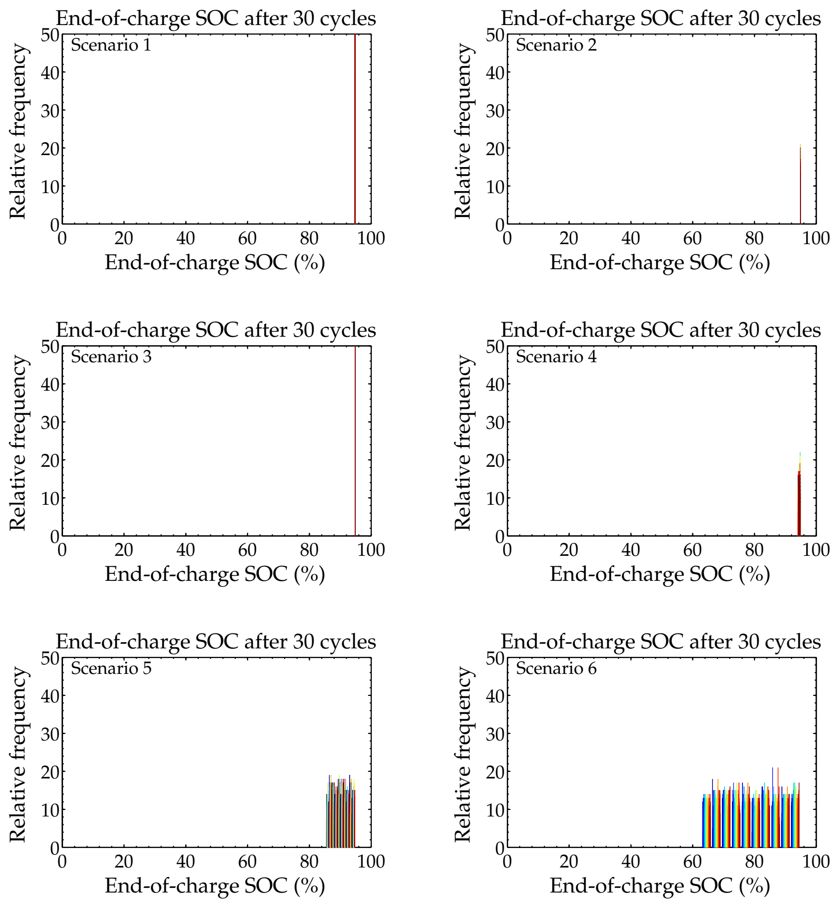


Figure 5.16: SOC dispersion after 30 discharge/charge cycles.

based phenomenon, and so dispersion increases when the simulation is allowed to run longer.

The majority of imbalance is caused by nonuniform coulombic efficiencies or leakage currents. Note that the simulated variation in these parameter values was fairly modest. In practice, the variation could be even higher. Therefore, we consider these to be the dominant causes of imbalance.

NEXT, WE RAN SIMULATIONS to see how quickly a battery pack could be brought back into balance from an initial imbalanced state. We started with the set of data from scenario 7 at the end of 30 discharge/charge cycles, where cell SOCs had diverged to span the interval from 55 % to 95 %. Already, we see that without balancing the battery pack can become worthless very quickly, even though all the cells therein are still perfectly healthy. We can also recognize that we need to balance a little more than 1 % of pack capacity per balancing period to keep the pack in balance, since it diverged about 40 % over 30 cycles.

During the balancing phase, the pack was nominally at rest (there was no externally applied current). Cell terminal voltages were measured once per second. The measurement added ± 0.5 mV of uniformly distributed measurement noise to approximate system noises and quantization errors. All cells having a measured voltage that was at least 2 mV above the present minimum measured cell voltage were selected for balancing using a dissipative switched-resistor strategy.

Balancing was accomplished by placing a 100 Ω resistor in parallel with every cell selected for balancing. This causes a small amount of discharge, lowering SOC and lowering terminal voltage for these cells.

Four hours (14,440 s) of real time were simulated per balancing period, where balancing decisions were made once per second. This period is shorter than would normally be available per discharge/charge operational cycle to balance a battery pack during plug-in mode so represents an approximate worst-case scenario. At the end of the four hours, a histogram of the SOC values for all battery-pack cells was recored. Then, *without reinitialization*, additional four-hour periods were simulated, with histograms recorded at the end of each four-hour period. Results are shown in Fig. 5.17.

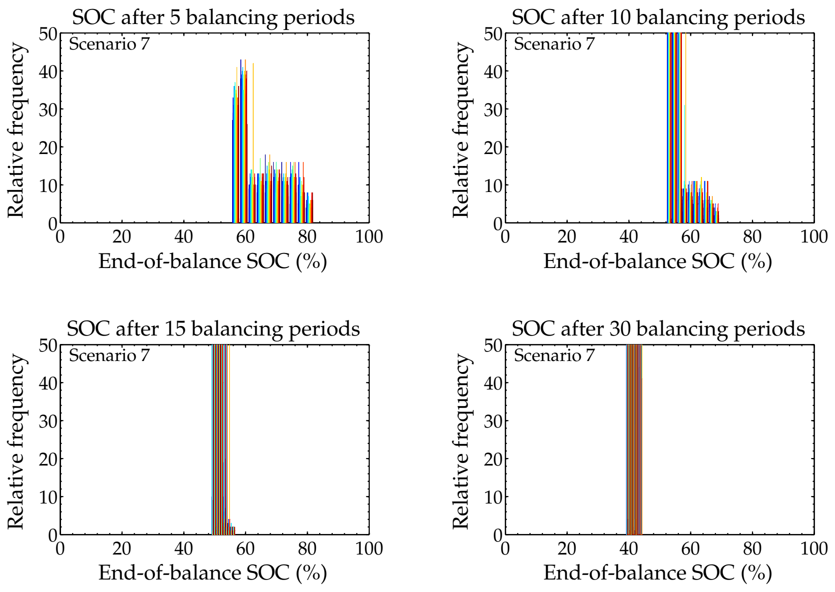


Figure 5.17: Pack balance after 30 discharge/charge cycles and a variable number of balancing periods.

In the first five balancing periods, the top SOC were lowered from 95 % down to about 80 %, reducing the overall battery-pack SOC dispersion from 40 % to about 25 %. The next five cycles reduced the dispersion to about 15 %, but we also see that the lower states of charge began to decrease: we sometimes balance the wrong cells due to the ± 0.5 mV measurement error. Balance improved slightly by 15 balancing periods, and somewhat more by 30 balancing periods, but by then the voltage measurement noise caused the remaining imbalance to be permanent.

So, we can conclude that 30 balancing periods are sufficient to balance a pack that had been exercised for 30 discharge/charge cycles. Even this relatively simple dissipative switched-resistor method can keep a battery pack in balance, if all that is desired is setpoint balancing.

The decision regarding whether to use dissipative or nondissipative balancing is not one of capability. Dissipative balancing is able to keep a battery pack balanced. However, there are issues of efficiency, heat generation, and battery-pack longevity. Nondissipative methods are more energy efficient, can extend the life of the battery-pack cells, and do not generate as much heat. Dissipative balancing, especially for battery packs having higher total capacity, require smaller balance resistors and higher balancing currents, which will in turn cause greater heating. And, dissipative balancing in general will not be able to maximize battery-pack available energy or power, or to extend life via quick charge transfer.

5.7 *Where to from here?*

We have now considered, at some level of detail, all of the BMS requirements outlined in Chap. 1. We have seen how to estimate SOC and SOH for every cell in the battery pack and how to balance the cells to keep the pack in a functional state. The SOC and SOH estimates can be used to compute a total energy estimate using the simple equations in Chap. 1. However, the power-limit estimation approach sketched in Chap. 1 needs some further elaboration. We devote the remainder of the book to topics surrounding the computation of battery-pack power limits.

6
Voltage-Based Power-Limit Estimation

In Sect. 1.13.1, we discussed the estimation needs of a battery management system and concluded that the two most fundamental requirements were abilities to estimate present battery-pack total energy and available power. Since no sensor exists to measure either of these quantities directly, we must compute estimates of their values based on estimates of more primitive quantities that include cell SOCs, resistance, and total capacity.

Referring back to the roadmap Fig. 3.1 that diagrams our progress through the contents of this book, we recognize that we have now seen various methods to estimate the state (including SOC) and health (including total capacity and resistance) of every battery cell. This information is sufficient to compute the total energy stored by every cell and by the overall battery pack using the equations in Sect. 1.14.

So, it remains to study how to compute cell and battery-pack power limits. We previewed a simple approach to doing so in Sect. 1.15, but now return to look at this requirement more closely.

A POWER LIMIT tells us how quickly we may add energy to or remove energy from the battery pack without violating a set of design constraints. In this chapter, we assume that these constraints impose hard limits on cell terminal voltage, which is the common practice today. We will generalize the simple approach presented in Chap. 1 and see how to apply it using a higher-fidelity equivalent-circuit model of a cell, resulting in improved battery-pack dynamic power limits.

The essential reason for competing power limits is to optimize a tradeoff between the performance delivered by the battery pack and its expected lifetime. Voltage limits are enforced by the power-limits calculation in an attempt to prevent excessively rapid aging. However, it is important to recognize that voltage limits are a means to an end and not the end itself. The real concern, is not directly how large or small a cell's voltage might be, but rather the incremental damage that will be experienced by the cell if it is operated at high power levels. Voltage limits are used because they are easily computed, but voltage is an indirect and not very predictive indicator of how quickly the cell will age. Thus, in Chap. 7, we will look at some ideas for computing power limits based on model-predicted future incremental degradation rather than by placing constraints on future terminal voltage.

6.1 *Traditional, terminal-voltage-based power limits*

The power limits that are estimated by a battery management system are communicated to the load-management system and can be used for multiple purposes. If the load relies on the battery pack as its sole power source, such as in an electric vehicle, then the load controller must simply ensure that the power limits provided by the battery management system are never violated even if this results in a loss of performance provided by the load. If the load has multiple power sources, such as in a hybrid-electric vehicle, then the load controller uses the maximum limits from the battery management system as part of its strategy to blend the capabilities of the two sources in an intelligent way to satisfy the load requirements while optimizing some performance criterion.

In neither of these cases is a rapidly changing instantaneous estimate of available power of much use.[1] A slowly changing value is greatly preferred, to avoid requiring abrupt changes in the battery-pack load. For load-planning purposes, a predictive estimate of power is also better than an instantaneous estimate to allow for scheduling over a near-future time horizon.

[1] True available power does not change very quickly, so neither should its estimate.

So, we consider the problem of computing a predictive estimate of a constant power level available to the load over some future time horizon of duration ΔT s. If the load draws power up to but not exceeding this computed limit for the entire ΔT s, then no battery-pack design limits should be violated. Because the constant level of power that can be sustained for ΔT s is less than or equal to the maximum instantaneous sustainable power level, the predictive power estimate is conservative in some sense, and short-term exceedances can sometimes be absorbed.

The computation and communication of power limits is done much more frequently than once every ΔT s. This continual recomputation of available power acts as a type of low-pass-filter operation that smooths out changes in the estimates and enables the load-management system to avoid abrupt performance loss. This overlapping moving-window approach is illustrated in Fig. 6.1.

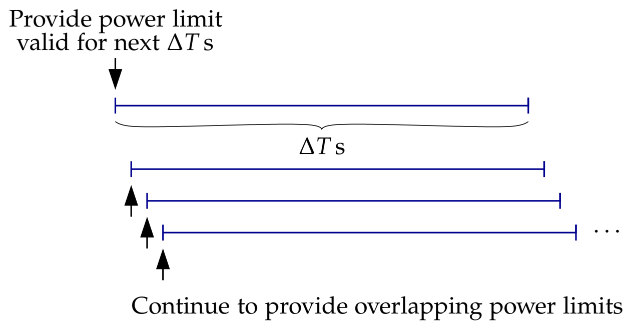

Provide power limit valid for next ΔT s
ΔT s
· · ·
Continue to provide overlapping power limits

Figure 6.1: Power limits must be predictive over a future time horizon ΔT.

For example, our prediction horizon might be $\Delta T = 10$ s, and our update rate might be 1 Hz. At time zero, we compute power limits that must be maintained until time $t = 10$ s. We guarantee that if the load draws power within these computed limits, no battery-pack design limits will be violated. However, at time $t = 1$ s, we recompute power limits that must be maintained from $t = 1$ s until time $t = 11$ s. These limits replace those computed at time $t = 0$ s and may be greater or smaller, depending on the actual battery-pack activity during the interval between $t = 0$ s and $t = 1$ s. At $t = 2$ s, we compute power limits valid from that point in time until $t = 12$ s, and so forth. This predict-and-update sequence is repeated continuously while the battery pack operates.

More formally, the problem that we address in this chapter may be described in the following way:

a) *Discharge power*: Based on present battery-pack conditions, estimate the maximum discharge power that may be maintained constant for ΔT seconds without violating design limits on cell terminal voltage, SOC, or maximum design power and current.

b) *Charge power*: Based on present battery-pack conditions, estimate the maximum battery charge power that may be maintained constant for ΔT seconds without violating preset design limits on cell terminal voltage, SOC, maximum design power, or current.

c) *Both discharge and charge power*: Any combination of (a) and (b), where ΔT may have different values for charge and discharge.

The notation and assumptions we employ are as follows. We denote:

- The number of cells in the battery pack by N;
- Cell terminal voltage at discrete-time step k for cell number i in the pack by $v_k^{(i)}$; where design limits $v_{\min} \le v_k^{(i)} \le v_{\max}$ must be enforced for all cells;
- SOCs at time k for cell i by $z_k^{(i)}$, where we enforce $z_{\min} \le z_k^{(i)} \le z_{\max}$;

- Cell power at time k for cell i by $p_k^{(i)}$, where we enforce $p_{\min} \le p_k^{(i)} \le p_{\max}$; and,
- Cell current at time k for cell i by $i_k^{(i)}$, where we enforce $i_{\min} \le i_k^{(i)} \le i_{\max}$.

Any particular limit ($v_{\max}$, $v_{\min}$, $z_{\max}$, $z_{\min}$, $i_{\max}$, $i_{\min}$, $p_{\max}$, or $p_{\min}$) may be removed if desired by replacing its value by $\pm\infty$, as appropriate. Any limit may furthermore be a function of temperature and other factors pertaining to the present battery pack operating condition. Different cells may have different limits should it be desirable. The battery pack is assumed to comprise N_s cell modules connected in series, where each cell module comprises N_p individual cells connected in parallel, with $N_s \ge 1$, $N_p \ge 1$, and $N = N_s N_p$.

6.2 *Voltage-based power limits, using a simple cell model*

As previewed in Chap. 1, a standard method for estimating power limits is one that we will refer to as the *Hybrid Pulse Power Characterization (HPPC)* method specified by the Partnership for New Generation Vehicles (PNGV). This method assumes the simplified equivalent-circuit model of a cell shown in Fig. 6.2. For this model, the instantaneous terminal voltage can be expressed as

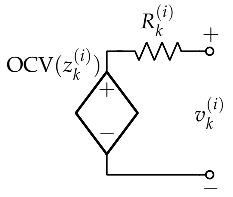


Figure 6.2: Crude equivalent-circuit model of a cell.

$$v_k^{(i)} = \mathrm{OCV}(z_k^{(i)}) - i_k R_k^{(i)}, \tag{6.1}$$

which can be rewritten to solve for cell current as

$$i_k = \frac{\mathrm{OCV}(z_k^{(i)}) - v_k^{(i)}}{R_k^{(i)}}.$$

To compute a power estimate, we first assume that we are concerned only with keeping the terminal voltage between $v_{\min}$ and $v_{\max}$. Then, discharge power is computed by clamping the cell terminal voltage to $v_{\min}$:

$$p_{\mathrm{dis},k}^{(i)} = v_k^{(i)} i_k = v_{\min} \frac{\mathrm{OCV}(z_k^{(i)}) - v_{\min}}{R_k^{(i)}}. \tag{6.2}$$

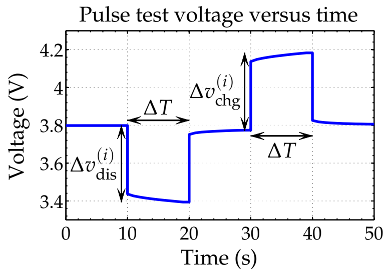


Figure 6.3: Determining the resistances $R_{\mathrm{dis},\Delta T}$ and $R_{\mathrm{chg},\Delta T}$ via a lab test.

As written, if $R_k^{(i)}$ is set to the equivalent-series resistance R_0 of the cell, then Eq. (6.2) computes the present instantaneous available power. To compute a power limit that applies over a future time horizon instead, we can modify the resistance in the denominator such that $R_k^{(i)} > R_0$ to model the greater change in voltage expected when applying an input pulse for a longer duration. But, what value of $R_k^{(i)}$ should we use?

To answer this question, notice that Eq. (6.1) uses $R_k^{(i)}$ to model the instantaneous change in cell terminal voltage when a pulse of current is applied to a resting cell. We can use the same equation to predict the overall change in voltage over a time interval of ΔT seconds if we instead choose $R_k^{(i)}$ such that $i_k R_k^{(i)}$ is representative of the accumulated voltage drop when a constant-current pulse is applied to a resting cell for ΔT seconds. A laboratory cell test such as the one illustrated in Fig. 6.3 can be performed to determine this value. In the figure, the cell has been allowed to rest for ten seconds, then a constant-current discharge pulse is applied for ΔT seconds ($\Delta T = 10\,\text{s}$ in this example), then the cell voltage is allowed to recover and a constant-current charge pulse is applied for ΔT seconds. We measure ΔV_{dis} as the initial rest voltage minus the minimum voltage during the constant-current discharge, and ΔV_{chg} as the intermediate rest voltage minus the maximum voltage during the constant-current charge. Then, we compute (positive) effective discharge and charge resistances over a ΔT-second pulse duration as:[2]

$$R_{\text{dis},\Delta T}^{(i)} = \left| \frac{\Delta v_{\text{dis}}^{(i)}}{i_{\text{dis}}} \right|, \qquad \text{and} \qquad R_{\text{chg},\Delta T}^{(i)} = \left| \frac{\Delta v_{\text{chg}}^{(i)}}{i_{\text{chg}}} \right|.$$

[2] This test should be repeated at multiple SOC and temperature setpoints to capture the influences in these factors on the effective discharge and charge resistances.

We assume that ΔT seconds may be represented in discrete time as exactly $k_{\Delta T}$ sample intervals. That is, $k_{\Delta T} = \Delta T / \Delta t$. With the results found to date, we can now approximate

$$v_{k+k_{\Delta T}}^{(i)} \approx \text{OCV}(z_k^{(i)}) - i_k R_{\text{dis},\Delta T}^{(i)}$$

for discharge currents and

$$v_{k+k_{\Delta T}}^{(i)} \approx \text{OCV}(z_k^{(i)}) - i_k R_{\text{chg},\Delta T}^{(i)}$$

for charge currents.

So, to compute maximum discharge current based on a design lower voltage limit, we set $R_k^{(i)} = R_{\text{dis},\Delta T}^{(i)}$ and clamp the future voltage $v_{k+k_{\Delta T}}^{(i)} = v_{\min}$. Then,

$$i_{\max,k}^{\text{dis,volt}(i)} = \frac{\text{OCV}(z_k^{(i)}) - v_{\min}}{R_{\text{dis},\Delta T}^{(i)}}.$$

If we are concerned only with maintaining cell voltage limits within the interval $v_{\min}$ to $v_{\max}$, then pack discharge power may be calculated as

$$p_{\max,k}^{\text{dis,volt}} = N_s N_p v_{\min} \min_i \left(i_{\max,k}^{\text{dis,volt}(i)} \right).$$

Similarly, to compute cell maximum absolute charge current, we set $R_k^{(i)} = R_{\text{chg},\Delta T}^{(i)}$ and clamp the future voltage $v_{k+k_{\Delta T}}^{(i)} = v_{\max}$. Then,

$$i_{\min,k}^{\text{chg,volt}(i)} = \frac{\text{OCV}(z_k^{(i)}) - v_{\max}}{R_{\text{chg},\Delta T}^{(i)}}.$$

Note that charging current is nonpositive and therefore the maximum absolute charge current is the minimum charge current in a signed sense. Pack charge power based only on observing voltage limits is then calculated as

$$p_{\min,k}^{\text{chg,volt}} = N_s N_p v_{\max} \max_i \left(i_{\min,k}^{\text{chg,volt}(i)} \right).$$

6.2.1 Rate limits based on SOC, maximum current, and power

We can extend the basic HPPC method quite easily to include SOC-based limits with a time horizon ΔT in addition to the voltage-based limits. For a constant current i_k, the SOC recurrent relationship is:

$$\begin{aligned} z_{k+k_{\Delta T}}^{(i)} &= z_k^{(i)} - \frac{\eta_k^{(i)} k_{\Delta T} \Delta t}{Q^{(i)}} i_k \\ &= z_k^{(i)} - \frac{\eta_k^{(i)} \Delta T}{Q^{(i)}} i_k. \end{aligned}$$

We will assume that $\eta_k = 1$ for discharge and that $\eta_k = \eta \leq 1$ for charge currents.

If we have design limits such that $z_{\min} \leq z_k^{(i)} \leq z_{\max}$ for all cells in the pack, we can compute current i_k to enforce these limits. Simple algebra gives

$$i_{\max,k}^{\text{dis,soc}(i)} = \frac{z_k^{(i)} - z_{\min}}{\Delta T / Q^{(i)}} \tag{6.3}$$

$$i_{\min,k}^{\text{chg,soc}(i)} = \frac{z_k^{(i)} - z_{\max}}{\eta \Delta T / Q^{(i)}}, \tag{6.4}$$

by replacing future SOC $z_{k+k_{\Delta T}}^{(i)}$ with either $z_{\min}$ or $z_{\max}$, as appropriate.

We understand very well by now that cell SOC is never known exactly. However, if we use a nonlinear Kalman filter to estimate the SOCs of the battery-pack cells, we additionally receive confidence intervals on the estimates. These confidence intervals can be used to make power estimates more conservative. For example, if we are comfortable assuming that a $3\sigma_z$ confidence interval on the SOC estimate will almost certainly contain the true SOC, then we can revise the discharge and charge current limits to be

$$i_{\max,k}^{\text{dis,soc}(i)} = \frac{\left(z_k^{(i)} - 3\sigma_{z,k}^{(i)}\right) - z_{\min}}{\Delta T / Q^{(i)}} \tag{6.5}$$

18:59:59.

$$i^{\text{chg,soc}(i)}_{\text{min},k} = \frac{\left(z^{(i)}_k + 3\sigma^{(i)}_{z,k}\right) - z_{\max}}{\eta \Delta T / Q^{(i)}}. \tag{6.6}$$

Once all cell current limits have been calculated, the pack discharge and charge currents that satisfy all design constraints are computed as:

$$i^{\text{dis}}_{\max,k} = \min\left(i_{\max}, \min_i i^{\text{dis,soc}(i)}_{\max,k}, \min_i i^{\text{dis,volt}(i)}_{\max,k}\right) \tag{6.7}$$

$$i^{\text{chg}}_{\min,k} = \max\left(i_{\min}, \max_i i^{\text{chg,soc}(i)}_{\min,k}, \max_i i^{\text{chg,volt}(i)}_{\min,k}\right). \tag{6.8}$$

Battery-pack power may be calculated using the sum of all cell powers, where cell powers are computed as the product of the maximum allowed cell current and predicted future voltage:

$$\begin{aligned}
p^{\text{chg}}_{\min,k} &= N_p \max\left(N_s p_{\min}, \sum_{i=1}^{N_s} i^{\text{chg}}_{\min,k} v^{(i)}_{k+k_{\Delta T}}\right) \\
&\approx N_p \max\left(N_s p_{\min}, \sum_{i=1}^{N_s} i^{\text{chg}}_{\min,k}\left(\text{OCV}(z^{(i)}_k) - i^{\text{chg}}_{\min,k} R^{(i)}_{\text{chg},\Delta T}\right)\right); \\
p^{\text{dis}}_{\max,k} &= N_p \min\left(N_s p_{\max}, \sum_{i=1}^{N_s} i^{\text{dis}}_{\max,k} v^{(i)}_{k+k_{\Delta T}}\right) \\
&\approx N_p \min\left(N_s p_{\max}, \sum_{i=1}^{N_s} i^{\text{dis}}_{\max,k}\left(\text{OCV}(z^{(i)}_k) - i^{\text{dis}}_{\max,k} R^{(i)}_{\text{dis},\Delta T}\right)\right).
\end{aligned}$$

6.3 *Voltage-based power limits, using a full cell model*

This enhanced version of the HPPC method is still limited. First, the cell model it uses is too primitive to give precise results. Overly optimistic or pessimistic predictions could be generated, either posing a safety or battery-health hazard, or being inefficient in battery use. Second, the equations assume that the cell is in an equilibrium condition prior to the application of a current pulse, which is not true in general. To compensate for the uncertainty introduced by these limitations, the HPPC power estimate is usually derated by multiplying its value by a factor less than one. This makes the final estimate somewhat more conservative. However, a maximum-power algorithm that uses a better cell model can give better power-limit estimates.

We now assume a more accurate model of cell dynamics in a discrete-time state-space form (such as the enhanced self-correcting cell model used throughout this book):[3]

$$\begin{aligned}
\boldsymbol{x}^{(i)}_{k+1} &= \boldsymbol{f}\big(\boldsymbol{x}^{(i)}_k, i_k\big) \\
v^{(i)}_k &= h\big(\boldsymbol{x}^{(i)}_k, i_k\big).
\end{aligned}$$

[3] Much of the remainder of this chapter is based on: Plett, G.L., "High-Performance Battery-Pack Power Estimation Using a Dynamic Cell Model," *IEEE Transactions on Vehicular Technology*, 53(5), 2004, pp. 1,586–93.

Then, we can use this model to predict cell voltage ΔT seconds into the future by

$$v_{k+k_{\Delta T}}^{(i)} = h\big(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_{k+k_{\Delta T}}\big),$$

where $\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}$ may be found by simulating the state equation open-loop into the future for $k_{\Delta T}$ time samples.

We assume that the input current to each cell remains constant from time index k to $k + k_{\Delta T}$ and denote it simply as i_k. We then use a bisection search algorithm—to be explained in Sect. 6.4—to find $i_{\max,k}^{\text{dis,volt}(i)}$ and $i_{\min,k}^{\text{chg,volt}(i)}$ by looking for the i_k that causes equality in

$$\begin{aligned} v_{\min} &= h\big(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_k\big), \quad \text{or} \\ 0 &= h\big(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_k\big) - v_{\min} \end{aligned} \tag{6.9}$$

to find $i_{\max,k}^{\text{dis,volt}(i)}$, and by looking for the i_k that causes equality in

$$\begin{aligned} v_{\max} &= h\big(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_k\big), \quad \text{or} \\ 0 &= h\big(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_k\big) - v_{\max} \end{aligned} \tag{6.10}$$

to find $i_{\min,k}^{\text{chg,volt}(i)}$. Once again, SOC-based current limits $i_{\max,k}^{\text{dis,soc}(i)}$ and $i_{\min,k}^{\text{chg,soc}(i)}$ are computed using Eqs. (6.3) and (6.4), respectively, if SOC is known exactly, or using Eqs. (6.5) and (6.6), respectively, if SOC is being estimated using a nonlinear Kalman filter.

The bisection algorithm must compute future predicted voltage repeatedly for different candidate input-current levels. This calculation can be simplified when the state equation is linear; that is, when

$$\boldsymbol{x}_{k+1}^{(i)} = \boldsymbol{A}\boldsymbol{x}_k^{(i)} + \boldsymbol{B}i_k,$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are constant matrices. This condition is true for the ESC cell model when the input current is constant over the entire prediction horizon. Then, we have

$$\boldsymbol{x}_{k+k_{\Delta T}}^{(i)} = \boldsymbol{A}^{k_{\Delta T}}\boldsymbol{x}_k + \underbrace{\left(\sum_{j=0}^{k_{\Delta T}-1} \boldsymbol{A}^{k_{\Delta T}-1-j}\right)}_{\boldsymbol{A}_{k_{\Delta T}}} \boldsymbol{B}i_k. \tag{6.11}$$

This formulation can be used to speed calculation of future state and future voltage within the steps required by the bisection algorithm.

Once the cell current limits are established, overall battery-pack discharge and charge current limits are computed using Eqs. (6.7) and (6.8). Charge power is then computed as

$$p_{\min,k}^{\text{chg}} = N_p \max\left(N_s p_{\min}, \sum_{i=1}^{N_s} i_{\min,k}^{\text{chg}} v_{k+k_{\Delta T}}^{(i)}\right)$$

$$= N_p \max\left(N_s p_{\min}, \sum_{i=1}^{N_s} i_{\min,k}^{\text{chg}} h(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_{\min,k}^{\text{chg}})\right),$$

and discharge power is computed as

$$\begin{aligned} p_{\max,k}^{\text{dis}} &= N_p \min\left(N_s p_{\max}, \sum_{i=1}^{N_s} i_{\max,k}^{\text{dis}} v_{k+k_{\Delta T}}^{(i)}\right) \\ &= N_p \min\left(N_s p_{\max}, \sum_{i=1}^{N_s} i_{\max,k}^{\text{dis}} h(\boldsymbol{x}_{k+k_{\Delta T}}^{(i)}, i_{\max,k}^{\text{dis}})\right). \end{aligned}$$

To use a full equivalent-circuit cell model, then, all that remains is to see how to determine i_k to meet future cell-voltage limits. We look at this next.

6.4 *Bisection search*

To compute dynamic power limits for a battery cell using a comprehensive cell model, we must be able to solve Eq. (6.9) for i_k leading to $i_{\max,k}^{\text{dis,volt}(i)}$, and Eq. (6.10) for i_k leading to $i_{\min,k}^{\text{chg,volt}(i)}$. To do so, we require a method to find a root of a nonlinear equation. Here, we use the bisection search algorithm to do so.

Generically, the bisection search algorithm looks for a root of some function $g(x)$ (i.e, a value of x such that $g(x) = 0$) where it is known a priori that at least one root lies in the interval $x_1 < \text{root} < x_2$. One way of knowing that this condition is met is that the sign of $g(x_1)$ is different from the sign of $g(x_2)$. The bisection algorithm repeatedly shrinks the interval between x_1 and x_2 but maintains the sign difference to ensure that a root always remains between the endpoints.

Fig. 6.4 shows an example nonlinear function having a root between x_1 and x_2 to illustrate the bisection algorithm. In this particular example, $g(x_2) > 0$ and $g(x_1) < 0$, but these signs could be reversed and the method would still work.

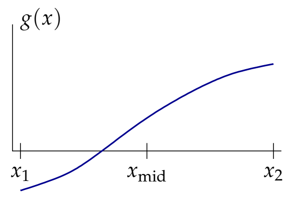


Figure 6.4: An interval containing a single zero crossing of function $g(x)$.

The first step bisection algorithm is to evaluate the function $g(\cdot)$ at the midpoint $x_{\text{mid}} = (x_1 + x_2)/2$. Based on the sign of the evaluation, either x_1 or x_2 is replaced by x_{mid} to retain different signs on $g(x_1)$ and $g(x_2)$. We see that the interval containing the root is halved in width by this algorithmic step. This bisection iteration is repeated until the interval between x_1 and x_2, (i.e., the resolution of the root of $g(x)$) is as small as desired. If ε is the desired root resolution, the algorithm will require at most

$$\text{maximum number of iterations} = \lceil \log_2 (|x_2 - x_1|/\varepsilon) \rceil \tag{6.12}$$

iterations, where $\lceil \cdot \rceil$ rounds its argument to the next highest integer.

18:59:59.

A MATLAB implementation of the bisection method is listed below. The inputs to the function are a function handle `g`, the range limits `x1` and `x2` within which to search for a root, and the required resolution `res` of the identified root:

```
% Search interval x1...x2 in fn g(.) for a root, with resolution res
function x = bisect(g,x1,x2,res)
  maxIter = ceil(log2(abs(x2-x1)/res));

  dx = x2 - x1; % set the search interval dx = x2 - x1
  if( g(x1) >= 0 )
    dx = -dx; x1 = x2;    % root now b/w (x1,x1 + dx), and g(x1) < 0
  end

  for theIter = 1:maxIter-1
    dx = 0.5 * dx; xmid = x1 + dx;

    if g(xmid) <= 0,
      x1 = xmid;
    elseif abs(dx) <= res,
      break
    end
  end
  x = x1 + 0.5*dx; % the final bisection
end
```

The function first computes the maximum number of bisection steps `maxIter` that will be required, using Eq. (6.12). It then computes the present width of the root-uncertainty window, `dx`. The limits `x1` and `x2` are examined, and interchanged if necessary to ensure that the root lies between x_1 and $x_1 + \texttt{dx}$, where the constraint $g(x_1) < 0$ is satisfied. The function enters a loop for a maximum of `maxIter-1` iterations where the interval width is repeatedly halved and `x1` is updated if necessary to maintain the constraints that the root lies between x_1 and $x_1 + \texttt{dx}$ and $g(x_1) < 0$. Before returning, knowing that the root lies within the final values of x_1 and $x_1 + \texttt{dx}$, the function computes its final estimate as the midpoint of this interval. The final root-location error is then within $\pm\texttt{dx}/2$.

As an example of how to use the `bisect.m` function, consider the following code segment:

```
g = @(x) x^3;
bisect(g,-1,2,1e-5)
```

First, we create an *anonymous function* named `g`. This function has input argument `x`, as denoted by the `@(x)` syntax, and computes the cube of its input argument as its return value. Anonymous functions are convenient when we wish to implement a simple computation without wishing to store the computational steps in a program file (however, as we will see later, anonymous functions are not required when using `bisect.m`). The variable `g` has data type `function_handle` in MATLAB. Then, `bisect.m` is invoked with the anonymous function `g` as its first input argument, for $x_1 = -1$, $x_2 = 2$, and the desired root resolution being 10^{-5}. The output root estimate is 1.9073×10^{-6}, which is closer than 10^{-5} to the true root location of 0.

When computing cell power limits, bisection is incorporated in the overall algorithm as follows:

- First, three simulations are performed to predict what every cell's voltage would be $k_{\Delta T}$ samples into the future for constant-current inputs $i_k = 0$, $i_k = i_{\max}$, and $i_k = i_{\max}$.
- If cell voltages are predicted to be between $v_{\min}$ and $v_{\max}$ for the maximum absolute rates $i_{\min}$ and $i_{\max}$, then these rates may be used:
  $$i_{\max,k}^{\text{dis,volt}(i)} = i_{\max} \qquad \text{and} \qquad i_{\min,k}^{\text{chg,volt}(i)} = i_{\min}.$$
  Bisection is not needed.
- If the cell voltages, even during rest, are outside of the range $v_{\min}$ to $v_{\max}$, then the battery pack is in an unsafe condition. In this case, we might decide to set the maximum rates to zero. The default then is
  $$i_{\max,k}^{\text{dis,volt}(i)} = 0 \qquad \text{and} \qquad i_{\min,k}^{\text{chg,volt}(i)} = 0.$$
  However, these defaults may be overridden by values that would bring the pack back inside valid limits. For example, we may set $i_{\max,k}^{\text{dis,volt}(i)} > 0$ on an overcharged battery cell or $i_{\max,k}^{\text{chg,volt}(i)} < 0$ for an overdischarged battery cell. The current limits in this case could be solved via bisection as in the next case.
- If the voltage is outside of limits when a maximum current is simulated, but is within limits during rest, we know that the true maximum rate may be found by bisecting between rate equal to zero and its maximum value. So, bisection is performed between current limits $(i_{\min}, 0)$ or $(0, i_{\max})$.

TO ESTIMATE POWER LIMITS using bisection and an ESC cell model, we need to define a bisection objective function, which will itself involve a state and voltage prediction calculation $k_{\Delta T}$ seconds into the future. For constant input current, the ESC cell-model state equation is linear, with

$$\boldsymbol{x}_{k+1}^{(i)} = \boldsymbol{A}\boldsymbol{x}_k^{(i)} + \boldsymbol{B}i_k.$$

To incorporate this in a bisection search, we first define anonymous matrix functions to compute state-space $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices based on cell input current:

```
A = @(ik) diag([1 exp(-1/(RC)) exp(-abs(ik*Gamma/(3600*Q)))]);
B = @(ik) [-1/(3600*Q) 0; (1-exp(-1/RC)) 0; ...
          0 (1-exp(-abs(ik*Gamma/(3600*Q))))];
```

In this code, we assume that SOC is the top element of vector $x_k^{(i)}$, a single diffusion-current state is in the middle of the vector, and that the hysteresis state is last. The input to each of these anonymous functions is the cell input current (needed in the hysteresis element computations) and the input to the overall cell model is assumed to be a 2-vector comprising the input current as the top element and the sign of the input current as the bottom element. Therefore $\boldsymbol{A}$ is 3×3 and $\boldsymbol{B}$ is 3×2 in dimension.

We take advantage of linearity inside the bisection algorithm via Eq. (6.11). Because $\boldsymbol{A}$ is diagonal, the matrix power $\boldsymbol{A}^{k_{\Delta T}}$ is simply the diagonal matrix comprising the scalar powers of the diagonal elements. Similarly, the summation can be written as

$$\begin{aligned}\sum_{j=0}^{k_{\Delta T}-1} \boldsymbol{A}^{k_{\Delta T}-1-j} &= \left(\sum_{j=0}^{k_{\Delta T}-1} \boldsymbol{A}^{-j}\right) \boldsymbol{A}^{k_{\Delta T}-1} = \left(\sum_{j=0}^{k_{\Delta T}-1} \left(\boldsymbol{A}^{-1}\right)^{j}\right) \boldsymbol{A}^{k_{\Delta T}-1} \\ &= \left(\boldsymbol{I} - \boldsymbol{A}^{-1}\right)^{-1} \left(\boldsymbol{I} - \boldsymbol{A}^{-k_{\Delta T}}\right) \boldsymbol{A}^{k_{\Delta T}-1} \\ &= \left(\boldsymbol{I} - \boldsymbol{A}^{-1}\right)^{-1} \left(\boldsymbol{A}^{k_{\Delta T}-1} - \boldsymbol{A}^{-1}\right) \\ &= (\boldsymbol{A} - \boldsymbol{I})^{-1} \left(\boldsymbol{A}^{k_{\Delta T}} - \boldsymbol{I}\right), \end{aligned} \tag{6.13}$$

if we assume that all diagonal elements of $\boldsymbol{A}$ are strictly less than 1 in magnitude. For diagonal elements of $\boldsymbol{A}$ that are equal to 1, we instead have

$$\sum_{j=0}^{k_{\Delta T}-1} \boldsymbol{A}^{k_{\Delta T}-1-j} = k_{\Delta T}. \tag{6.14}$$

This allows us to write very efficient code to simulate a cell's state and voltage $k_{\Delta T}$ samples into the future. Consider the following function, `simCellKDT.m`:

```
% Simulate cell for KDT samples, with input current equal to ik, initial
% state = x0, A and B functions, temperature = T, with model parameters
% R0, R, M and the model structure "model".
function [vDT,xDT] = simCellKDT(ik,x0,A,B,KDT,T,model,R0,R,M)
  Amat = A(ik); Bmat = B(ik); dA = diag(Amat);
  if ik == 0,
    ADT = diag([KDT, (1-dA(2)^KDT)/(1-dA(2)), KDT]);
  else
    ADT = diag([KDT, (1-dA(2)^KDT)/(1-dA(2)), (1-dA(3)^KDT)/(1-dA(3))]);
  end
  xDT = (dA).^KDT.*x0 + ADT*Bmat*[ik; sign(ik)];
  vDT = OCVfromSOCtemp(xDT(1),T,model) - R*xDT(2) + M*xDT(3) - ik*R0;
end
```

The function first evaluates the anonymous $\boldsymbol{A}$-matrix and $\boldsymbol{B}$-matrix functions for the proposed input-current level `ik` to compute `Amat` and `Bmat`, respectively. The matrix $\boldsymbol{A}_{k_{\Delta T}}$ of Eq. (6.11) is then computed as `ADT` using either Eq. (6.13) or (6.14), depending on whether the corresponding entries of A are 1 or less than 1. (The SOC entry is always 1 and the hysteresis-state entry is 1 when input current is zero.) Finally, the future state is computed using Eq. (6.11) and the future voltage is computed from the future state.

At this point, we have defined a bisection function and a function that can predict future cell state and voltage. To complete the power-estimation code, we require functions that connect the two.

When computing the discharge-power limit while considering both terminal-voltage and SOC design constraints, we can use the following code:

```
function g = bisectDischarge(ik,x0,A,B,KDT,T,model,R0,R,M)
  [vDT,xDT] = simCellKDT(ik,x0,A,B,KDT,T,model,R0,R,M);
  g = max(vmin - vDT,zmin - xDT(1)); % max must be less than zero
end
```

The function first simulates the cell for $k_{\Delta T}$ samples to predict future state and voltage for the proposed level of input current `ik`. It then computes a result `g` that can be bisected on, where MATLAB variable `vmin` stores the value of $v_{\min}$ and `zmin` stores the value of $z_{\min}$. Notice that `vmin - vDT` must be negative for an acceptable future voltage `vDT` and that `zmin - xDT(1)` must be negative for an acceptable future SOC `xDT(1)`. Therefore, the maximum of the two must also be less than zero to enforce both conditions simultaneously. If we bisect over a region x_1 to x_2 to find the location where $g(x) = 0$, we are solving for the point where either the voltage or the SOC limit has been attained (we don't know which from this function, but that information is not needed).

When computing the charge-power limit while considering both terminal-voltage and SOC design constraints, we can use the following function:

```
function g = bisectCharge(ik,x0,A,B,KDT,T,model,R0,R,M)
  [vDT,xDT] = simCellKDT(ik,x0,A,B,KDT,T,model,R0,R,M);
  g = min(vmax - vDT,zmax - xDT(1)); % min must be greater than zero
end
```

Its operation is very similar to `bisectDischarge.m`, except that both `vmax - vDT` and `zmax - xDT(1)` must be positive for acceptable future values of voltage and SOC. Therefore, by constraining their minimum to be positive, we simultaneously satisfy both conditions.

To bisect to find maximum discharge current, we use code like:

```
gDis = @(x) bisectDischarge(x,x0,A,B,KDT,T,model,R0,R,M)
ilimit = bisect(gDis,0,imax,ires);
```

where `ires` is the desired resolution for the solution for battery-pack current. To bisect to find maximum absolute charge current, we use code like:

```
gChg = @(x) bisectCharge(x,x0,A,B,KDT,T,model,R0,R,M)
ilimit = bisect(gChg,imin,0,ires);
```

Power-limits estimation example

We close this chapter with a simulation example showing the similarities and differences between the HPPC method based on a simplified cell model and the bisection method based on the comprehensive ESC cell model. The data used in the example were collected from a lithium-ion cell that was subjected to a sequence of 16 UDDS drive-cycle profiles, separated by discharge pulses and five-minute rests. The overall profile of cell SOC versus time is plotted in Fig. 6.5. SOCs increases by about 5 % during each UDDS drive-cycle profile, but is brought down about 10 % during the discharge between profiles. The entire design operating range for this cell (10 % SOC to 90 % SOC, delineated on the figure as the region between the thin dashed lines) is excited during the cell test.

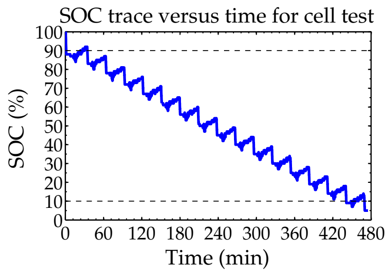


Figure 6.5: SOCs versus time for cell test used in example.
© 2004 IEEE. Reprinted, with permission, from Fig. 5 in Plett, G.L., "High-Performance Battery-Pack Power Estimation Using a Dynamic Cell Model," *IEEE Transactions on Vehicular Technology*, 53(5), 2004, pp. 1,586–93.

Fig. 6.6 shows a comparison between measured cell voltage for this test and the voltage that is predicted by an ESC cell model that was fit to the dynamics of this particular cell. The difference between true and estimated cell terminal voltages is very small (root-mean-squared voltage estimation error is less than 5 mV). The top frame shows data collected over the entire test; the lower frame shows a zoom on one cycle to illustrate the quality of the model's estimates. In the discussion that follows, we will consider the power predictions produced by the bisection method to represent the true capability of the cell, justifying this assumption by the fidelity of the cell model's voltage estimates, as supported by the plots in Fig. 6.6.

For the following results, we assume that the battery pack has $N_s = 40$ and $N_p = 1$. Cells have nominal capacity of 7.5 Ah and $\Delta T = 10$ s for both charge and discharge. Operational limits for voltage, current, SOC, and power used by the power-limits calculations are listed in Table 6.1. The resolution on current used in the bisection algorithm was 0.1 A.

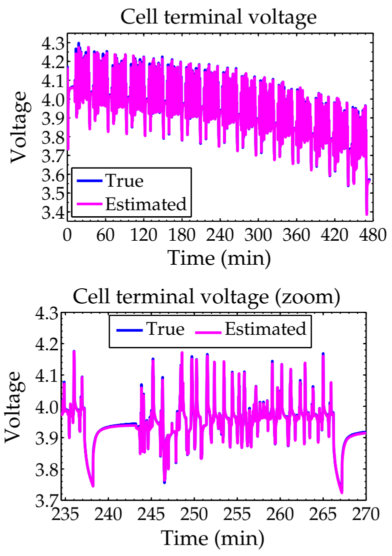


Figure 6.6: Comparison between measured and modeled cell voltage.
© 2004 IEEE. Reprinted, with permission, from Figs. 3 and 4 in Plett, G.L., "High-Performance Battery-Pack Power Estimation Using a Dynamic Cell Model," *IEEE Transactions on Vehicular Technology*, 53(5), 2004, pp. 1,586–93.

Parameter	Minimum	Maximum
Voltage	$v_{\min} = 3.0$ V	$v_{\max} = 4.35$ V
Current	$i_{\min} = -200$ A	$i_{\max} = 200$ A
SOC	$z_{\min} = 0.1$	$z_{\max} = 0.9$
Power	$p_{\min} = -\infty$	$p_{\max} = \infty$

Table 6.1: Design limits for power-limit estimation example.

18:59:59.

Fig. 6.7 compares discharge power estimates produced by the HPPC and bisection methods. The upper frame shows results for the entire test, and the lower frame shows a zoom into the results for one cycle. The overall discharge power is limited throughout most of the test by the requirement to keep cell voltage above v_{min}; however, after about 400 min, both methods are also limited at times by low cell SOC approaching z_{min}.

Overall, we see that the two methods produce generally similar estimates. At high SOCs, the HPPC method predicts higher power than is actually available (by as much as 9.8 %), and at mid to low SOCs, the HPPC method underpredicts the available power. If the load controller were to discharge at the rates calculated by the HPPC method, then the cell would be overdischarged in some cases (lowering its lifetime) and under-utilized in other cases.

The zoom in the mid-SOC region shows greater detail. In this region, the methods produce nearly identical predictions. A notable feature of the bisection method; however, is that it takes into account the entire dynamics of the cell when making a prediction. The strong discharges around times 237 and 267 minutes draw down the cell's diffusion voltages and hence also its terminal voltage. Consequently, so less discharge power is available from the cell than the HPPC method predicts, because the HPPC method considers the cell to be in an equilibrium state when making its estimate.

The two methods are also compared with respect to charge power, as shown in Fig. 6.8. The upper plot shows the entire dataset and the lower plot shows a zoom into one region. The plots show predictions of absolute charge power (charge power itself is computed as a negative value).

Again, at this scale, the estimates appear to be nearly identical over most of the test. The HPPC method sometimes overpredicts charge power at high SOCs. It also overpredicts power at low SOCs as it uses a fixed charge resistance that ignores the increase in charge resistance in that region.

The zoom in the lower frame of the figure illustrates better the differences between the predictions. Here, we see that the strong discharges around times 237 and 267 minutes polarize the cell by drawing cell diffusion voltages down, thus allowing for greater short-term charging power without violating terminal-voltage limits. This effect is not captured by the HPPC method.

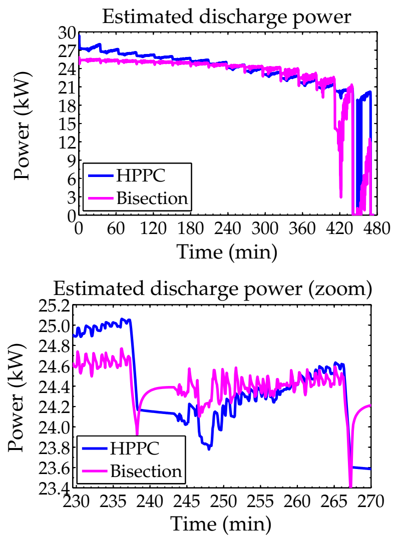


Figure 6.7: Comparison between predictions of discharge power.

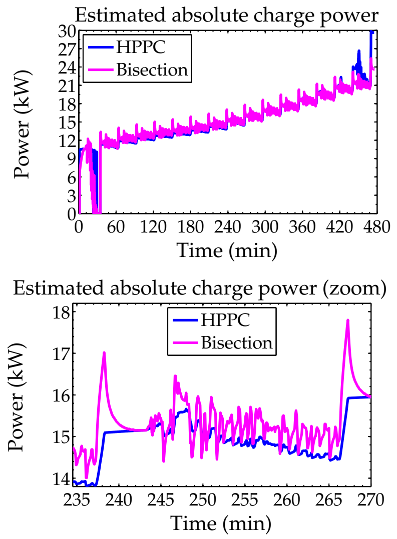


Figure 6.8: Comparison between predictions of charge power.

6.5 *Where to from here?*

In this chapter we have presented two methods that can be used to predict battery discharge and charge power. Both incorporate voltage,

18:59:59.

SOC, power, and current design constraints, and work for a user-specified prediction horizon ΔT s. The results indicate that the two methods produce similar results.

The bisection method requires a good cell model and significantly more computation than the HPPC method. But, if a Kalman filter is being used to estimate cell SOC, then the cell model will already be present and an estimate of the state and its confidence interval will be available for use. The bisection method produces dynamic power estimates and is able to take advantage of recent strong discharges to allow higher absolute short-term charge power, and is able to take advantage of recent strong charges to allow higher short-term discharge power.

The implicit assumption in this chapter is that power should be computed to enforce voltage limits on a cell. This is not the essential issue, however. Really, we desire to minimize the incremental degradation that is being experienced by the cell. For example, even in present BMSs, we usually derate the power limits at warm temperatures to slow down temperature rise.[4] But, this single change hardly begins to address a detailed optimized strategy regarding how to compute power limits based on aging.

[4] This is done to slow down aging, even though the cell is perfectly capable of delivering higher power at warm temperatures than at cold temperatures as its internal resistance decreases with higher temperatures.

To understand better how to compute power limits to optimize a tradeoff between performance and life, we need to discuss how cells age and then investigate some more advanced power-estimation algorithms that take advantage of these aging models. This is the topic of the next and final chapter.

7
Physics-Based Optimal Controls

As we approach the end of this book, we also approach the frontier of knowledge in battery-management methods that use equivalent-circuit models of battery cells. We have seen that the electronics aspects involved in the design of a BMS are important but routine. Cell SOC is well defined, and established methods can be used to compute accurate estimates of state of charge and to provide accompanying confidence intervals on the estimates. Similarly, we have seen good methods to estimate cell resistance and total capacity, yielding SOH estimates. Cell energy calculation is straightforward and several types of cell balancing—with varying levels of complexity and speed—can be implemented.

Improvements can still be made to all of the above, but the present state of the art provides adequate battery management for many applications. There remain some questions regarding the long-term efficacy of existing methods on aged battery packs, simply because there are few examples of high-capacity lithium-ion battery packs that have been fielded for a decade or more at this point. The most significant gains that can be made in battery-management methods are to how power-limit calculations are done, and some of these new ideas will be introduced in this chapter.

Davide Andrea, a pioneer in lithium-ion battery management and author of a well-respected text on the subject,[1] has said: "Using current electronics and knowledge it takes about two years and $250K to build a custom battery management system.[2]" It is not a trivial task, but it is very doable.

[1] Andrea, D., *Battery Management Systems for Large Lithium Ion Battery Packs*, Artech House, 2010.

[2] The quotation is from a seminar he gave to the Colorado Cleantech Fellows in November, 2012.

7.1 *Minimizing degradation*

Referring back to the roadmap Fig. 3.1, which monitors our progress through the material in this book, it might seem that we are finished. A BMS that incorporates all of the methods that we have studied will work very well; however, it will also be overdesigned in many cases.

The problem is with how power limits are presently calculated. The fundamental requirement imposed when computing these limits is that certain design maximum and minimum voltage limits must never be violated by any cell in the battery pack.

But why? The real concern is cell degradation.[3] The underlying assumptions are that:

- If voltage limits are violated, then the cell will degrade quickly and fail prematurely;
- If voltage limits are properly maintained, the cell will have a long and productive life.

But, in fact, voltage limits *may* be violated for short periods of time in some situations without causing rapid aging. Consider, for example, a cell that has been rated for operation between 3.0 V and 4.2 V. These ratings are defined by the manufacturer of the cell, and the battery applications designer is told that the cell should never be operated outside of this range. The curious BMS design engineer wants to know why. Is 4.199 V really all that different from 4.201 V? Why is one permitted and the other not permitted?

If we were to ask the cell's designer if it is okay to let the cell voltage sometimes reach 4.25 V, the answer would be, "Yes." Well, how about 4.3 V? The cell designer will begin to look worried, but will say, "Yes, for short periods of time." How about 4.35 V? "Yes, but for *very* short periods of time." At some point in this interchange, the cell designer will stop committing to an answer.

We learn something very important from this discussion. We discover that an exact limit on cell terminal voltage is not itself the criterion that the cell designer is concerned about. The manufacturer-supplied voltage limits are instead indirect indicators of something else. The actual causes of aging are electrochemical and mechanical processes occurring inside of the cell, and cell terminal voltage is a poor but measurable indicator of the state of these processes.

Statistically speaking, based on many empirical cell tests, the manufacturer has determined that the cell will give a reasonable tradeoff between performance and life if its voltage does not exceed 4.2 V. However, under certain conditions it is perfectly fine to exceed this value; under other conditions, unacceptably fast aging may occur even if this value is maintained, especially for older cells whose internal dynamics have changed.

So, the real issue when computing cell power limits is not the cell's terminal voltage but rather the amount of aging or degradation that is expected to occur at different power levels. Cell power limits *should* really be calculated to optimize more directly a tradeoff be-

[3] Maintaining safety is the primary concern of a BMS, as noted at the beginning of Chap. 1. However, cell terminal voltage alone is neither the fundamental cause nor a good indicator of most safety hazards. Rather, hazards that are dormant during normal cell operation can become serious risks when certain cell degradation mechanisms are triggered. For example, overcharging a lithium-ion cell can cause lithium plating, which forms lithium dendrites that can short circuit the cell and lead to thermal runaway. Overcharging a lead-acid cell can lead to a buildup of explosive gasses. Both of these mechanisms are caused by excessive potentials at locations internal to the battery cell rather than at the cell terminals, so cell terminal voltage is only approximately an indicator of a problem. If we model the degradation mechanisms at critical locations inside the cell and monitor and control the cell to prevent them, as proposed, then we will also avoid the corresponding safety hazards.

tween performance delivered by the cell and the rate of incremental degradation experienced by the cell.

To be able to do this, we must be able to model degradation mathematically and devise model-based optimized controls to calculate the best tradeoff. Some have suggested that if this is done correctly, battery packs in some applications may be able to deliver more than 200 % of their voltage-limited power without compromising life.[4] Alternately, the battery pack could be reduced significantly in size and still deliver required performance. These possible cost savings and/or performance gains are ample incentives to make a strong attempt to design physics-based power limits instead of voltage-based power limits.

[4] See, for example, Smith, K.A. and Wang, C.-Y., "Power and thermal characterization of a lithium-ion battery pack for hybrid-electric vehicles," *Journal of Power Sources*, 160, 2006, pp. 662–673.

7.1.1 Modeling cell degradation

In Chap. 4, we saw that much is known *qualitatively* about how lithium-ion battery cells degrade. But, how about *quantitatively*? That is, can we make accurate parameterized mathematical models for all of the degradation mechanisms?

We have seen that the interactions between ideal-cell behaviors and the different degradation mechanisms are complex. Further, they are not presently well understood. So, at this point, we don't know whether we can make models of all mechanisms. Also, most cell manufacturers include *additive packages* of chemicals that are part of the cell construction that do not take part in ideal-cell operations but instead are designed to impede degradation. These additives are trade secrets and so are not disclosed to the BMS designer. This makes precise modeling of the rates of degradation very difficult.

However, a factor working in our favor is that we don't need to model *all* mechanisms *perfectly* to have a useful result. Further, for purposes of cell control via physics-based power-limit calculations, we don't need to model any mechanism that is not influenced by a variable that we have influence over. If we model the most severe degradation mechanisms reasonably well, then we have a chance at designing controls that make a difference.

The literature on quantitative modeling of degradation-mechanism modeling is quite sparse at this point. Over time, we expect that this will improve as more electrochemical modelers become aware of the need for such models. In this chapter, we will discuss two quantitative physics-based degradation models that can be used alongside equivalent-circuit models as part of a power-limits calculation. Other models require knowledge of electrochemical variables that could be generated by a reduced-order physics-based model of ideal-cell dynamics, and will be investigated in the planned Volume III of this series.

Here, we look first at a model of solid–electrolyte interphase formation and growth. This degradation mechanism is considered to be the most significant life-limiting mechanism in cells having graphite negative electrodes. So, if we can impose cell controls that slow down SEI growth, then we can extend life.

The second model we look at is of lithium plating on overcharge. This mechanism is one that would not normally occur if we observed the manufacturer-specified upper voltage limit on the cell, but which can reduce a cell to inoperability in a few cycles if we are not careful. Because we are now ignoring this uppervoltage limit, we must model lithium plating and take it into account when computing cell maximum power limits.

7.2 SEI formation and growth

Ramadass and colleagues have proposed a model that describes the formation and growth of an SEI layer on negative-electrode solid particles during charging.[5] This model assumes that solvent reduction at the surface of the particle is the main side-reaction mechanism for degradation.

[5] Ramadass, P., Haran, B., Gomadam, P.M., White, R.E., and Popov, B.N., "Development of First Principles Capacity Fade Model for Li-Ion Cells," *Journal of the Electrochemical Society*, 151(2), 2004, pp. A196–A203.

Here, we summarize the main points from the Ramadass model and build on that work to develop a simple recursive discrete-time model of SEI growth and associated capacity loss and resistance rise.[6] The order-reduction method uses volume averaging to create an algebraic zero-dimensional (0D) model of the infinite-order PDE model. This *reduced-order model (ROM)* of the SEI growth mechanism is a first step toward creating a complete coupled ROM of all dominant cell degradation mechanisms, which could then be used in an optimal control scheme.

[6] Adapted from, Randall, A.V., Perkins, R.D., Zhou, X., Plett, G.L., "Controls Oriented Reduced Order Modeling of SEI Layer Growth," *Journal of Power Sources*, 209, 2012, pp. 282–288.

7.2.1 Full-order model (FOM)

Solid–electrolyte interphase side reactions in the negative electrode are considered to be one of the primary causes of cell aging in lithium-ion cells having graphitic negative electrodes. There are numerous reduction reactions that can lead to the deposition of solid SEI products on the electrode surface and the details are not well understood, being very dependent upon the composition of the electrolyte solution. Ramadass et al. make the general assumption that the side reactions consume lithium ions and solvent from the electrolyte and form a surface film from compounds such as Li_2CO_3, LiF, Li_2O, and so forth, depending on the nature of the solvent.

There is significant porosity to the SEI film. Thus, it is still possible for lithium to enter and exit the particle through the film, although it does add an ionic resistivity R_{film} that increases cell resistance as well. Moreover, while the film makes it more difficult for solvent to reach the particle surface and create more SEI, the porosity is sufficient for some solvent to penetrate and so the SEI layer continues to grow slowly as solvent diffuses through the layer during charge. Further, the intercalation of lithium into the graphite negative electrode leads to an increase in the lattice volume, which in turn stretches the SEI layer and causes it to fracture and to expose more of the active material to the electrolyte, fueling the side reaction and contributing to SEI growth.

The model proposed by Ramadass et al. does not attempt to describe all of these individual mechanisms in detail. Rather, it homogenizes all of the effects into a simplified description. The main assumptions made when creating the model were:[7]

[7] The concepts needed to define the FOMs and ROMs of SEI growth depend heavily on the theoretical topics taught in Vol. I of this series. However, the resulting models can be implemented alongside an ECM of a cell without needing a full electrochemical reduced-order cell model.

1. The main side reaction is due to the reduction of an organic solvent, expressed as $\text{S} + 2\text{Li}^+ + 2e^- \rightarrow \text{P}$, where "S" refers to the solvent and "P" to the product formed in the side reaction. Specific species of solvent and product are not specified.
2. The reaction occurs only when charging the cell.
3. The products formed are a mixture of different species, resulting in averaged mass and density constants used when describing the formation and growth of the SEI film.
4. The side reaction is assumed to be irreversible and occurs at potential U_s, which is chosen to be 0.4 V versus Li/Li$^+$.
5. The initial resistance of the SEI layer developed during cell formation is $0.01\,\Omega\text{m}^2$.
6. There is no overcharge reaction considered (i.e., lithium plating is not modeled).

We have removed assumption 2 in the models presented here, which now predicts side reactions to occur while the cell is resting and—to some extent—even during discharge. This modification seems to enable the model to match observed calendar-life aging of battery cells better.

The SEI growth model proposed by Ramadass is tightly coupled with a Newman-style physics-based model of ideal-cell dynamics. For the negative electrode, the local molar flux density of lithium j_{total} from the solid to the electrolyte is given by a sum of the intercalation flux density j and the side-reaction flux density j_s:

$$j_{\text{total}} = j + j_s, \tag{7.1}$$

18:59:59.

where the flux densities are measured in [mol m^{-2} s^{-1}]. In this equation, j is computed via the Butler–Volmer electrochemical kinetic expression

$$j = \frac{i_0}{F}\left[\exp\left(\frac{(1-\alpha)F}{RT}\eta\right) - \exp\left(-\frac{\alpha F}{RT}\eta\right)\right],$$

which is driven by the overpotential

$$\eta = \phi_s - \phi_e - U_{\text{ocp}}(c_{s,e}) - FR_{\text{film}}j_{\text{total}},$$

where i_0 [A m^{-2}] is the intercalation exchange-current density and $U_{\text{ocp}}(\cdot)$ is the equilibrium potential in the negative electrode, evaluated as a function of the solid-phase concentration at the surface of the particle, $c_{s,e}$.

The kinetics of the side reaction are described using a *Tafel equation*, which assume that the side reaction is considered irreversible:

$$j_s = -\frac{i_{0,s}}{F}\exp\left(-\frac{\alpha_s F}{RT}\eta_s\right), \tag{7.2}$$

where the side-reaction overpotential is described as

$$\eta_s = \phi_s - \phi_e - U_s - FR_{\text{film}}j_{\text{total}},$$

where U_s is the equilibrium potential of the side reaction.

Once the side reaction flux density j_s has been calculated, the film thickness δ_{film} [m] can be calculated by solving the ordinary differential equation

$$\frac{\partial \delta_{\text{film}}}{\partial t} = -\frac{M_P}{\rho_P}j_s, \tag{7.3}$$

where M_P [kg mol^{-1}] is the average molecular weight of the constituent compounds of the SEI layer and ρ_P [kg m^{-3}] is the average density of the constituent compounds. This allows the overall film resistance to be calculated as

$$R_{\text{film}} = R_{\text{SEI}} + \delta_{\text{film}}/\kappa_P, \tag{7.4}$$

where R_{SEI} [Ω m^2] is the initial film resistance that is produced during the formation cycle of the cell and κ_P [S m^{-1}] is the conductivity of the SEI film.

In addition to the resistance change, there is a capacity loss caused by the side reaction current during charge, leading to the relationship for total-capacity change:

$$\frac{\partial Q}{\partial t} = \int_0^{L^{\text{neg}}} a_s A F j_s \, dx, \tag{7.5}$$

where L^{neg} [m] is the thickness of the negative electrode, a_s [m^2 m^{-3}] is the specific interfacial surface area of the electrode particles, and A [m^2] is the plate area of the current collector.

7.2.2 *Simplifying the model*

To implement an optimal-control strategy, the BMS must be able to calculate the side reaction flux density j_s very quickly and accurately. Solving the coupled PDE equations described above (plus the physics-based ideal-cell model) is too complicated for such a process. The j_s model needs to be much faster and simpler. In this section, we present a simpler incremental model for approximating j_s, R_{film}, and Q.

To create a volume-averaged 0D ROM, we add three additional assumptions to those of Ramadass et al.:

1. The cell is always in a quasi-equilibrium state, allowing the exchange-current density i_0 to be calculated from the cell SOC alone, neglecting local variations in electrolyte and solid surface concentration. The estimated value of j_s then corresponds to a suddenly applied current pulse of magnitude i_{app}.
2. The intercalation and the side-reaction flux densities are uniform over the negative electrode. This allows us to state that the total reaction flux density j_{total} is related to the applied cell current i_{app} via the following relationship:

$$j_{\text{total}} = \frac{i_{\text{app}}}{a_s AFL^{\text{neg}}}, \tag{7.6}$$

   where the volume of the electrode is $V = AL^{\text{neg}}$.
3. The anodic and cathodic charge-transfer coefficients of the intercalation reaction are equal ($\alpha = 0.5$).

From the above assumptions, an incremental degradation model can be formulated as follows. First, at any point in time the lithiation state of the negative electrode is calculated as

$$\theta = \theta_{\min} + z_{\text{cell}} \left(\theta_{\max} - \theta_{\min}\right),$$

where $\theta_{\max}$ and $\theta_{\min}$ are the stoichiometric limits of negative-electrode lithiation (i.e., the value of θ in $\text{Li}_\theta\text{C}_6$ when the cell is fully charged and discharged, respectively) and z_{cell} is a value between zero and one that indicates the cell SOC. Then, assuming that the lithium concentration is uniform across the active-material particles, we compute $c_{s,e} = c_{s,\max}\theta$ and therefore $U_{\text{ocp}}(c_{s,e}) = U_{\text{ocp}}(c_{s,\max}\theta)$ for the electrode materials being used.

Ultimately, we desire to find j_s. We begin by rearranging Eq. (7.1) and substituting Eq. (7.6):

$$\begin{aligned} j &= j_{\text{total}} - j_s \\ &= \frac{i_{\text{app}}}{a_s AFL^{\text{neg}}} - j_s. \end{aligned}$$

18:59:59.

Then, from this formulation of j and assumption 3, we can solve for the intercalation-reaction overpotential η

$$\begin{aligned} j &= \frac{i_0}{F}\left[\exp\left(\frac{F}{2RT}\eta\right) - \exp\left(-\frac{F}{2RT}\eta\right)\right] \\ &= \frac{2i_0}{F}\sinh\left(\frac{F}{2RT}\eta\right) \\ \eta &= \frac{2RT}{F}\text{asinh}\left(\frac{Fj}{2i_0}\right). \end{aligned}$$

To continue, we note the similarity between the expressions for η and η_s to find:

$$\begin{aligned} \eta &= \phi_s - \phi_e - U_{\text{ocp}}(c_{s,e}) - FR_{\text{film}}j_{\text{total}} \\ \eta_s &= \phi_s - \phi_e - U_s - FR_{\text{film}}j_{\text{total}} \\ &= \eta + U_{\text{ocp}}(c_{s,e}) - U_s \\ &= \frac{2RT}{F}\text{asinh}\left(\frac{Fj}{2i_0}\right) + U_{\text{ocp}}(c_{s,e}) - U_s. \end{aligned}$$

A key observation is that the film resistance cancels from the calculation, meaning that we do not need to know its value in order to model SEI growth.

We substitute this result into Eq. (7.2) to get

$$\begin{aligned} j_s &= -\frac{i_{0,s}}{F}\exp\left(\frac{-F}{2RT}\left(\frac{2RT}{F}\text{asinh}\left(\frac{Fj}{2i_0}\right) + U_{\text{ocp}}(c_{s,e}) - U_s\right)\right) \\ &= -\frac{i_{0,s}}{F}\exp\left(\frac{-F}{2RT}\left(\frac{2RT}{F}\text{asinh}\left(\frac{F\left(\frac{i_{\text{app}}}{a_sAFL^{\text{neg}}} - j_s\right)}{2i_0}\right) + U_{\text{ocp}}(c_{s,e}) - U_s\right)\right) \\ &= -\frac{i_{0,s}}{F}\exp\left(\frac{F\left(U_{\text{ocp}}(c_{s,e}) - U_s\right)}{2RT}\right)\exp\left(\text{asinh}\left(\frac{\frac{-i_{\text{app}}}{a_sAL^{\text{neg}}} + Fj_s}{2i_0}\right)\right). \end{aligned}$$

We'll see how to solve this equation for j_s shortly.

Once we have solved for j_s, it can then be incorporated into incremental equations for film resistance and capacity loss. We assume that j_s is constant over some small time interval Δt, where its value is denoted as $j_{s,k}$ for the kth interval. We can then convert the continuous-time film thickness relationship Eq. (7.3) to discrete time as:

$$\delta_{\text{film},k} = \delta_{\text{film},k-1} - \frac{M_P\Delta t}{\rho_P}j_{s,k-1}, \tag{7.7}$$

noting that the sign of j_s is negative. This result can be used to calculate the film resistance by converting Eq. (7.4) into a discrete-time recurrence:

$$R_{\text{film},k} = R_{\text{film},k-1} - \frac{M_P\Delta t}{\rho_P\kappa_P}j_{s,k-1}. \tag{7.8}$$

Similarly, we can discretize the capacity-loss Eq. (7.5):

$$Q_k = Q_{k-1} + (a_s AFL_n \Delta t)\, j_{s,k-1}. \tag{7.9}$$

In summary, the proposed ROM equations are:

$$\begin{aligned}
\theta &= \theta_{\min} + z_{\text{cell}}\left(\theta_{\max} - \theta_{\min}\right)\\
j_{s,k} &= -\frac{i_{0,s}}{F}\exp\left(\frac{F\left(U_{\text{ocp}}(c_{s,e}) - U_s\right)}{2RT}\right)\exp\left(\operatorname{asinh}\left(\frac{\frac{-i_{\text{app}}}{a_s AL^{\text{neg}}} + Fj_{s,k}}{2i_0}\right)\right)
\end{aligned} \tag{7.10}$$

$$\begin{aligned}
R_{\text{film},k} &= R_{\text{film},k-1} - \frac{M_P \Delta t}{\rho_P \kappa_P} j_{s,k-1}\\
Q_k &= Q_{k-1} + (a_s AFL_n \Delta t)\, j_{s,k-1}.
\end{aligned}$$

7.2.3 *Simplifying the calculation*

As it is written now, Eq. (7.10) is an implicit function, meaning that $j_{s,k}$ is not isolated on one side of the equation. It is not immediately clear how to solve this equation for $j_{s,k}$. The approach used in the ROM proposed by Randall et al. was to use iteration. That is,

1. We begin by guessing a value for $j_{s,k}$ (e.g., zero).
2. We substitute the value for $j_{s,k}$ into the right-hand side of Eq. (7.10), computing a new left-hand-side result. We set the new value for $j_{s,k}$ to this new left-hand-side result.
3. Then, we repeat step 2 until we observe no significant change in the value $j_{s,k}$.

This method works well for this problem, and we arrive at a good solution in fewer than 10 iterations.

However, there is also a closed-form solution for $j_{s,k}$, which is more accurate and executes more quickly.[8] This result is not obvious, but can be derived relatively quickly.

[8] This was first reported in Plett, G., "Algebraic Solution for Modeling SEI Layer Growth," *ECS Electrochemistry Letters*, 2(7), 2013, pp. A63–A65.

First, we simplify notation by defining new temporary functions $A(\theta)$ and $B(i_{\text{app}})$, and constant C:

$$\begin{aligned}
j_{s,k} &= \underbrace{-\frac{i_{0,s}}{F}\exp\left(\frac{F\left(U_{\text{ocp}}(c_{s,e}) - U_s\right)}{2RT}\right)}_{A(\theta_n)}\exp\left(\operatorname{asinh}\left(\underbrace{\frac{-i_{\text{app}}}{2a_s i_0 AL^{\text{neg}}}}_{B(i_{\text{app}})} + \underbrace{\frac{F}{2i_0}}_{C} j_{s,k}\right)\right)\\
&= A(\theta)\exp\left(\operatorname{asinh}\left(B(i_{\text{app}}) + Cj_{s,k}\right)\right).
\end{aligned}$$

Note that the new function $A(\theta)$ is different from the current-collector plate area having the same symbol, which should be clear from the context, and that $A(\theta) < 0$ and $C > 0$ always. Also note that the value of $A(\theta)$ can be precomputed and stored in a lookup table versus θ, so is not difficult to calculate in real time.

A useful identity for simplifying this further is:

$$\exp(\operatorname{asinh}(x)) = x + \sqrt{x^2+1}.$$

So, writing $A(\theta)$ simply as A and $B(i_{\text{app}})$ simply as B, we can write

$$j_{s,k} = A\left[(B + Cj_{s,k}) + \sqrt{(B+Cj_{s,k})^2 + 1}\right].$$

This equation can be solved for $j_{s,k}$ using standard algebraic manipulations. First, we isolate the radical, and then square both sides of the equation:

$$\begin{aligned}
\frac{j_{s,k}}{A} - B - Cj_{s,k} &= \sqrt{(B+Cj_{s,k})^2+1} \\
j_{s,k}(1-CA) - AB &= A\sqrt{(B+Cj_{s,k})^2+1} \\
(j_{s,k}(1-CA) - AB)^2 &= A^2\,(B+Cj_{s,k})^2 + A^2.
\end{aligned}$$

Then, we expand and rearrange terms

$$\begin{aligned}
0 &= A^2\,(B+Cj_{s,k})^2 + A^2 - (j_{s,k}(1-CA) - AB)^2 \\
&= A^2\left(B^2 + 2BCj_{s,k} + C^2 j_{s,k}^2\right) + A^2 \\
&\quad - \left(j_{s,k}^2(1-CA)^2 - 2AB(1-CA)j_{s,k} + A^2B^2\right).
\end{aligned}$$

Next, we collect like terms

$$\begin{aligned}
0 = &\left(A^2C^2 - (1-CA)^2\right) j_{s,k}^2 + \left(2A^2BC + 2AB(1-CA)\right) j_{s,k} \\
&+ \left(A^2B^2 + A^2 - A^2B^2\right).
\end{aligned}$$

Note that $(1-CA)^2 = 1 - 2CA + A^2C^2$, so this simplifies to

$$0 = (2CA-1)\,j_{s,k}^2 + (2AB)\,j_{s,k} + (A^2).$$

The key point is that this equation is in a quadratic form, so we can solve for the roots easily using the quadratic formula:

$$\begin{aligned}
j_{s,k} &= \frac{-2AB \pm \sqrt{4A^2B^2 - 4A^2(2CA-1)}}{2(2CA-1)} \\
&= \frac{AB \pm A\sqrt{B^2 + (1-2CA)}}{(1-2CA)}.
\end{aligned}$$

But, which root to use? The Routh test gives us some guidance. We form the Routh array:

$$\begin{array}{c|cc}
j_{s,k}^2 & 2CA-1, & A^2 \\
j_{s,k} & 2AB & \\
1 & A^2 &
\end{array}$$

Then, we check the number of sign changes as we traverse the leftmost column from top to bottom. Since $2CA - 1 < 0$ and $A^2 > 0$, we are guaranteed two sign changes regardless of the sign of $2AB$, which means that this equation always has exactly one positive real root and one negative real root.

Physically, we know that the side-reaction flux density must be negative (because $A < 0$), so we want to take the smaller root of the quadratic solution. So, our final ROM solution for the side-reaction flux density is to compute:

$$\begin{aligned} A(\theta) &= -\frac{i_{0,s}}{F}\exp\left(\frac{F\left(U_{\text{ocp}}(c_{s,e}) - U_s\right)}{2RT}\right) \\ B(i_{\text{app}}) &= \frac{-i_{\text{app}}}{2a_s i_0 A L^{\text{neg}}} \\ C &= \frac{F}{2i_0} \\ j_{s,k} &= \frac{A(\theta)B(i_{\text{app}}) + A(\theta)\sqrt{B(i_{\text{app}})^2 + (1 - 2CA(\theta))}}{(1 - 2CA(\theta))}. \end{aligned}$$

7.3 SEI ROM results

The validity of this ROM depends first on the accuracy of the underlying PDE FOM, which we assume here to be exact. It then depends on how closely the reduced-order approximation of j_s matches the exact calculation of j_s.

To compare the FOMs and ROMs, we conducted a series of simulations. In each simulation, the cell was initially at rest. Then, a sudden pulse of current was applied and the instantaneous resulting j_s from the FOM was compared to the computed j_s from the ROM. To simulate the FOM, we used COMSOL Multiphysics® coupled with a MATLAB script to cycle through the series of simulations and to analyze results.[9]

[9] COMSOL Multiphysics is a registered trademark of The COMSOL Group. From now on, this product will be referred to simply as COMSOL.

Specifically, each simulation considered a 1 s time interval where the cell current i_{app} was modeled as a Heaviside step function that was applied halfway through the interval. We found that the initial 0.5 s rest interval facilitated convergence of the solution by allowing the PDE solver to adjust its initial conditions before applying the step current. The simulation cell parameters that we used are listed in the appendix to this chapter on pg. 310. In particular, the cell had a 1.8-Ah capacity.

For the full-order PDE simulations, applied current was varied from 0 A to 5.4 A in steps of 0.1 A, initial cell SOC was varied from 0 % to 100 % in steps of 2 %, and temperature was varied from −35 °C to 45 °C in steps of 20 C°. For the reduced-order simulations, which run much more quickly, applied current was varied over the same range in steps of 0.05 A, initial cell SOC was varied in steps of 1 %, and temperature was varied in steps of 10 C°.

As one point of comparison between the FOM and ROM simulations, the set of 14,025 FOM simulations took more than eight days to complete on an Intel i7 processor, while the set of 112,200 ROM simulations took a total of about 2.6 seconds to complete on the same machine. The speedup, on a per-simulation basis, is more than 2,000,000 : 1. This is the primary advantage of the ROM over the FOM.

When comparing modeling results between the FOM and ROM, we note that film thickness, film resistance, and capacity loss are all deterministic functions of side-reaction flux density j_s. Therefore, if we are able to compute j_s accurately, the other computations will be accurate as well. So, in the following, we consider only simulation results concerning j_s.

Fig. 7.1 shows j_s as computed by the ROM, which we now denote as $j_{s,\mathrm{ROM}}$, at different initial cell SOCs and charge-current levels. The top frame shows results at 25 °C and the lower frame shows a compilation of $j_{s,\mathrm{ROM}}$ over a range of temperatures. More negative values correspond to worse degradation. We see two trends that match experience: degradation is worst at high SOCs and at high charge rates.

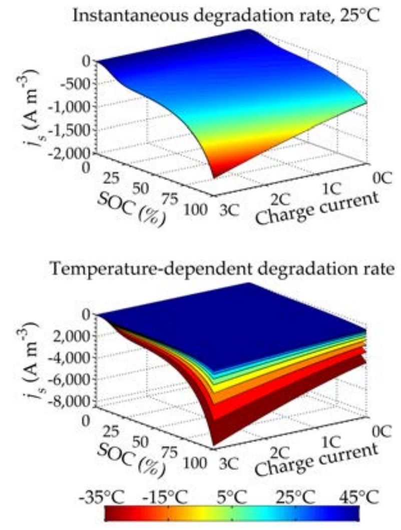


Figure 7.1: Instantaneous SEI degradation rate as computed by the ROM. (Adapted from Fig. 1 in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

Fig. 7.2 shows results of a single FOM simulation compared to the corresponding results from a ROM simulation.. This example was conducted at 25 °C, 50 % SOC, and by applying a 1C charge pulse at $t = 0.5$ s. Both the FOM and ROM solutions have a nonzero negative side-reaction flux j_s even when the cell is at rest. This is due to the fact that we have removed Ramadass' assumption 2 of the SEI growth model, and so also allow for the side reaction when current in the external circuit is zero. The figure shows that the ROM matches both the rest SEI side-reaction rate and the charge-pulse SEI side-reaction rate of the FOM very well.

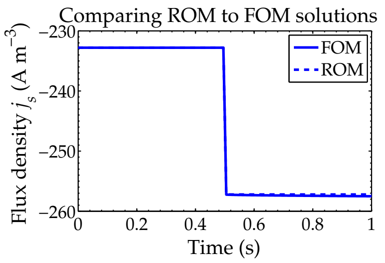


Figure 7.2: Comparing a single FOM to ROM simulation.
(Adapted from Fig. 2 in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

Plotted on the same scale, the FOM results are nearly indistinguishable from the ROM results. So, for comparison purposes, we define a relative error between them as

$$j_{s,\mathrm{err}\%} = \frac{j_{s,\mathrm{FOM}} - j_{s,\mathrm{ROM}}}{j_{s,\mathrm{FOM}}} \times 100,$$

where $j_{s,\mathrm{FOM}}$ is chosen to be the value of j_s from the FOM solution immediately after the application of the current pulse. Fig. 7.3 plots the relative error between the PDE and ROM solutions for all 25 °C simulations. Between 10 % and 90 % SOC (e.g., representative extremum operating conditions for xEV cells), the maximum relative error was 0.44 %.

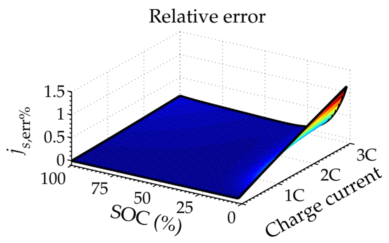


Figure 7.3: Relative error between FOM and ROM.
(Adapted from Fig. 3 in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

To further illustrate the performance of the ROM and to see the

18:59:59.

dependence of SEI layer growth rate on SOC and charge rate, Fig. 7.4 plots the same results as the top frame of Fig. 7.1 in a different format. The top frame of Fig. 7.4 shows j_s as a function of SOC at different charge rates (lines are plotted from 0C to 3C in steps of 0.5C). The bottom frame shows j_s as a function of charge rate at different SOCs (plotted from 0 % SOC to 100 % SOC in steps of 10 % SOC). In all plots, the FOM result is drawn as a solid line and the ROM result is drawn as a dashed line. In most cases, it is impossible to distinguish between the FOM and ROM results visually.

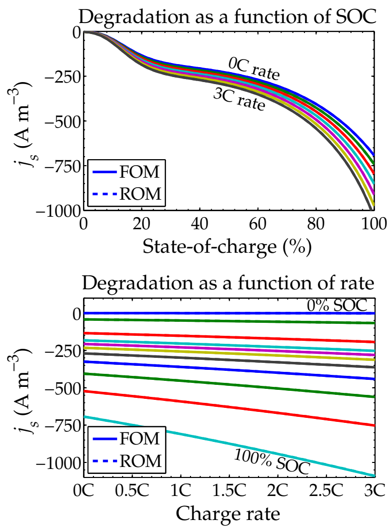


Figure 7.4: Another view of FOM versus ROM model results.
(Adapted from Fig. 4 in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

Fig. 7.5 shows how modeling error varies with temperature. The ROM predictions are best at high temperatures and less good at low temperatures. Worst-case $j_{s,\text{err\%}}$ in the 10 % to 90 % SOC range varies from 0.41 % at 45 °C to 0.55 % at −35 °C.

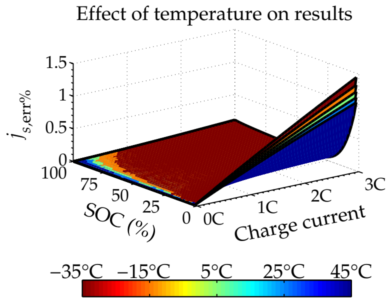


Figure 7.5: Effect of temperature on modeling error.
(Adapted from Fig. 5a in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

Next, we investigate the effect of Δt on the results. Instead of selecting the value for $j_{s,\text{FOM}}$ immediately after the application of the current pulse, $j_{s,\text{FOM}}$ is now selected to be the PDE solution 0.5 s seconds after the application of the current pulse, at the $t = 1$ s point. Relative modeling error is plotted in Fig. 7.6. This error is once again worst at low temperatures and low values of SOC (where the absolute amount of degradation is small). Relative errors over 10 % are observed in some cases, but in the ranges of SOC most important for control, where SOC is greater than 25 %, the worst-case $j_{s,\text{err\%}}$ is far less, varying from 0.85 % at 45 °C to 1.04 % at −35 °C.

Fig. 7.7 investigates the effect of a prolonged constant-current charge at a 1C rate, as might be experienced when a cell is being charged. The FOM is simulated for 3,000 s, starting with the cell at rest at 10 % SOC, and 1D profiles of $j_s(x)$ across the negative electrode are plotted at time steps 100 s, 1,000 s, 2,000 s, and 3,000 s. (The position variable is normalized to have value 0 at the current-collector/electrode boundary and value 1 at the electrode/separator boundary.) Overlaid on the plot are the average j_s values predicted by the ROM at that SOC level, and the actual j_s values averaged over the 1D electrode from the FOM solution. In the ROM simulation, the SOC is updated on a second-by-second basis to achieve the present SOC at every point, which is then used to compute the value of j_s using the method explained herein. We see that the ROM is accurate even over prolonged constant-current charge profiles, indicating that assumption 1 of the ROM is reasonable.

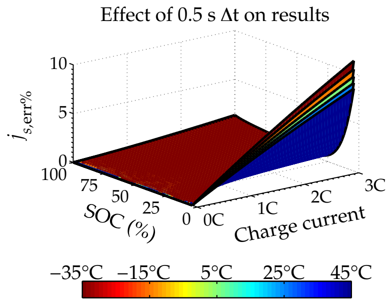


Figure 7.6: Effect of delay on modeling error.
(Adapted from Fig. 5b in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

Overall, the simulations show that, at least for these parameter values, the ROM and FOM are in good agreement. Further, the ROM predictions can be computed knowing only cell SOC, temperature, and the present values of input current. A full physics-based model is not needed. Therefore, the 0D ROM can be used alongside an equivalent-circuit model in a BMS to assist in controls designs that seek to minimize cell degradation due to SEI growth.

7.4 *Lithium plating on overcharge*

The success of the ROM of SEI growth to predict the performance of a Tafel equation leads us to expect that the same approach will work well for creating ROMs of other aging mechanisms that are modeled in the same way.[10] However, one significant mechanism has been modeled in a somewhat different way: Arora et al. use a modified Butler–Volmer equation instead of a Tafel equation to predict *lithium deposition* or *lithium plating* on overcharge.[11]

Overcharge manifests first as a metallic lithium deposit on the surface of the negative-electrode solid particles during charge, predominantly near the separator. Subsequently, the lithium further combines with electrolyte material to form other compounds such as Li_2O, LiF, Li_2CO_3, and polyolefins. The nature of the final product is not our major concern; rather, the issue is that lithium is irreversibly lost. This phenomenon is an unintended side reaction that can lead to severe capacity fade, electrolyte degradation, and a possible safety hazard as metallic lithium dendrites are formed that can short-circuit the cell.

Here, we briefly address creation of a ROM of Arora's FOM.[12] It does not work as well as the ROM of SEI growth, especially for prolonged charging events, so is probably better suited for predicting degradation in an HEV-like application where random charging is expected. As with the SEI model, a full physics-based model of cell behaviors is not needed as input to its calculations; rather, it can receive all required inputs from an equivalent-circuit model of cell behavior. However, a section in the planned Volume III of this series will show that much better lithium-plating predictions can be made if a physics-based cell model is used instead.

Lithium plating is not usually considered to be a dominant degradation mechanism because the cell manufacturer-specified terminal-voltage limits are designed to avoid conditions that would be conducive to plating. However, cell terminal voltage is a poor indicator of internal cell potentials—especially at cold temperatures—and so plating can still happen: when it does, there is immediate severe capacity loss. Further, since we are removing voltage limits in our physics-based power calculations, we require a model of lithium plating in order to be able to devise optimized controls to minimize aging.

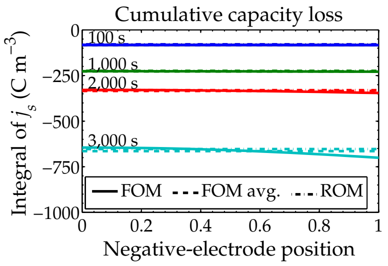


Figure 7.7: Long-term modeling error. (Adapted from Fig. 6 in Randall et al., *Journal of Power Sources*, 209, 2012, pp. 282–288.)

[10] See, for example, Darling, R. and Newman, J., "Modeling Side Reactions in Composite $Li_yMn_2O_4$ Electrodes," *Journal of the Electrochemical Society*, 145(3), 1998, pp. 990–998.

[11] Arora, P., Doyle, M., and White, R.E., "Mathematical Modeling of the Lithium Deposition Overcharge Reaction in Lithium-Ion Batteries Using Carbon-Based Negative Electrodes," *Journal of the Electrochemical Society*, 146(10), 1999, pp. 3,543–53.

[12] Adapted from, Perkins, R.D., Randall, A.V., Zhou, X., Plett, G.L., "Controls Oriented Reduced Order Modeling of Lithium Deposition on Overcharge," *Journal of Power Sources*, 209, 2012, pp. 318–325.

7.4.1 *Physics-based model of overcharge*

The full-order PDE model proposed by Arora et al. made the following assumptions:

1. The main side reaction is expressed as $\mathrm{Li}^{+} + e^{-} \rightarrow \mathrm{Li}_{(\mathrm{s})}$ and occurs at potential $U_s = 0\,\mathrm{V}$ versus $\mathrm{Li}/\mathrm{Li}^{+}$ during an overcharge event. This lithium metal is expected to form first near the electrode/separator boundary where the surface overpotential tends to have the greatest magnitude.
2. Lithium metal deposited on the negative electrode reacts quickly with solvent or salt molecules in the vicinity, yielding $\mathrm{Li_2CO_3}$, LiF, or other insoluble products. A thin film of these products protects the solid lithium from reacting with the electrolyte. Solid lithium can still dissolve during discharge, but once lithium is consumed in a insoluble product, it is permanently lost.
3. The insoluble products that are formed are a mixture of different species, resulting in averaged mass and density used in the model to describe the formation and growth of a resistive film.
4. Only the overcharge reaction is considered (e.g., SEI film growth and other degradation mechanisms are not modeled).

The overcharge FOM is tightly coupled with a Newman-style physics-based model of ideal-cell dynamics. Just as with the SEI model, the local molar flux density of lithium j_{total} in the negative electrode is given by a sum of the intercalation flux density j and the side-reaction flux density j_s. The intercalation flux density is expressed using the standard Butler–Volmer equation,

$$j(x,t) = \frac{i_0}{F}\left[\exp\left(\frac{(1-\alpha)F}{RT}\eta(x,t)\right) - \exp\left(-\frac{\alpha F}{RT}\eta(x,t)\right)\right]$$

which is driven by the overpotential

$$\eta(x,t) = \phi_s(x,t) - \phi_e(x,t) - U_{\text{ocp}}(c_{s,e}) - FR_{\text{film}}j(x,t),$$

where i_0 is the exchange current density,

$$i_0 = k\left(c_{s,\max} - c_{s,e}\right)^{1-\alpha}\left(c_e\right)^{1-\alpha}\left(c_{s,e}\right)^{\alpha},$$

and U_{ocp} is the equilibrium potential that is evaluated as a function of the solid phase concentration at the surface of the particle.

Arora expresses the side-reaction flux density j_s (i.e., the rate of irreversible lithium loss due to lithium plating) as

$$j_s(x,t) = \min\left(0, \frac{i_{0,s}}{F}\left[\exp\left(\frac{(1-\alpha_s)F}{RT}\eta_s(x,t)\right) - \exp\left(-\frac{\alpha F}{RT}\eta_s(x,t)\right)\right]\right), \tag{7.11}$$

18:59:59.

where $\alpha_s \neq \alpha$ in general,

$$\eta_s(x,t) = \phi_s(x,t) - \phi_e(x,t) - U_s - FR_{\text{film}} j_s(x,t),$$

and where the side-reaction exchange current density is computed as $i_{0,s} = k_s(c_e)^{1-\alpha_s}$. Side reaction is semi-irreversible in the sense that it includes an anodic rate term, but doesn't allow overall positive side-reaction flux.

The side reaction occurs only at spatial locations in the negative electrode where $\eta_s(x,t) < 0$. This is enforced in Eq. (7.11) by the $\min(\cdot)$ function, which sets $j_s(x,t) = 0$ for values of x where $\eta_s(x,t) \geq 0$, but to the value computed by the Butler–Volmer equation when $\eta_s(x,t) < 0$.

When charging a lithium-ion cell, the local side-reaction overpotential decreases over time. It is not uniform across the electrode; instead, the value near the separator tends to decrease more quickly than at other places in the electrode. A typical scenario is sketched in Fig. 7.8. In this example, plating will occur over the interval where $\eta_s(x,t) < 0$, from $x = x_0$ to $x = L^{\text{neg}}$. Note that this illustration shows that the cell can be quite far away from 100 % SOC (because $\eta_s > 0$ over much of the electrode cross section) and still have plating occur near the separator if a large enough charge-current pulse is applied to the cell's terminals. Therefore, SOC is only one variable of importance: ultimately, the local overpotential determines whether plating occurs.

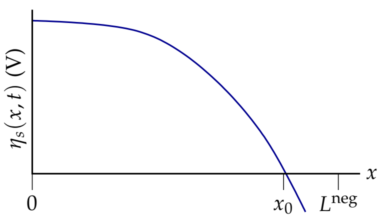


Figure 7.8: Illustrating overpotential during charge across the negative-electrode width.
(Adapted from Fig. 1 in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

To make a ROM of lithium plating, our first goal is to solve for $\eta_s(x,t)$ and then from it to compute j_s. Subsequently, we can incorporate j_s into incremental equations describing the evolution of film resistance and capacity loss, much as we did for SEI growth. We will assume that j_s is constant over some small time interval Δt, and denote it as $j_{s,k}$ for the kth interval. Then, the film-thickness, film-resistance, and capacity-loss equations take on the same form as for SEI growth (Eqs. (7.7) through (7.9)), but with a different input side-reaction flus density $j_{s,k}$.

7.5 Plating ROM results

When creating a ROM of this FOM of lithium deposition on overcharge, we quickly encounter a major difference between the Ramadass and Arora models. The Ramadass model assumes that SEI grows everywhere in the electrode, all the time. The rate of growth depends on the local side-reaction overpotential, but since this variable is continuous across the electrode we can approximate the overall rate of side reaction in the electrode simply by averaging the side-reaction overpotential across the electrode to come up with an average (or total) side-reaction flux density.

The Arora model instead assumes that lithium plating is a binary phenomena. Anywhere in the electrode that the side-reaction overpotential is positive, lithium plating will not occur. However, anywhere in the electrode where this overpotential is negative, lithium plating will occur with severity described by a Butler–Volmer equation. Therefore, we cannot simply average the side-reaction overpotential across the electrode to predict lithium plating. Instead, we must attempt to approximate the profile of this overpotential, find where the overpotential is negative, and integrate the lithium loss over the region of the electrode where plating is occurring.

The details of the derivation of the ROM are presented in the paper by Perkins et al. referenced in footnote 12. As with the ROM of SEI growth, it results with an implicit calculation for the average side-reaction flux density $\bar{j}_s$. However, unlike the SEI-model ROM, there is no closed-form solution for this calculation and a nonlinear solver is required to compute $\bar{j}_s$ at every iteration. Even so, it executes much more quickly than the FOM, and can give good predictions in some settings. We review some results from that paper here.

The validity of this ROM depends first on the accuracy of the underlying full-order partial-differential-equation model, which we assume here to be exact. It then depends on how closely the reduced-order approximation of $\bar{j}_s$ matches the exact calculation of $\bar{j}_s$. In this section, results from both the FOM and ROM for $\bar{j}_s$ are compared.

To compare the FOMs and ROMs, we conducted a series of simulations. In each simulation, the cell was initially at rest. A sudden pulse of current was then applied and the resulting $\bar{j}_s$ from the PDE model, averaged over a one-second interval subsequent to the pulse, was compared to the computed $\bar{j}_s$ from the ROM.

To simulate the PDE model, we used COMSOL Multiphysics coupled with a MATLAB script to cycle through the series of simulations and analyze results. Specifically, each simulation comprised a 1.2 s time interval, where the cell current i_{app} was modeled as a step function, which was applied at $t = 0.2$ s. We found that the initial rest interval facilitated convergence of the solution by allowing the PDE solver to adjust its initial conditions before applying the step current.

The cell parameters that we used in the simulations match those used in Arora and are listed in the appendix on pg. 311. The applied current was varied between 0C and 3C in increments of C/33; The initial cell SOC was varied between 0 % and 100 % in steps of 1 %, and temperature was held constant at 25 °C. We found that the adjustable tuning factor required by the ROM worked well with a value of $\beta = 1.7$ (this implies that electrolyte concentration near the separator changes nearly twice as quickly as it does near the current collector for a suddenly applied pulse).

A total of 10,100 simulations were run. As one point of comparison, the set of full-order PDE simulations took approximately 12 hours to complete, using an average of three processor cores on an Intel i7 CPU, while the set ROM simulations took approximately 21 seconds to complete, using an average of one core on the same machine. The speedup, on a per-simulation per-core basis, is more than 5,000 : 1. This is the primary advantage of the ROM over the FOM.

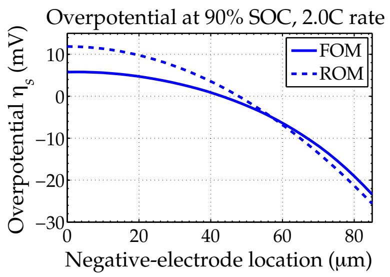


Figure 7.9: Cross-sectional view of side-reaction overpotential during a charge pulse.
(Adapted from Fig. 2a in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

Fig. 7.9 plots the side-reaction overpotential across the negative electrode for this cell model immediately following the onset of a 2C charge-current pulse applied to a cell having initial 90 % SOC, and where $x = 0$ is adjacent to the current-collector and $x = 85\,\mu$m is adjacent to the separator. Lithium plating occurs wherever and whenever $\eta_s < 0$, so from the FOM result, we expect lithium to deposit between about $x = 42\,\mu$m and $x = 85\,\mu$m. From the ROM, we expect lithium deposit to occur between about $x = 49\,\mu$m and $x = 85\,\mu$m. Therefore, in this example, the ROM underpredicts the width of the deposition region, but it actually overpredicts the amount of deposition since it predicts a more negative overpotential than the FOM over this region.

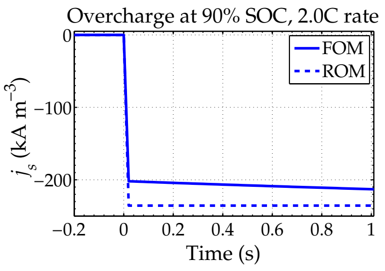


Figure 7.10: Overcharge predictions for the same scenario.
(Adapted from Fig. 2b in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

This can be seen more easily in Fig. 7.10. The time-average deposition rate predicted by the ROM is somewhat higher than the time-average deposition rate of the FOM over the 1 s interval.

Fig. 7.11 displays aggregate results of predicted overcharge rates over all scenarios for the FOM and the ROM solutions. As expected, deposition is worse at high SOC and high charge rates. The FOM and ROM solutions generally agree very well, with greatest mismatch at high charge rates.

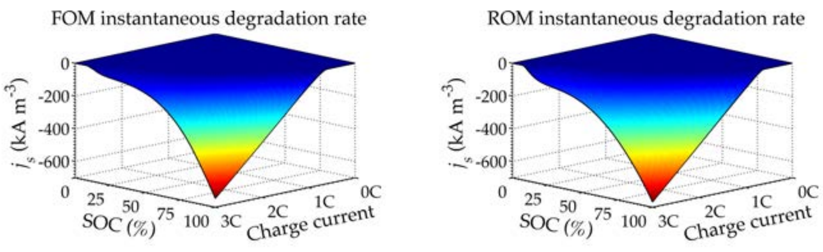


Figure 7.11: FOM and ROM predictions of capacity loss due to overcharge.
(Adapted from Fig. 3 in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

Fig. 7.12 shows a different view of the results. Cross sections through both the FOM and ROM solution spaces are plotted and compared. The left frame shows how the two methods compare where each pair of lines represents a specific charge rate. As noted before, but perhaps more clearly seen here, the difference between the FOM and ROM solutions are greatest at high charge rates. The

18:59:59.

right frame shows how the two methods compare where each pair of lines represents a specific initial SOC. The difference is greatest at moderate SOC levels.

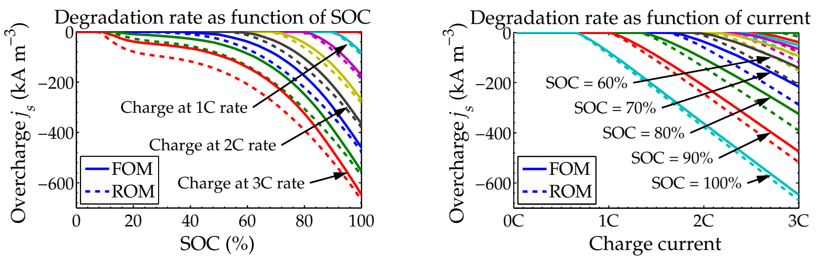


Figure 7.12: Cross-sectional views of Fig. 7.11.
(Adapted from Fig. 4 in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

Finally, Fig. 7.13 illustrates the error between the FOM and ROM solutions in two ways. The top frame shows the regions where the two methods agree on whether lithium deposition will occur and the region where they disagree. The region of disagreement is the very narrow sliver at around 2.4C and 25 % SOC, where the ROM predicts overcharge but the FOM does not. Otherwise, the boundaries are identical. The bottom frame shows the error between the solutions, calculated as $\bar{j}_{s,\mathrm{ROM}} - \bar{j}_{s,\mathrm{FOM}}$. The maximum error is approximately 65 000 A m^{-3}, which seems large but represents a relative error only on the order of 10 %.

For the purpose of control system design, the results of the top frame are the most important. Since lithium deposition can be such a severe degradation mechanism, a charging control scheme should avoid ever commanding a control action that would cause any lithium deposition to occur. A time-optimal charger based only on the FOM model of lithium deposition would control current to follow the upper contour in the left frame. This allows the maximum charge rate at any point in time while causing no lithium plating. In comparison, a time-optimal charger based only on the ROM model of lithium deposition would control current to follow the lower contour in the figure. This will result in somewhat slower charging. But, because the ROM overpredicts the amount of lithium deposition, it will also result in a charging scheme that is conservative, which is a beneficial feature.

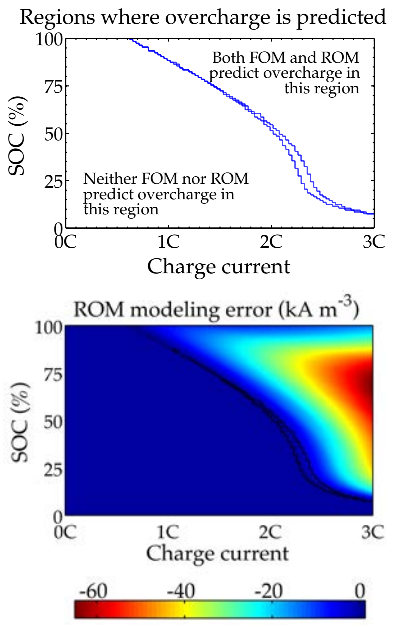


Figure 7.13: Decision regions for the FOM and ROM.
(Adapted from Fig. 5 in Perkins et al., *Journal of Power Sources*, 209, 2012, pp. 318–325.)

Note that constant-current, constant-voltage charging limited to a 1C rate will avoid lithium plating if the voltage limit is chosen properly. The constant-current step will bring SOC to about 80 %, after which the constant-voltage step will gradually charge the rest of the way. However, we see that rates much higher than 1C may be used initially to charge a cell much more quickly.

We conducted additional simulations to investigate the effect of charge-pulse duration Δt. That is, how long can the charge pulse

18:59:59.

be sustained before the FOM and the ROM results are significantly different? We found that pulse lengths less than 10 s are generally well matched, but that pulse lengths much greater than 10 s can give significant FOM versus ROM mismatch. For long pulse durations, the quasi-static nature assumed in the derivation of the ROM is violated, and a significant offset is noted in actual time-varying $\phi_e(x,t)$ versus the at-rest $\phi_e(x,t)$, moving the crossover point of $\eta_s(x,t)$. This causes the ROM to under-predict the value of lithium plating computed by the FOM. For this reason, we propose that the ROM is of most value for computing current limits in dynamic applications such as HEVs, where a bias in ϕ_e cannot develop due to the random nature of power demand, but is of less value for controlling full charges, such as for EV applications.

We make one final comment regarding efficiency. The speedup of the ROM versus the FOM can be much greater than 5,000 : 1 if ROM solutions are precomputed and stored in a lookup table. Then, "computing" any value of $\bar{j}_{s,\text{ROM}}$ would be nearly instantaneous via table lookup. We note that, unlike the SEI model, the ROM solution for lithium plating changes as the film resistance changes. However, the film resistance changes very slowly. Therefore, the entire table might be updated by the BMS once per operational period (e.g., once per day), and then utilized throughout that operational period for significant performance gains.

7.6 Optimized power limits

We have now seen that there are quite a few causes of cell degradation and attempts to model two of the more significant mechanisms. Much more work remains to be done in this area, first by electrochemists and materials scientists to develop FOMs, then by control-systems engineers to convert these to computationally efficient high-fidelity ROMs.

But, how does one use these models to compute power limits to slow aging? We look at a few methods next.

We have seen that none of the cell degradation mechanisms are tied directly to cell terminal voltage, but are rather functions of cell internal stress factors. Therefore, assuming that degradation mechanisms can be well modeled, it makes more sense to compute power limits based on predicted capacity loss and/or impedance rise due to these stress factors than to compute them based on voltage limits. Clearly, there's a lot of work to do before this becomes practical, but the potential benefits indicate that the effort is worthwhile.

The next sections very briefly introduce some optimization methods that might be used with the physics-based degradation mechanisms to compute better power limits. In these sections, we consider two distinct control problems:[13]

1. For applications such as EV, E-REV, or PHEV where the battery pack is charged from an external source: What is the optimal charge profile? Can we fast-charge? For a fixed charge duration, what is the best charging strategy?
2. For any dynamic application, including all xEV applications while the car is being driven: What is the maximum absolute charge power that can be maintained over the next ΔT seconds? What is the maximum discharge power that can be maintained over the next ΔT seconds?

Different kinds of optimized controls may be better for these two problems.

[13] A third problem of interest that is well beyond our scope here is: Considering xEV as storage units for the smart grid, when does it make sense to lend energy to the grid? What should be the rental fee charged for allowing energy to be borrowed?

7.7 *Plug-in charging*

The plug-in charging problem lends itself well to being solved by a nonlinear-programming method. One example is the sequential quadratic programming algorithm, implemented in MATLAB's Optimization Toolbox™ as `fmincon.m`.

Nonlinear programming is a generic optimization method that attempts to find solutions to problems that can be posed in the framework:

$$x^* = \arg\min f(x), \quad \text{such that} \begin{cases} \boldsymbol{c}(\boldsymbol{x}) & \le & \boldsymbol{0} & \quad \boldsymbol{A}\boldsymbol{x} & \le & \boldsymbol{b} \\ \boldsymbol{c}_{\text{eq}}(\boldsymbol{x}) & = & \boldsymbol{0} & \quad \boldsymbol{A}_{\text{eq}}\boldsymbol{x} & = & \boldsymbol{b}_{\text{eq}} \\ \boldsymbol{lb} & \le & \boldsymbol{x} & \quad \boldsymbol{x} & \le & \boldsymbol{ub}, \end{cases}$$

where $f(\boldsymbol{x})$ is a scalar cost function that we wish to minimize by choosing optimum input vector $\boldsymbol{x}^*$ such that the following constraints are satisfied:

- Nonlinear vector inequality constraint function $\boldsymbol{c}(\boldsymbol{x}) \le \boldsymbol{0}$,
- Nonlinear vector equality constraint function $\boldsymbol{c}_{\text{eq}}(\boldsymbol{x}) = \boldsymbol{0}$,
- Linear vector inequality constraint function $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$,
- Linear vector equality constraint function $\boldsymbol{A}_{\text{eq}}\boldsymbol{x} = \boldsymbol{b}_{\text{eq}}$, and
- Bounds $\boldsymbol{lb} \le \boldsymbol{x} \le \boldsymbol{ub}$ for all entries in vector $\boldsymbol{x}$,

for user-specified $f(\boldsymbol{x})$, $\boldsymbol{c}(\boldsymbol{x})$, $\boldsymbol{c}_{\text{eq}}(\boldsymbol{x})$, $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{A}_{\text{eq}}$, $\boldsymbol{b}_{\text{eq}}$, $\boldsymbol{lb}$, and $\boldsymbol{ub}$.

To use nonlinear programming to solve a specific problem, we must define an appropriate input vector $\boldsymbol{x}$, cost function $f(\boldsymbol{x})$, and constraint matrices and functions. For the plug-in charging problem, we choose $\boldsymbol{x}$ to be a vector of cell applied current versus time, $f(\boldsymbol{x})$ to be some estimate of the cell degradation that would be caused by that applied current, and the other functions and matrices to make the problem work.

For example, we might want to find

$$\boldsymbol{i}^* = \arg\min \sum_{k=0}^{K-1} -j_s\left(i_k, z_k, T_k\right)$$
$$\text{such that} \begin{cases} z_{\min} \le & z_k & \le z_{\max} \\ z_K = & z_{\text{end}} & \\ -I_{\max} \le & i_k & \le I_{\max} \end{cases}$$
$$\text{and } z_k = z_0 - \sum_{j<k} i_j \Delta t / Q.$$

This states that we want to minimize the accumulated capacity loss that would be experienced by a cell if we were to start at SOC z_0 and end at SOC z_{end} over a period of K sampling intervals, where current is limited between $\pm I_{\max}$, SOC is limited between $z_{\min}$ and $z_{\max}$, and the standard SOC equation holds (approximating coulombic efficiency as perfect).

This formulation can be recast into the nonlinear-programming paradigm with a little work. First, consider the SOC equation. We can write it in vector form as

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\boldsymbol{CV}} z_0 - \frac{\Delta t}{Q} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix}}_{\boldsymbol{LT}} \underbrace{\begin{bmatrix} i_0 \\ i_1 \\ \vdots \\ i_{K-1} \end{bmatrix}}_{\boldsymbol{x}},$$

where $\boldsymbol{CV}$ is a column vector of ones and $\boldsymbol{LT}$ is a lower-triangular matrix of ones.

Using this formulation, we can write an equation for the z_K constraint

$$z_K = z_0 - \frac{\Delta t}{Q} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \boldsymbol{x} = z_{\text{end}},$$

or, in the prescribed format for nonlinear programming,

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix}}_{\boldsymbol{A}_{\text{eq}}} \boldsymbol{x} = \underbrace{\frac{Q}{\Delta t}\left(z_0 - z_{\text{end}}\right)}_{\boldsymbol{b}_{\text{eq}}}.$$

The limit $z_{\min} \le z_k$ can be written as

$$\underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\boldsymbol{CV}} z_{\min} \le \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\boldsymbol{CV}} z_0 - \frac{\Delta t}{Q} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix}}_{\boldsymbol{LT}} \underbrace{\begin{bmatrix} i_0 \\ i_1 \\ \vdots \\ i_{K-1} \end{bmatrix}}_{\boldsymbol{x}}$$

$$(\boldsymbol{CV})(z_{\min} - z_0) \le -\frac{\Delta t}{Q}(\boldsymbol{LT})\,\boldsymbol{x}$$
$$(\boldsymbol{LT})\,\boldsymbol{x} \le \frac{Q}{\Delta t}(\boldsymbol{CV})(z_0 - z_{\min}).$$

Similarly, $z_k \le z_{\max}$ can be written as

$$-(\boldsymbol{LT})\boldsymbol{x} \le \frac{Q}{\Delta t}(\boldsymbol{CV})(z_{\max} - z_0).$$

Putting the last two constraints together gives

$$\underbrace{\begin{bmatrix} \boldsymbol{LT} \\ -\boldsymbol{LT} \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{x} \le \underbrace{\frac{Q}{\Delta t}\begin{bmatrix} (\boldsymbol{CV})(z_0 - z_{\min}) \\ (\boldsymbol{CV})(z_{\max} - z_0) \end{bmatrix}}_{\boldsymbol{b}}.$$

The constraints on input current can be satisfied by setting

$$\boldsymbol{lb} = -I_{\max}(\boldsymbol{CV}), \qquad \text{and} \qquad \boldsymbol{ub} = I_{\max}(\boldsymbol{CV}).$$

Then, all that's left is to specify the cost function $f(\boldsymbol{x})$. There are no nonlinear constraints in this problem. Given what we've seen in this chapter, we might consider j_s to represent the SEI growth model, or the overcharge model, or the sum of both.

7.8 Fast-charge example

To illustrate the nonlinear-programming method, we investigate controllers designed to determine optimized charging strategies using the degradation models generated to date. In the first control scenarios, the cell initially had a specified SOC value between 10 % and 90 %, and the charger was required to charge the cell optimally from this initial state to a 90 % SOC over a period of two hours. Cell SOC was not allowed outside the range of 10 % to 90 %, but current was unconstrained.

We first looked at using only the SEI-growth degradation model in the control strategy. Results are plotted in Fig. 7.14. The top frame shows the optimal profile of current versus time (positive current is discharge), and the bottom frame shows a profile of cell SOC versus time.

The results may be surprising. The optimal charging strategy when using this degradation model is to discharge the cell quickly to its minimum permitted SOC, wait as long as possible, and then charge the cell quickly to the desired end-point SOC. The cost of discharging plus charging turns out to be less than the cost of maintaining a high SOC for an extended period of time. Referring back to the top frame of Fig. 7.4 helps us see why this is the case. Resting at

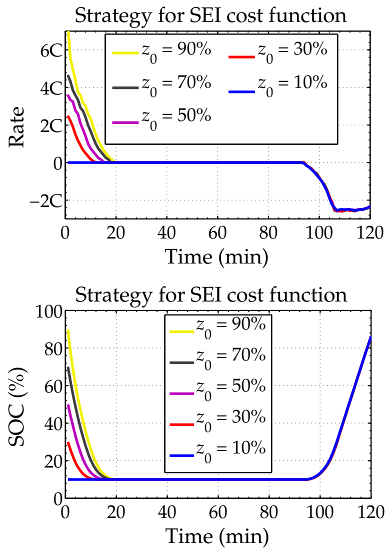


Figure 7.14: Fast-charge strategy using SEI degradation model.

a high SOC has a much faster degradation rate than discharging and then charging back to that SOC.

Next, we look at the same problem but we change the cost function $f(x)$ to add together degradation from SEI growth and from lithium deposition on overcharge. The results are plotted in Fig. 7.15. These are qualitatively similar, but different in some details. In particular, the final charge event is at a much lower rate, and charge current tapers off at high SOC to avoid lithium plating.

Fig. 7.16 overlays as a red line a parametric plot of the optimal charging trajectory from 10 % SOC to 90 % SOC on the combined degradation function of SEI growth plus lithium deposition on overcharge. The lowest cost is achieved by moving fairly directly from the starting point to a point near a 2C charge rate near 30 % SOC. Then, the trajectory abruptly changes course to avoid the very rapid degradation that would occur if it continued in that direction, and follows the edge between regions of the cost function of moderate versus severe degradation.

Third, we looked at fast-charging strategies, with results plotted in Fig. 7.17. The top frame shows results when using the SEI film-growth ROM as the cost function; the bottom frame shows results when using the sum of the SEI and lithium-plating models. In these experiments, the cell was set to an initial SOC of 50 %, then was allowed 15, 30, 45, 60, 75, 90, 105, or 120 minutes to charge to 90 %.

Again, the two strategies are similar, but not identical. If sufficient time is granted, the charger will discharge the cell to the minimum allowed SOC, and then charge the cell. If less time is granted, the charger will discharge the cell only partially before charging. If even less time is granted, charger charges cell immediately.

Before leaving this section, we have to ask, "Are any of these results realistic?" Yes and no. They are realistic in the sense that the SEI and overcharge models are realistic and so, if we ignore all other kinds of degradation, these profiles of current versus time are the best we can do to prolong the life of the battery. However, it is unlikely that any application would actually fully discharge a cell before later recharging it to full capacity. For example, if we think that we can let our electric vehicle charge overnight, but then have some emergency in the middle of the night where we need to use the vehicle, it would not be acceptable to have the battery at its lowest possible SOC at that point. Therefore, it is unlikely that we would follow these optimal charging profiles exactly. However, they can at least guide our charging practices. We now know that charging a battery pack to its top state before it is necessary to do so shortens its life. So, we might instead choose to charge first to an intermediate SOC that guarantees some acceptable minimum vehicle range, then

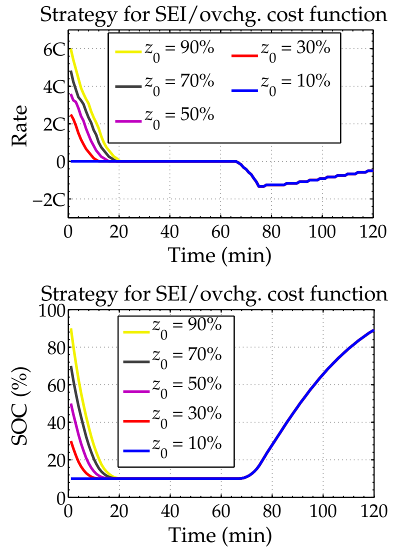


Figure 7.15: Fast-charge strategy using SEI plus lithium-plating degradation model.

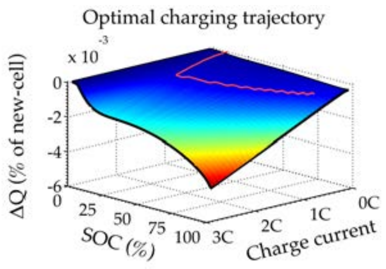


Figure 7.16: Optimal charging trajectory.

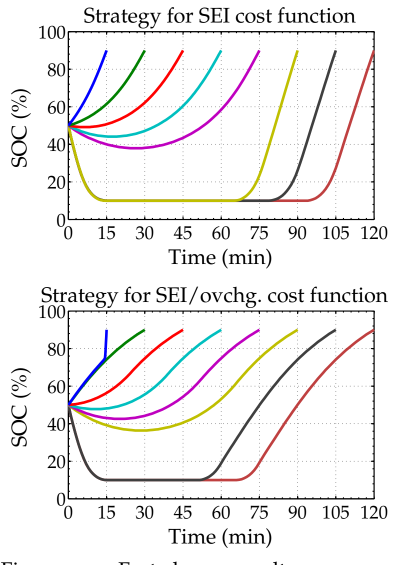


Figure 7.17: Fast-charge results.

18:59:59.

wait as long as possible, and then charge to full capacity only when needed.

7.9 *Dynamic power calculation*

The second possible application of optimized controls using physics-based models of degradation considers the problem of determining dynamic power limits. This is the same problem as considered in Chap. 6, except that now we remove voltage limits from consideration and instead substitute physics-based degradation considerations.

That is, we wish to find the maximum levels of discharge and charge power, based on present battery-pack conditions, that may be maintained constant for ΔT seconds without violating preset design limits on SOC, or maximum design power or current, all while avoiding lithium plating and giving an acceptable tradeoff between performance and capacity loss due to SEI growth. As before, we handle this problem by first looking for the maximum discharge and charge currents that the cell can withstand, and then convert those values to power by multiplying by voltage.

The method we propose here is not yet thoroughly tested but is based on established methodologies used in numerous other applications of modern control theory. With some work, we believe that it will give good results.

It is closely related to a control-system design paradigm called *model predictive control (MPC)*. The idea is to:

- Determine a $k_{\Delta T}$-length sequence of control signals using a model of the system to be controlled, to predict future system performance that will cause the system's controlled variables to converge toward desired values;
- Implement only the first element in this sequence;
- Repeat.

This allows us, for example, to predict a constant-current input that would not violate limits and would optimize a cost function if applied for the full ΔT seconds ($k_{\Delta T}$ sample periods), but only implement the first of these, then repeat. This is exactly the same paradigm as we assumed for voltage-based power-limits calculations in Chap. 6.

Standard MPC is a little different from what we will look at here. Standard MPC reformulates the system model to use changes to the input signal $\Delta \boldsymbol{u}_k$ as input to the model rather than the direct input signal $\boldsymbol{u}_k$ itself. This formulation implicitly adds an integrator to the dynamics, which is good for conventional applications of feedback control because it eliminates steady-state tracking error, but is unnecessary for power-limits estimation. It is also well-suited for setpoint control: when $\Delta \boldsymbol{u}_k = 0$, then $\boldsymbol{u}$ is a constant and the system output $\boldsymbol{y}$ approaches a steady-state constant. Again, this is not necessary for power estimation. Standard MPC also does not allow the state-space model to have a direct feedthrough "$\boldsymbol{D}$" term, which we need here since cell ohmic resistance is captured in this term and is vital to our calculations.

Rather than reformulating our models to use standard MPC, we reformulate MPC to use our models. We will use a similar idea to MPC, leading up to the same form of quadratic optimization used by MPC. The system model we assume is:

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k\\ \boldsymbol{y}_k &= \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k,\end{aligned}$$

where $\boldsymbol{y}_k$ are the performance variables that we would like to control to some limit or to maintain within some hard constraints. That is, $\boldsymbol{y}_k$ may be different from the normal system outputs that we have called $\boldsymbol{y}_k$ in the past. To determine physics-based power limits, we would want to include SOC and locally linearized models of lithium plating and SEI growth in this performance output.

To develop the method, we define the vectors of future inputs and performance variables:

$$\begin{aligned}\boldsymbol{U} &= \begin{bmatrix} \boldsymbol{u}_k & \boldsymbol{u}_{k+1} & \cdots & \boldsymbol{u}_{k+k_{\Delta T}} \end{bmatrix}^T\\ \boldsymbol{Y} &= \begin{bmatrix} \boldsymbol{y}_k & \boldsymbol{y}_{k+1} & \cdots & \boldsymbol{y}_{k+k_{\Delta T}} \end{bmatrix}^T.\end{aligned}$$

Then, by recursively evaluating the state-space model we write,

$$\underbrace{\begin{bmatrix} \boldsymbol{y}_k \\ \boldsymbol{y}_{k+1} \\ \boldsymbol{y}_{k+2} \\ \vdots \\ \boldsymbol{y}_{k+k_{\Delta T}} \end{bmatrix}}_{\boldsymbol{Y}} = \underbrace{\begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A} \\ \boldsymbol{C}\boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}^{k_{\Delta T}} \end{bmatrix}}_{\boldsymbol{F}} \boldsymbol{x}_k + \underbrace{\begin{bmatrix} \boldsymbol{D} & & & \boldsymbol{0} \\ \boldsymbol{C}\boldsymbol{B} & \boldsymbol{D} & & \\ \boldsymbol{C}\boldsymbol{A}\boldsymbol{B} & \boldsymbol{C}\boldsymbol{B} & & \\ \vdots & \vdots & \ddots & \\ \boldsymbol{C}\boldsymbol{A}^{k_{\Delta T}-1}\boldsymbol{B} & \boldsymbol{C}\boldsymbol{A}^{k_{\Delta T}-2}\boldsymbol{B} & \cdots & \boldsymbol{D} \end{bmatrix}}_{\boldsymbol{\Phi}} \underbrace{\begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{u}_{k+1} \\ \boldsymbol{u}_{k+2} \\ \vdots \\ \boldsymbol{u}_{k+k_{\Delta T}} \end{bmatrix}}_{\boldsymbol{U}}$$

$$\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{\Phi}\boldsymbol{U}.$$

We define a desired reference trajectory vector $\boldsymbol{R}_s$ that we would like $\boldsymbol{Y}$ to match as closely as possible, and penalty matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ that weight the importance of tracking this trajectory versus keeping control inputs small. Then, we can formulate a cost function that we wish to minimize:

$$J = (\boldsymbol{R}_s - \boldsymbol{Y})^T \boldsymbol{Q} (\boldsymbol{R}_s - \boldsymbol{Y}) + \boldsymbol{U}^T \boldsymbol{R} \boldsymbol{U}$$

$$
\begin{aligned}
&= (\boldsymbol{R}_s - [\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{\Phi}\boldsymbol{U}])^T \boldsymbol{Q}(\boldsymbol{R}_s - [\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{\Phi}\boldsymbol{U}]) + \boldsymbol{U}^T\boldsymbol{R}\boldsymbol{U} \\
&= \boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{R}_s - \boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{F}\boldsymbol{x}_k - \boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{U} \\
&\quad - \boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{R}_s + \boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{U} \\
&\quad - \boldsymbol{U}^T\boldsymbol{\Phi}^T\boldsymbol{Q}\boldsymbol{R}_s + \boldsymbol{U}^T\boldsymbol{\Phi}^T\boldsymbol{Q}\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{U}^T\boldsymbol{\Phi}^T\boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{U} + \boldsymbol{U}^T\boldsymbol{R}\boldsymbol{U}.
\end{aligned}
$$

To simplify this, note that each term is a scalar, and hence equal to its own transpose. So,

$$
\begin{aligned}
J = {}& [\boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{R}_s - 2\boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{F}\boldsymbol{x}_k] \quad \text{(not a function of } \boldsymbol{U}) \\
&+ 2[\boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{\Phi} - \boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{\Phi}]\boldsymbol{U} \\
&+ \boldsymbol{U}^T[\boldsymbol{\Phi}^T\boldsymbol{Q}\boldsymbol{\Phi} + \boldsymbol{R}]\boldsymbol{U}.
\end{aligned}
$$

Let,

$$
\begin{aligned}
\boldsymbol{H} &= 2[\boldsymbol{\Phi}^T\boldsymbol{Q}\boldsymbol{\Phi} + \boldsymbol{R}] \\
\boldsymbol{f}^T &= 2(\boldsymbol{x}_k^T\boldsymbol{F}^T\boldsymbol{Q}\boldsymbol{\Phi} - \boldsymbol{R}_s^T\boldsymbol{Q}\boldsymbol{\Phi}).
\end{aligned}
$$

Then,

$$J = \frac{1}{2}\boldsymbol{U}^T\boldsymbol{H}\boldsymbol{U} + \boldsymbol{f}^T\boldsymbol{U} + \text{constant}.$$

Further, we can put constraints on $\boldsymbol{Y}$ via

$$\boldsymbol{Y}_{\text{min}} \le \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{\Phi}\boldsymbol{U} \le \boldsymbol{Y}_{\text{max}},$$

which can be written as

$$
\begin{aligned}
\boldsymbol{\Phi}\boldsymbol{U} &\le [\boldsymbol{Y}_{\text{max}} - \boldsymbol{F}\boldsymbol{x}_k] \\
-\boldsymbol{\Phi}\boldsymbol{U} &\le [\boldsymbol{F}\boldsymbol{x}_k - \boldsymbol{Y}_{\text{min}}],
\end{aligned}
$$

which can both be combined in the matrix inequality

$$\underbrace{\begin{bmatrix} \boldsymbol{\Phi} \\ -\boldsymbol{\Phi} \end{bmatrix}}_{\boldsymbol{A}_{\text{ineq}}} \boldsymbol{U} \le \underbrace{\begin{bmatrix} \boldsymbol{Y}_{\text{max}} - \boldsymbol{F}\boldsymbol{x}_k \\ F\boldsymbol{x_k} - \boldsymbol{Y}_{\text{min}} \end{bmatrix}}_{\boldsymbol{b}_{\text{ineq}}},$$

or, $\boldsymbol{A}_{\text{ineq}}\boldsymbol{U} \le \boldsymbol{b}_{\text{ineq}}$.

So, we now have defined vectors and matrices $\boldsymbol{H}$, $\boldsymbol{f}^T$, $\boldsymbol{A}_{\text{ineq}}$, and $\boldsymbol{b}_{\text{ineq}}$ that match a quadratic programming problem, which is

$$\boldsymbol{U}^* = \arg\min \frac{1}{2}\boldsymbol{U}^T\boldsymbol{H}\boldsymbol{U} + \boldsymbol{f}^T\boldsymbol{U}$$

such that

$$\boldsymbol{A}_{\text{ineq}}\boldsymbol{U} \le \boldsymbol{b}_{\text{ineq}}.$$

The solution can be found via `quadprog.m` using MATLAB's Optimization Toolbox™. Note, we can use

$$\boldsymbol{U} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix}^T \boldsymbol{u}$$

to make a fast single-variable optimization problem, to give us the maximum discharge and charge current values that would apply to all times.

But, what values to use for the design constants? When computing charge-power limits, we set reference $\boldsymbol{R}_s$ for the SOC state to 1.0; when computing discharge-power limits, we set it to 0.0. We further need to augment our standard ESC cell model with additional states that describe locally linearized SEI growth; then, we must put constraints on these states to minimize degradation. Further, we must augment the model with additional performance outputs that predict locally linearized lithium plating so that we can put hard constraints on this output to prevent lithium plating in the negative electrode. This remains a research opportunity at this point.

7.10 *Where to from here?*

We have now reached the end of this volume on battery-management algorithms based on equivalent-circuit models of battery cells. We've come a long way in our understanding!

- We have seen the major functional requirements of a BMS. These include a need to be able to sense battery-stack and individual cell voltages, as well as module temperatures and battery-pack current. Further, the BMS must be able to sense loss of isolation and to control the contactors that connect the battery pack to its load. The electronics and controls of the battery pack must be designed to protect the operator and the battery pack itself in both normal and abuse scenarios. The BMS must be able to communicate with its environment to control the charger, transmit operational limits to the load controller, and to report logged abnormal events to a service technician. Its algorithms must be able to estimate cell SOCs, states of health, total energy and available power.
- We have reviewed the formulation of equivalent-circuit models of cells and have seen how to use these models to simulate battery packs comprising cells wired in arbitrary configurations. In particular, we saw that battery packs made from series-cell modules behave differently from packs made from parallel-cell modules, even if they have the same number of cells wired in series and in parallel. Understanding how the overall battery pack will behave, particularly in fault situations, is important to the BMS designer. In addition, it is important to be able to cosimulate the battery pack and its load. We explored the relatively simple example of an electric-vehicle load, but any other load could also be modeled and simulated along with the battery pack.

- The majority of this book explored algorithms designed to perform the estimation and control tasks of the BMS. We looked first at SOC estimation and saw that simple methods tend to behave poorly. So, we then invested a considerable amount of time to understand how model-based estimation methods work, and focused in particular on extended and sigma-point Kalman filters as methods to estimate the state vector of an equivalent-circuit model of a battery cell. Both of these methods can work quite well in practice, with the sigma-point Kalman filter having a slight performance edge and extended Kalman filters having a slight complexity advantage. We also looked at some very practical ways to ensure that these algorithms give robust predictions without user intervention, including approaches to handling sensor faults and biases. The bar-delta approach to co-estimating all cell SOCs in a battery pack was shown to reduce the computational requirements of the BMS processor greatly.
- We then looked at SOH estimation. First, we discussed the dominant physical causes of cell degradation and saw that the most easily measurable indicators of aging are cell total capacity and resistance. We discovered that the sensitivity of cell voltage to resistance is relatively high, implying that it should be fairly easy to estimate cell resistance. We developed a simple method to do so that works well. We also saw that sensitivity of voltage to total capacity is very low, meaning that it is much more difficult to estimate total capacity. We quickly explored joint and dual nonlinear Kalman filters as methods that could be used to estimate the entire degradation state of the cell, but spent most of our time investigating some simpler methods based on regression techniques. Standard least-squares regression was shown to produce biased results, but methods based on total-least-squares were proposed that worked much better and produced estimates that were accompanied by confidence intervals and a goodness-of-fit metric that allowed the application to know whether the estimates are reliable.
- Next, we looked at cell balancing. First, we saw the factors that lead to long-term imbalance in a battery pack and some that might be expected to do so but actually do not. We considered some questions to address when designing a balancing system and some circuit concepts that could be used to equalize cells. A simulation example showed that dissipative balancing can be sufficient if the objective is only to keep the pack in long-term balance. However, if we additionally want to use balancing circuitry to maximize the energy and power output of the battery pack each cycle and to extend life by differentially processing power on a cell-by-cell basis, then nondissipative balancing will be required.

- We returned to the question of power-limit estimation, which we had quickly previewed in the first chapter. We saw that voltage-based HPPC power limits could easily be augmented with SOC, maximum-design current, and maximum-design power limits as well. We then replaced the simple cell model assumed by the HPPC method with the full ESC cell model and saw how a bisection algorithm could rapidly compute a power limit that more accurately represented the actual dynamic capabilities of the cell.
- Finally, we introduced the idea of physics-based power limits to more directly optimize a tradeoff between the performance of a battery pack and its rate of degradation. While we cannot do a full investigation of this topic using equivalent-circuit cell models as a basis, we found that we could make simplified 0D ROMs of some degradation mechanisms and use these inside of controls to compute fast-charge and dynamic power limits.

This book has presented state-of-the-art methods in battery-management algorithms using equivalent-circuit cell models as a basis. These algorithms can work very well and are representative of the methods used in practically every fielded BMS at this point. However, there are still gains to be made, principally in being able to compute power limits using physics-based models to more directly control tradeoffs between performance and degradation rate. The planned third volume of this series, *Battery Management Systems: Volume III, Physics-Based Methods* will explore these topics.

- We first review the continuum-scale physics-based model from Volume I, but reformulate the model equations in terms of more readily identifiable quantities. We then show how the model-order reduction approach from Volume I can be applied to this reformulated model and how to simulate cells and battery packs with this new ROM.
- An enormous challenge when wishing to use physics-based models is "How do we find the parameter values?" That is, what are the conductivities, diffusivities, and other material properties required to simulate the model equations. Historically, the process of *system identification* has been performed by trained scientists who tear down the cells and measure the quantities directly. We will show how to find the required values using simple cell-level laboratory tests instead, without the need for cell teardown.
- A BMS that uses physics-based models must perform the same tasks as one that uses equivalent-circuit based models. Thus, for example, we need to perform state estimation. We show how to apply the nonlinear Kalman filters developed in the present book

to the physics-based models, which requires an additional step beyond what we have seen here.

- SOH estimation using physics-based models can yield the same kinds of information that we have seen in this book but can additionally detect how the stoichiometric operating windows have shifted in each electrode. This is valuable input to power-limit calculation, because voltages and internal electrochemical potentials that an aged cell can withstand are often different from those of a new cell. This enables maximizing battery-pack usage throughout the lifetime of the pack.
- Most important, we can bring powerful methods to bear on the problem of power-limit estimation. We will look at some reduced-order 1D physics-based degradation models that give better performance than those presented in this book and will see how to compute power limits using model predictive control to avoid lithium plating and to minimize other kinds of aging as well.

In closing, I hope that some of the material covered in this book has sparked your imagination and that it will enable you to contribute to making BMSs of the future even better!

Appendix: Parameters used for SEI simulations

The following is adapted from Table 1 in Randall, A.V., Perkins, R.D., Zhou, X., Plett, G.L., "Controls Oriented Reduced Order Modeling of SEI Layer Growth," *Journal of Power Sources*, 209, 2012, pp. 282–288.

Symbol	Units	Neg. electrode	Separator	Pos. electrode
L	μm	88	20	80
R	μm	2	—	2
A	m^2	0.0596	0.0596	0.0596
σ	$\mathrm{S\,m^{-1}}$	100	—	100
ε_s	—	0.49	—	0.59
ε_e	—	0.485	1	0.385
brug	—	4	—	4
$c_{s,\max}$	$\mathrm{mol\,m^{-3}}$	30,555	—	51,555
$c_{e,0}$	$\mathrm{mol\,m^{-3}}$	1,000	1,000	1,000
$\theta_{\min}$	—	0.03	—	0.95
$\theta_{\max}$	—	0.886	—	0.487
D_s	$\mathrm{m^2\,s^{-1}}$	3.9×10^{-14}	—	1.0×10^{-14}
D_e	$\mathrm{m^2\,s^{-1}}$	7.5×10^{-10}	7.5×10^{-10}	7.5×10^{-10}
t_+^0	—	0.363	0.363	0.363
k	$\mathrm{A\,m^{5/2}\,mol^{-3/2}}$	4.854×10^{-6}	—	2.252×10^{-6}
α	—	0.5	—	0.5
U_s	V	0.4	—	—
R_{SEI}	$\Omega\,\mathrm{m}^2$	0.01	—	—
$i_{0,s}$	$\mathrm{A\,m^{-2}}$	1.5×10^{-6}	—	—

Appendix: Parameters used for overcharge simulations

The following is adapted from Table 1 in Perkins, R.D., Randall, A.V., Zhou, X., Plett, G.L., "Controls Oriented Reduced Order Modeling of Lithium Deposition on Overcharge," *Journal of Power Sources*, 209, 2012, pp. 318–325.

Symbol	Units	Neg. electrode	Separator	Pos. electrode
L	μm	85	76.2	179.3
R	μm	12.5	—	8.5
A	m^2	1	1	1
σ	S m^{-1}	100	—	3.8
ε_s	—	0.59	—	0.534
ε_e	—	0.36	1	0.416
κ_e	S m^{-1}	0.2875	0.2875	0.2875
brug	—	1.5	—	1.5
$c_{s,\max}$	mol m^{-3}	30,540	—	22,860
$c_{e,0}$	mol m^{-3}	1,000	1,000	1,000
$\theta_{\min}$	—	0.10	—	0.95
$\theta_{\max}$	—	0.90	—	0.175
D_s	$\text{m}^2\,\text{s}^{-1}$	2.0×10^{-14}	—	1.0×10^{-13}
D_e	$\text{m}^2\,\text{s}^{-1}$	7.5×10^{-11}	7.5×10^{-11}	7.5×10^{-11}
t_+^0	—	0.363	0.363	0.363
k	$\text{A m}^{5/2}\,\text{mol}^{-3/2}$	2×10^{-6}	—	2×10^{-6}
α	—	0.5	—	0.5
α_s	—	0.7	—	—
U_s	V	0.0	—	—
R_{SEI}	$\Omega\,\text{m}^2$	0.002	—	—
$i_{0,s}$	A m^{-2}	10	—	—

18:59:59.

18:59:59.

About the Author

Gregory L. Plett is a professor of electrical and computer engineering at the University of Colorado Colorado Springs (UCCS). His research since 2001 has been focused on applications of control-systems theory to the management and control of high-capacity battery systems such as found in hybrid and electric vehicles. Current investigations include physics-based reduced-order modeling of ideal lithium-ion dynamics, system identification of physics-based model parameters using only current–voltage input–output data, physics-based reduced-order modeling of degradation mechanisms in electrochemical cells, estimation of cell internal state and degradation state, state-of-charge, state-of-health, and state-of-life estimation, power and energy prediction to extend life, and battery pack fast charging. Research is both theoretical and empirical: the UCCS High-Capacity Battery Research and Test Laboratory houses equipment to test cells, modules, and battery packs and is home to custom battery-management system and hardware battery-pack simulator projects, which together enable cutting-edge research in advanced but practical algorithm prototyping. He offers courses in control systems and in battery modeling and management to support this research. Dr. Plett holds a bachelor of engineering in computer systems engineering degree from Carleton University (Ontario, Canada) and MS and Ph.D. degrees in electrical engineering from Stanford University (Stanford, California). He is a senior member of the Institute of Electrical and Electronic Engineers and life member of the Electrochemical Society.

18:59:59.

Index

18:59:59.

18:59:59.