Stochastic Optimal Control of Lithium-ion Battery Operations

Pedret Sagnier, Berta

In partial fulfillment of the requirements for the degree in Telecommunications Technologies and Services Engineering and the degree in Engineering Physics

Advisor: Richard D Braatz

Tutor: Arnau Doria Cerezo

Cambridge, MA

May 2022

Abstract

Optimal charging of lithium-ion batteries maximizes battery life while ensuring fast charging and safe usage. The properties of manufactured batteries can differ from design values and change over time due to degradation. The limitation of past battery simulations is that they use fixed deterministic values for these parameters that may not be accurately known. Stochastic analysis includes real-world uncertainties in simulations to represent this manufacturing variation. This study aims to propagate the uncertainty of model parameters onto output states, such as voltage or cell temperature. Output uncertainty is characterized by these states' upper and lower bounds.

This research integrates stochastic uncertainties in optimal battery control by using probabilistic distributions to define model parameters, such as the ambient temperature. The uncertainty propagation is then performed using linear sensitivity analysis. The linearized sensitivity is validated using Monte Carlo with several hundreds of replicates, proving that sensitivity analysis is significantly less computationally expensive. A methodology is designed to quantify uncertainty propagation in lithium-ion batteries for any set of probabilistic parameters and optimal charging paths. This methodology computes linear sensitivities on any system of differential-algebraic equations. For battery modeling, it can accurately compute sensitivities on mixed continuous-discrete simulations, solving typical issues found with following discrete stages and the control of non-measurable states.

The methodology given is a powerful tool for stochastic battery simulations. It can help redefine accurate optimal charging paths for future onboard real-time applications and determine safer manufacturing specifications. Multiple case studies are presented to validate this methodology, including reaction kinetics and optimal charging paths. The examples analyzed consider how an uncertain ambient temperature affects the battery's voltage, temperature, and state of health for relevant optimal charging protocols.

Keywords: sensitivity analysis, uncertainty quantification, stochastic parameters, optimal control, battery modeling, lithium-ion batteries.

Resum

El control òptim en bateries de ions de liti maximitza la vida útil de la bateria alhora que garanteix la càrrega ràpida i un ús segur. Les propietats de les bateries fabricades poden diferir del valor de disseny i canviar amb el temps en degradar-se. La limitació d'anteriors simulacions de bateries és que utilitzen valors deterministes per aquests paràmetres que no es coneixen amb precisió. L'anàlisi estocàstic inclou aquestes incerteses d'aplicacions reals en simulacions. Després d'integrar les incerteses, l'objectiu és quantificar-ne la propagació per veure com afecta als estats finals.

Aquest treball integra incerteses estocàstiques en el control òptim de bateries definint amb distribucions de probabilitat alguns paràmetres, com la temperatura ambient. La propagació de la incertesa es fa utilitzant anàlisi de sensibilitat lineal. S'ha dissenyat una metodologia per quantificar aquesta propagació d'incertesa a les bateries de ions de liti per a qualsevol conjunt de paràmetres incerts i model de càrrega òptim. Aquesta metodologia pot calcular sensibilitats lineals en qualsevol sistema d'equacions discretes i algebraiques. En simulació de bateries, permet calcular sensibilitats en sistemes de càrrega híbrids (continus-discrets) resolent els inconvenients típics d'implementar salts discontinus o controlar estats no observables.

La metodologia dissenyada és una eina versàtil per a simulacions estocàstiques de bateries. Pot redefinir rutes de càrrega òptimes i precises per futures aplicacions en temps real o per determinar noves especificacions de fabricació més segures. Per demostrar la precisió d'aquest mètode, els resultats presenten múltiples casos d'estudi, incloent la cinètica de reacció i els models de càrrega òptims. Els exemples mostren els efectes de definir una temperatura ambient incerta en estats de la bateria com el voltatge, la temperatura o l'estat de salut per protocols rellevants de càrrega òptima.

Paraules clau: anàlisi de sensibilitat, quantificació d'incertesa, paràmetres estocàstics, control òptim, simulació de bateries, bateries de ions de liti.

Resumen

El control óptimo de baterías de iones de litio maximiza la vida útil de la batería garantizando también la carga rápida y un uso seguro. Las propiedades de las baterías al fabricarse pueden diferir de los valores de diseño o cambiar con el tiempo por degradación. La limitación de anteriores simulaciones de baterías de litio es que utilizan valores deterministas para parámetros que no se conocen con precisión. El análisis estocástico incluye estas incertidumbres reales en las simulaciones. Después de integrar las incertidumbres, el objetivo es cuantificar su propagación para saber cómo afectan a los estados finales.

Esta investigación integra incertidumbres estocásticas en el control óptimo de baterías definiendo como distribuciones de probabilidad algunos parámetros del modelo, como la temperatura ambiente. La propagación de la incertidumbre se implementa utilizando análisis de sensibilidad lineal. Se ha diseñado una metodología para cuantificar esta propagación de incertidumbre en baterías de iones de litio para cualquier conjunto de parámetros inciertos y cualquier modelo de carga óptima. Esta metodología permite calcular sensibilidades lineales en cualquier sistema de ecuaciones discretas y algebraicas. Para simulaciones de baterías, puede calcular sensibilidades en sistemas de carga híbridos (continuos-discretos) resolviendo limitaciones comunes de implementar saltos discontinuos y controlar estados no observables.

La metodología utilizada es una herramienta versátil para simulaciones de baterías estocásticas. Puede redefinir rutas de carga óptimas y precisas para futuras aplicaciones en tiempo real o para determinar nuevas especificaciones de fabricación más seguras. Para demostrar la precisión de este método, los resultados presentan múltiples casos de estudio, incluida la cinética de reacción y modelos de carga óptimos. Los ejemplos definen una temperatura ambiente incierta y consideran su efecto en estados de la batería como el voltaje, la temperatura o el estado de salud en protocolos relevantes de carga óptima.

Palabras clave: análisis de sensibilidad, cuantificación de incertidumbres, parámetros estocásticos, control óptimo, simulación de baterías, baterías de iones de litio.

Nomenclature

The next list describes the symbols used within the body of the document:

Symbols

ψ	State of battery model	k	Reaction rate constant
σ_x	Standard deviation of x	V_x	Variance (matrix) of x
θ	Vector of uncertain parameters	**M(t)**	Sensitivity matrix
D_s	Solid-phase diffusion coefficient	L(t)	Sensitivity vector

Constant Charging Modes

CC	Current $[A]$	CP	Power $[W]$
CCe	electrolyte concentration $[kmol/m^3]$	CPo	Li Plating over-potential $[V]$
$CCss$	solid surface concentration $[kmol/m^3]$	CV	Voltage $[V]$

Acronyms

BMS	Battery Management System	NMPC	Nonlinear Model Predictive Control
CFD	Central Finite Differences	PET	Porous Electrode Theory
DAE	Differential-Algebraic Equations	SA	Sensitivity Analysis
FVM	Finite Volume Method	SOC	State of Charge
LCO	Lithium Cobalt Oxide	SOH	State of Health
LIB	Lithium Ion Battery	UQ	Uncertainty Quantification

Acknowledgments

First of all, I would like to express my deepest gratitude to my advisor Prof. Richard D. Braatz, for granting me this wonderful opportunity to work within his group. I appreciate his support, motivation, immense knowledge, and the fantastic and passionate community he has built here. I would also like to thank Marc D. Berliner for his dedicated work in battery modeling, which served as a perfect ground to start my research. I'm grateful for his work on this project, his wisdom, and the support he has given me.

Secondly, I would like to thank Prof. Arnau Doria for his advice and guidance during my research project. I appreciate his suggestions and insightful comments during our online meetings and his feedback support on the writing of this thesis.

Additionally, I want to thank the Massachusetts Institute of Technology, the Centre de Formació Interdisciplinaria Superior (CFIS), and the CELLEX foundation for making it possible and for their economic support.

Last but not least, I'm thankful for all of my friends and family's encouragement. I'm grateful for my Braatz group colleagues, who have shared tireless hours and limitless food with me in the office. I appreciate the presence of Pep, Guillem, and Hector in this incredible Boston experience. I especially want to thank David, Marina, and Louisa for their constant emotional support; it would not have been the same without all of you.

Thank you, and see you soon!

Berta

Contents

List of Figures

Chapter 1

Introduction

This research provides stochastic optimal control of lithium-ion batteries for charging and discharging operations through software simulations. Optimal control allows detecting paths that ensure fast charging times while potentially increasing battery lifetime. This work extends previous publications that successfully found fast battery charging methods using only deterministic parameters. Here, stochastic analysis is implemented to account for the intrinsic uncertainty of the battery model's parameters. Stochastic optimal control allows defining some time-invariant parameters as probability distribution functions to characterize later the effect of this dispersion on the control results. The uncertainty quantification on the output is done through parametric sensitivity analysis and allows describing the functionality ranges and back-off specifications of all the studied battery operations.

1.1 Context and motivation

Battery research has focused on finding chemical compositions with short charging times and high energy density while keeping safety constraints and long durability. Electrochemical power sources have evolved significantly in the last years by optimizing design, materials, operating ranges, and production costs. Lately, there has been a significant focus on optimizing how the battery is used once the chemistry is decided, for example, regarding lithium-ion batteries. Research is looking for methods to best charge, use, and discharge batteries that extend their lifetime (implying lower environmental impact), allow fast charging, and maximize energy density. Using mathematical models for simulation and controlling battery operations enable us to

study these issues quickly, safely, and with lower costs than physical experiments. It is an efficient method to project battery cycles for extended periods, determine their future perspectives, and see how degradation affects multiple operation paths.

Lithium-ion batteries are now ubiquitous in technology such as cell phones, laptops, and electric vehicles (EV). Compared to alternative technologies, they have proven to have one of the best energy density ratios with reasonable cost and specifications such as no memory effect, high voltage, and low self-discharge. They also represent a realistic alternative to fossil fuels which can change the environmental impact that technology is causing. As government incentives to reduce emissions are rising, the importance of batteries keeps growing, especially for automation, aerospace applications, and green energy storage. For a future wide adoption of EVs, batteries must be safe, cheap, allow for long driving ranges, and charge quickly with minimal degradation [7].

Optimal charging problems for lithium-ion batteries can help this purpose, aiming to minimize charging time while maximizing the battery lifetime. An accurate physical and chemical model of the Li-ion battery working principles is required for simulation and as a base for optimal charging control models. There exist multiple technologies devoted to battery modeling in different programming languages with diverse strengths and potentials. The software used for modeling in this research is the *PETLION.jl* [1] library made by M. Berliner, which models Li-ION batteries based on Porous Electrode Theory (PET) in the Julia programming language.

Some fast charging paths are determined using this software on [3] which gives interesting insights on possible optimal charging methods that could potentially increase the battery lifetime. A limitation of this previous work is that all the parameters are defined as deterministic values. Manufactured batteries can be diverse and have minor discrepancies in their parameters from cell to cell concerning the desired standard, so a deterministic approach does not suffice. Also, some of the parameters are hard to measure and determine experimentally. In fact, 'most of the effective model parameters in porous electrode theory are not practically identifiable from cycling data for a lithium-ion battery' [2]. Specifically, parameters such as the solid-phase diffusion coefficients ($D_{s,n}$ and $D_{s,p}$, m^2/s) and reaction rate constants (k_n and k_p, $m^{5/2}/(mol^{1/2}s)$) are hard to measure as they are affected by porosity and other cell factors. They are usually modeled as *effective* values which lump together multiple physical properties for simplicity. Since the experimental measurement is difficult, these parameters have been estimated with a broad set of techniques in past literature, giving results spanning numerous orders of magnitude. Stochastic analysis is crucial to understand better how this uncertainty can affect the battery's operations.

A common issue in simulation and physical models is that these give a deterministic approach to problems that are by nature uncertain. Furthermore, when providing real-world applications to these simulated problems, values can usually change due to multiple issues and diverse operating conditions worldwide. It is essential, thus, to include uncertainty issues in the given battery models to provide secure specification ranges of operation and ensure successful usage of batteries based on modeling results. For this purpose, stochastic analysis allows describing parameters as probabilistic distributions. The probabilistic distributions of the model parameters can be defined using the literature-estimated boundaries found. Integrating stochastic analysis on previous modeling technologies will help reduce unexpected dangerous outcomes in battery applications. Furthermore, a better understanding of specification boundaries and reasonable operating conditions could help to limit precise manufacturing standards.

1.2 General objectives

Optimal charging gives helpful insight into battery protocols and provides efficient usage cycles that can extend the battery lifetime. The addition of stochastic uncertainty quantification is required to quantify the effect of uncertain parameters in the operation modes of the battery. Once the parameters are stochastic, the states of the model will also have a probabilistic distribution. The goal of quantification is to determine the shape and bounds of these output distributions, which can help validate or completely change previous deterministic fast-charging results. The information from these statistical methods can ensure a prosperous and safer performance within any boundaries by giving functionality back-off on any state to specify more precise operation ranges.

The goal of this project is to expand previous work done on Optimal Charging of Li-ion batteries in [3] that was based on the PETLION deterministic modeling. The expansion includes stochastic uncertainty quantification using sensitivity analysis for all uncertain coefficients in the electrochemical model. We want to build a mathematical approximation for uncertainty quantification that is computationally efficient and accurate. We expect to quantify the effects of parametric uncertainties on multiple modeling constants and battery operating conditions. The goal is to determine the output probability distribution of any state for any charge paths and set stochastic parameters. Integrating stochastic uncertainties can also help better understand the significance of every uncertain parameter in the evolution of the battery states. The final results should give accurate specifications, be closer to real Li-ion battery applications, and provide

higher assurance that constraints such as maximum temperature or voltage are satisfied.

1.3 Overview

This project analyzes stochastic uncertainties in optimal control over available operating modes of lithium-ion batteries using the Julia programming language.

Chapter 2 describes the fundamentals of Li-ion battery operations, their most common applications, hazards of use, and the necessary chemical and physics background for software simulation using PETLION.

Chapter 3 contains the mathematical background for uncertainty quantification and the methodology used, which gives multiple options to implement the sensitivity analysis over the electrochemical battery simulation.

Chapter 4 describes some preliminary results implementing the mathematical methods into Julia programming. The chapter is divided into four test cases that make different approaches and report the results of simulations' accuracy and computational efficiency. Sensitivity Analysis is implemented using Finite Differences to analyze everyday battery operations' deviations, performance, and boundary constraints.

Chapter 5 employs the analysis from the mathematical background and preliminary results in Julia to describe the stochastic uncertainties effect through more complex PETLION charging paths. Precise operational boundaries and optimal charging constraints are given for Li-ion battery operations with multiple safe applications. The computational cost of each method is detailed, and the accuracy is verified against Monte Carlo sampling.

Finally, Chapter 6 concludes the main findings of this thesis and gives some future work insight.

Chapter 2

State of the art and Applications

2.1 Lithium Ion Batteries

Batteries play an essential role in our everyday lives, both on a household and large industrial scale. The integration of consumer electronics into portable and wearable devices and the rise of hybrid electric vehicles (HEVs) in the last century have emphasized the need for efficient and powerful energy storage devices. Batteries have also stood out as a more sustainable option that can reduce emissions and pollution from other energy sources. The increasing popularity of batteries has brought about massive progress in research and enterprises looking for devices with the highest power density and minimal size and weight. Different chemistries have been developed in the last years to provide energy storage based on Lead, Nickel, Lithium, or Sodium, which express various features. A complete analysis of sustainable battery options, which explains in-depth the features of Li-ion batteries against other alternatives, can be found on [19]. The Ragone chart in Fig. 2.1 serves as an indicator of the specific energy and power ranges available, which are fundamental for battery-electric and hybrid vehicles.

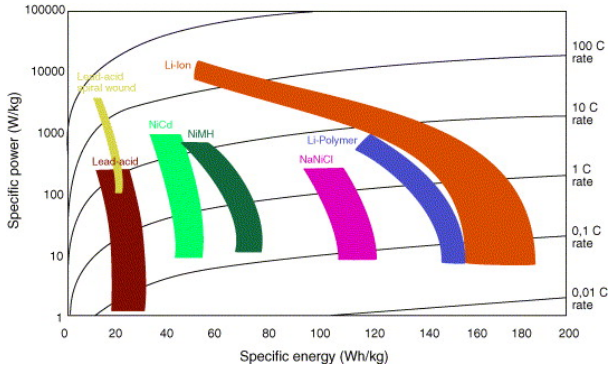

FIGURE 2.1: Ragone chart from 'SUBAT: An assessment of sustainable battery technology' [19]

Overall, among electrochemical accumulators, Li-ion batteries have the lowest environmental impact while providing one of the best trade-offs in terms of power density, cell voltage, low weight, and low self-discharge [19]. These features justify why Li-ion batteries are currently the most popular technology for energy storage as portable electronics become more and more important in our everyday lives. They are present in most phones, laptops, tablets, tools, cameras, drones, smartwatches, and general health-tracking devices. They are also used more and more for personal electric mobility and hybrid and electric vehicles. They are essential for renewable energy storage (such as solar and wind power), making them crucial in developing clean-energy systems. Their wide span of applications requires batteries to be efficient, safe, and have low production costs. At present, Li-ion batteries are performing close to their maximum power/ energy density capacity, so further research is being conducted on the materials, design of LIBs, and efficient usage.

The present relevance of Lithium-Ion Batteries is undoubted. The Nobel Prize in Chemistry 2019 was awarded to John B. Goodenough, M. Stanley Whittingham, and Akira Yoshino "for the development of lithium-ion batteries" in their diverse research which has brought about our current rechargeable future[1].

2.1.1 Battery fundamentals

Lithium-ion batteries can generate electrical energy by converting electrochemical energy into a flux of ions and electrons. Batteries are composed of an anode and cathode (which store the Lithium ions), split by a separator full of an electrolyte solution, and have negative and positive current collectors on each edge. When charging and discharging, the electrolyte carries the positively charged Li-ions between cathode and anode, generating an external flux of electrons that gives current[2]. The flux of ions inside the battery allows charging and discharging the battery multiple times with a low loss of function, which is one of their most significant advantages compared to previous technologies for energy storage and delivery.

When charged, Li^+ ions remain on the anode of the battery, which serves as a neutral storage medium for the charged atoms and electrons (usually graphite). When a load is connected between the current collectors, Li^+ migrates to its cathode (usually a metal oxide such as $LiCoO$), creating an external flux of electrons from the negative to the positive electrode. Each charge and discharge process is known as a cycle of the battery's lifetime. Cycles are associated with an irreversible loss of Li-ions in the battery's material, implying that fewer electrons will be

[1] See details in www.noberlprize.org
[2] Complete description and a diagram of how a Li-ion battery works are available on the U.S. Department of Energy website: www.energy.gov

available in the following cycle, reducing the battery's capacity. This aging process can seriously affect battery lifetime and performance. Aging will be mainly affected by the battery chemistry, the choice of materials and electrolytes, and the usage ranges (which is the focus of this research), such as power consumption and current extraction through the cycles. For example, batteries have faster degradation when charged over 80% state of charge (SOC) because of the high voltages required to reach the maximum charge.

2.1.2 Hazards and use assessment

Lithium-ion batteries have had multiple issues concerning safety in the past in production, storage, and usage. They tend to overheat and damage at high operational voltages, which is why it is so important to have accurate specifications that are correctly bounded even for uncertain parameters. On the one hand, they are currently used in various applications (with very different energy requirements) and very diverse operating conditions such as extreme temperatures. On the other hand, covering all these applications and making batteries more efficient, especially for electric vehicles, has brought lithium-ion chemistry closer to its operational physical limit. This is where battery modeling can make a big difference and why LIBs require safety mechanisms and sensors to ensure they are working within limits, such as Battery Management Systems described below.

Most of the safety issues are associated with the fact that the liquid electrolyte in Lithium-ion batteries is highly flammable [10]. A common failure is caused by *thermal runaway* that occurs when the battery's temperature rises above $\sim 80\,^{\circ}\mathrm{C}$, making the exothermic reactions inside the cell faster which further heats the battery. This process may result in fires or explosions [10]. In other scenarios, the battery could ignite or explode from other hazards such as breaking/crashing or other events that can cause an internal short-circuit of the battery. Extensive research has focused on better understanding the issues that cause these battery hazards and finding optimal materials (internal protection) or sensors/control methods (external protection) that improve LIB's safety [10]. Safety is a crucial step toward broad adoption of Li-ion batteries, especially in the future of electric mobility, and is a required milestone before further increasing their energy density and charge capacity (which could lead to worse hazards if uncontrolled).

To sum up, safety issues with Lithium-ion batteries could be caused by a simple malfunction or manufacturing issue, an external incident such as a crash causing some deformations/perforation, or simply lousy usage and operation. Modeling and stochastic optimal control can easily prevent the latter, especially for onboard applications. Furthermore, the integration of stochastic analysis could help reduce issues with manufacturing and malfunction by setting more

explicit specification bounds from the results of previous studies.

2.1.3 Future prospects

Present Lithium-ion batteries are highly optimized to fit the demanding requirements of the new digital era. They can currently operate for months to years with good function and minor loss of capacity, which is a considerable achievement given that many of the materials operate outside their thermodynamic stability windows [6]. Most advances in battery technology have resulted from years of research with global efforts and extensive investments looking for better-optimized systems. As a result, the modern battery is operating with properties that are getting closer and closer to the fundamental capacity of the materials: "further lithium removal from the cathode results in irreversible structural transformations or oxygen loss, while on the anode no vacancies in the lattice remain to accommodate more Li ions" [6]. The rest of the elements on the battery, such as the current collectors and electrodes, are getting thinner and thinner to improve energy density and reduce costs. It is reasonable to assume that research in this field will continue to evolve in the following years, and many exciting paths could improve current technology. Some paths to improve Li-ion battery standards include layering the cathodes, replacing the common graphite anode with higher storage density structures (such as silicon), or regarding the electrolyte, it is possible to increase its volume fraction or ionic transport capability [6].

There are other research branches devoted to finding alternatives to Lithium-ion systems. One possible path involves changing the traditional liquid electrolyte to others, such as ionic liquids and solid-state batteries (SSBs). Another alternative is enabling anion reduction-oxidation chemistry for Lithium with air or sulfur. Lastly, there can be considered alternative highly electronegative base materials beyond Li, such as Na, Mg, Ca, or Al [6]. Indeed, Elon Musk recently announced Tesla's interest in using manganese on their battery cells to use more accessible materials[3].

2.2 Battery modeling and optimal charging

Having accurate and computationally efficient methods that simulate battery operations and predict long time effects is crucial for improving usage efficiency and extending the lifetime of lithium-ion batteries. Battery cells' aging depends on how their power is consumed through the cycles and the current levels of usage. Advanced Battery Management Systems (ABMS) is responsible for monitoring the state of the batteries to ensure safe and efficient operation in

[3]Tesla Welt Podcast: Giga Berlin Opens speech https://www.youtube.com/watch?v=BDy7K-vOM8g

multiple scenarios. Battery packs that incorporate BMS and some capacity for external communication are 'smart battery packs', which can be charged according to optimal protocols. BMS usually monitors parameters such as the State of Charge (SoC), Depth of Discharge (DoD), State of Health (SoH), and operating temperature [17]. Battery modeling is relevant for accurate BMS, improving the charging/discharging cycles, enhancing short and long-term battery capacity, preventing material damage, and allowing the faster study of LIBs behavior under safe conditions.

According to [17], there are three levels of battery models depending on their degree of physical insight: white-box model (electrochemical), grey-box model (circuit-oriented), and black-box model (based on artificial intelligence, e.g., Neural Networks). The former is the one used on this project as it is the basis of the PETLION software. Electrochemical models can extensively determine Lithium-ion battery chemistry through a set of partial differential equations, making these systems exact. The system usually combines electrical principles and electrochemical equations to describe the system efficiently. "They are a true representation of what occurs in the cell and are known to be very accurate" [17] because this model can mathematically explain microscopic cell behavior. Another advantage is that all states are fully observable within battery modeling compared to real-life measuring, which allows 'virtual measurements' and a deeper understanding of multiple states. The only issue with electrochemical models is that the high level of detail and precision makes them computationally expensive and complicated to configure. Hence, a real-time application does not seem promising.

The Li-ion battery simulator used as the base for the uncertainty quantification in this project is the PETLION software, which stands for Porous Electrode Theory Lithium ION. The motivation behind using PETLION is that this open-source software has been thoroughly optimized in Julia to have fast computational times while simulating battery protocols rigorously. "PETLION is an open-source, high-performance computing implementation of the porous electrode theory (PET) model in Julia. [...] the software is seen to be two orders of magnitude faster than comparable software for some application" [1]. The general study of uncertainties requires running the same model multiple times for perturbed parameters, so a compiled language as Julia and a computationally efficient software are optimal options. The chemical and physical models of the battery in PETLION are built using the PET model.

2.2.1 Porous Electrode Theory (PET)

Porous Electrode Theory was developed around 1975 by John Newman and others as a powerful mathematical model to define battery components and processes. A general description

and multiple applications are given in [13]. PET is currently widely used to model battery cycling behavior. It is based on the porous electrode, which defines the battery's solid-phase material and conductive matrix as porous elements allowing intimate contact with the filling electrolyte solution. This porosity can provide a larger surface area for electrochemical reactions that are intrinsically slow and also store the essential reactants in proximity to the electrode surface, allowing high-rate discharge [13]. PET describes electrochemical kinetics (more accurate for battery models than simple Equivalent Circuit models), thermodynamics, and transport of lithium ions through Fickian diffusion and Ohmic conduction. The model is also called P2D (pseudo-two-dimensional model) since it gives an approximated uniform size to all particles inside the battery. One of the dimensions is the position x along the length of the battery between the external metal electrode contact points, and the second dimension r is the distance from the center of a solid particle [1] as can be seen in Fig. 2.2.

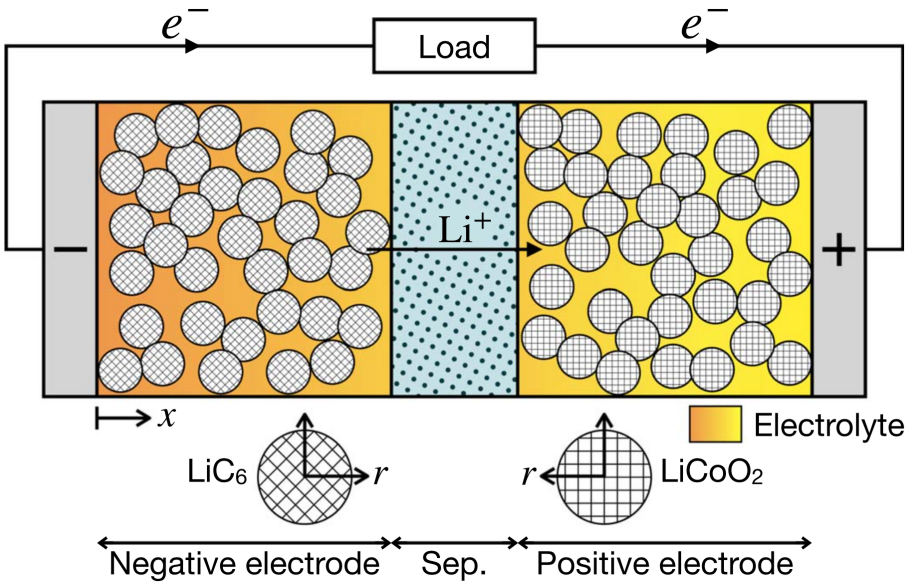

FIGURE 2.2: Schematic of the PET model for an LiC_6/LiCoO2 cell during discharge from [1]

The diagram in Fig. 2.2 shows the process of discharge when a load is connected between the extremes of the battery. The lithium ions can flow freely through the electrolyte and separator layer from the storing anode structure (graphite) to the metal oxide cathode (lithium cobalt oxide here), which creates a potential difference that makes the electrons flow through the load creating an electric current. The PET model is used for simulation as tightly coupled nonlinear partial differential-algebraic equations (PDAE). The **states of the model** are given by the electrolyte and volume-averaged solid concentrations c_e and c_s^{avg}, the ionic flux j, the electrolyte and solid potentials Φ_e and Φ_s, the applied current I, and the temperature T for non-isothermal simulations [1]. These states define the battery's evolution and are the ones that can be measured to quantify the uncertainty of the outputs.

2.2.2 The PETLION model and software

PETLION is an open-source software[4], a high-performance model that is easy to use and modify even for the general public. The software allows for multiple straightforward evaluations of battery operations. It can run on conventional operational modes such as constant current, voltage, and power. However, it also implements more novel options such as conditions based on temperature, lithium plating over-potential, and others. It can also be used for nonlinear model predictive control (NMPC) to find optimal charging paths under desired constraints.

The simulation equations are based on the PET model described and are implemented using the Finite Volume Method (FVM) in the x dimension. The software converts the PDAE system into a set of DAEs with time t as the independent variable. The differential equations on the model describe the evolution of the temperature and concentration variables. The algebraic equations apply conservation laws on the charge and mass. The DAE system's discretization consists of seven sections split into the x and r dimensions. In the x-direction there are N_p, N_s, and N_n points for the cathode, separator, and anode respectively. The discretizations in the r direction with Fickian diffusion are $N_{r,p}$ and $N_{r,n}$ for the cathode and anode solid particles. A complete description of the governing DAE system, FVM applications, parametrization, and discrete implementation of PET can be found in the documentation of the Matlab version LIONSIMBA [18], or for PETLION [1]. In this project, the number of discretization points used are $N_p = N_s = N_n = N = 10$ defining a PET model with 351 DAEs.

The PETLION software equations have been validated multiple times by comparison against other codes [1]. When compared to LIONSIMBA, a Matlab implementation of PET [18], all output state errors are below 10^{-5}. Usage of the Julia language allows PETLION to be easily accessible (as it has an open-source license), gives the model fast speed after compilation, and ensures multiple options for the numerical methods and ODE solvers. Julia has a growing ecosystem for tools on ODEs and physics-informed neural networks. PETLION was designed for fast serialized evaluations (highly efficient after the first compilation). It reduces the overhead by allowing the user to choose desired outputs from the model. The computational efficiency is verified when compared against similar battery-modeling software such as LIONSIMBA (on Matlab) and PyBaMM (on Python). On average for non-isothermal conditions, PETLION is 229×faster than LIONSIMBA, 68x faster for PyBaMM's "fast" mode, and 159×faster for PyBaMM's "safe" mode [1]. This efficiency can be directly related to the usage of Julia since MATLAB and Python are interpreted languages compared to the compiled nature of Julia.

[4]The code, installation instructions, and some usage examples are available in JuliaHub

This difference makes basic procedures such as 'for' loops faster, reducing the overhead usually incurred when reinterpreting the code on each iteration.

Finally, PETLION uses the most efficient solver for DAE systems in Julia, which was proven to be Sundials in [1]. The ensemble of characteristics and the effort devoted to optimizing the software by exploiting optimal Julia features and its open-source characteristic make PETLION an excellent option for battery simulation. As stated in [1]: "Currently, PETLION does not support forward or adjoint parametric sensitivity analyses" so the exploration of these features is an exciting step forward in this project.

2.2.2.1 Mixed continuous-discrete simulation for optimal charging

Past work done on optimal charging of batteries using PETLION was based on a mixed continuous-discrete model, which is also the approach used in this project. The continuous nature of the PETLION solver allows computing the states of the battery at any point in time for multiple possible charging operational modes. Conventional and straightforward battery charging modes are constant current (CC) and constant voltage (CV). The discrete part of the simulation accounts for the control boundaries that the solver can detect over the steady base. When charging, the interest of control is setting and following desired safety or optimal constraints such as a limit maximum voltage, a maximum temperature, or a minimum plating over-potential. Mixed continuous-discrete optimal control allows the integration of discrete boundaries in any charge path that leads to new limited operational methods.

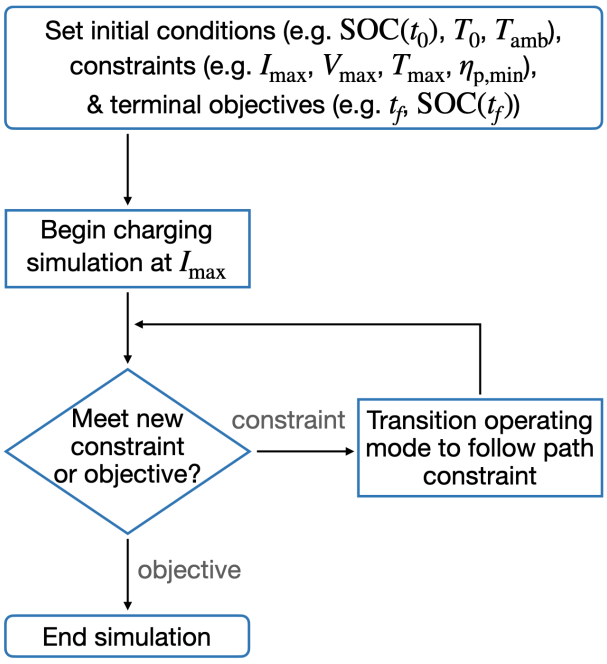

FIGURE 2.3: Flowchart for mixed continuous-discrete solution to charging protocols in PETLION [1]

When a specific boundary such as T_{max} is reached the simulation can continue by setting that new boundary as the base operational mode (in this case a CT charge at T_{max}). As an example, a CC-CT-CV charge is shown with boundaries $\{V_{max} = 4.1V,\ T_{max} = 40\,^{\circ}\text{C},\ SOC_{max} = 80\%\}$:

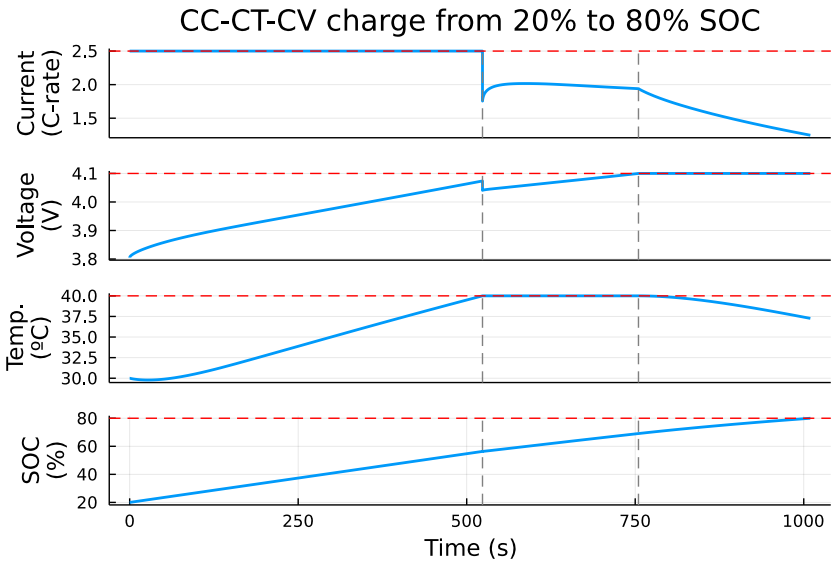

FIGURE 2.4: Charging default PETLION Lithium-ion Battery (with 1D temperature) following continuous-discrete modeling from SOC = 20% to 80% with boundaries on temperature and voltage.

PETLION includes new unconventional operational modes that can be of interest in continuous-discrete simulation when other boundaries are set. These include constant temperature (CT), Plating Over-Potential (CPo), constant electrolyte concentration (CCe), constant solid surface concentration (CCss), and constant mechanical stress (Cσ).

2.3 The Julia language

The Julia programming language was initially designed in 2010 with highly ambitious goals. It set out to be an open-source language, with a liberal license, with the speed of C with higher dynamism, familiar mathematical notation, and that was usable for general programming, statistics, string processing, linear algebra, and machine learning among others[5]. More than ten years after the first release, the language has had over one hundred contributors. Thousands of people worldwide are using Julia, and it is also taught in universities and used as base software in entire companies. Julia has proven to be fast and has a high performance in standard tests compared to other common languages such as Matlab and Python [20]. It is dynamically typed, has an easily reproducible environment, uses the multiple dispatch paradigm effectively, and has many general testing, debugging, and profiling tools.

[5]See Why we created Julia from the creators on the Julia blog - 14 February 2012

The nature of Julia and its focus on high speed makes it a perfect language for this research. The open-source nature allows direct usage of multiple optimized libraries. PETLION.jl code is currently available online as a Julia library for battery modeling. The versatility and speed of Julia allow modeling complex DAE systems efficiently and being able to run computationally expensive tasks in much shorter times. This efficiency is a big priority in optimal battery control. On the one hand, to make the development and programming much faster, given the significant difference of not waiting for weeks to obtain some long-time simulated results. On the other hand, speed and efficiency could make a big difference in adopting these technologies on online applications for Advanced Battery Management Systems (ABMS).

Some open-source libraries of Julia besides `PETLION.jl` that are fundamental to this project are `LinearAlgebra.jl`, `DifferentialEquations.jl`,`Dierckx.jl` (interpolation of discrete functions), `Statistics.jl`, `Distributions.jl` and `Plots.jl`. There is a specific library for local sensitivity analysis on Julia called `DiffEqSensitivity.jl`[6], but it has limitations regarding the choice of initial values of the sensitivity and their derivatives. One of the main functionalities of the PETLION software is its continuous-discrete operational modes with multiple schemes. The built-in sensitivity analysis of Julia does not suffice to keep the continuity of the analysis in the discrete changes of operational mode as it requires setting null sensitivities for every new system of equations.

The software developed in this project and the computational results are all run on a Julia 1.7.2 kernel. The version of the battery modeling PETLION.jl package used is 0.2.6, the current latest release (in May 2022).

[6]See complete features of the DifferentialEquations.jl library in [14]

Chapter 3

Mathematical background and Methodology

3.1 Uncertainty quantification

The main focus of this study is stochastic Uncertainty Quantification (UQ) which consists in quantitatively characterizing the uncertainty in real-world and computational applications. Most physics equations are deterministic and explain principles of reactions according to the expected behavior. It is known, though, that real-world problems and experiments are affected by a wide range of probabilistic factors which should be considered when assessing the performance of a simulation model. Especially in the study and analysis of batteries, which are expected to have a wide range of operating temperatures, capacities, and extended lifetimes, it is crucial to consider parameter variability and conditions.

UQ determines how likely an outcome change is if some of the model parameters are uncertain. To define the model, consider a system with a state vector $x \in \mathbb{R}^{n_x}$ which depends on a series of uncertain parameters $\theta \in \mathbb{R}^{n_\theta}$ (where n_x is the number of states, and n_θ is the number of uncertain parameters that affect the model. The main goal is to consider the effect of parametric uncertainties in θ on the probabilistic distribution of the states and variables of the model predictions. The process of UQ can be decomposed into three main steps:

1. Characterisation of the parameters and their uncertainty
2. Propagation of uncertainty in the model
3. Uncertainty management or evaluation and exploitation of the information obtained

Regarding the first characterization step, the model parameters' uncertainty can be represented as deviations from the standard/desired norm with some bounds. We define $\theta = \hat{\theta} + \delta\theta$ which represents a perturbation $\delta\theta$ about the nominal model parameters $\hat{\theta}$. Propagating uncertainties, we quantify how this input perturbation is transferred onto the output concerning the deterministic nominal value. For any state or variable of interest $\delta\psi = \psi - \hat{\psi}$. The aim is thus to quantify $\delta\psi$ to determine the probability distribution function (pdf) of any variables of interest such as ψ. Finally, the pdf characterization allows for defining upper and lower probability bounds and understanding the distribution of the results. When applied to battery modeling, this can help control charging states, detect real important boundaries, and solve safety issues.

3.1.1 UQ on Differential-Algebraic Equations

Differential-Algebraic Equations (DAEs) are systems that contain both differential (standard ODE systems) and algebraic equations. Algebraic equations usually represent some physical/ mathematical constraint that gives more information than standard differential dynamics. Since the PETLION model is built on a system of DAEs, the uncertainty quantification is applied to the complete system of equations, and the specific mathematical methodology must be known. A parametric DAE can be defined implicitly as $F(t, dx, x, \theta)$ which can be split:

$$\frac{dx}{dt} = f(t, x, \theta)$$
$$0 = g(t, x, \theta)$$

The part $g(t, x, \theta)$ has all the algebraic equations of the system. For the system's solution to be well defined, the dimension of the complete DAEs must be the same as the number of states: $F(t, dx, x, \theta) \in \mathbb{R}^{n_x}$ having an equal number of equations and unknowns. The goal of uncertainty quantification in this system is to compute the variation of any of the states x for the probabilistic parameters of interest $\theta = \hat{\theta} + \delta\theta$. Following the previous notation, the output of interest ψ can be any or all states in x.

Regarding the actual resolution of the DAE, it is usually defined in 2 steps: first, the initial conditions must be precisely computed, and secondly, the trajectory of the states can be found. General computational methods for solving DAEs consist of converting the systems into ODEs to use more common ODE solvers, but this can be arduous and even impossible for implicit algebraic constraints. The advantage of Julia is that there are multiple numerical solvers in Julia[1] for DAEs both in mass-matrix form or implicitly defined. Given the complexity of the system of equations required to simulate Li-ion battery operations in PETLION, a single run

[1] Julia DAEs introduction and solvers are described in diffeq.sciml.ai

of the nonlinear DAE solver can already be an intense computational task. A direct approach to quantifying uncertainties on the DAE would require running the complete model with the integrated solver multiple times to test every possible combination of uncertain parameters. Then, the ensemble of outputs could be analyzed, and one could attempt to fit the values into some probabilistic distribution curve to understand the dispersion or characterize worst-case boundaries. The main goal of Sensitivity Analysis for UQ is to determine the probabilistic output distribution of the states without having to re-run the nonlinear model a high number of times.

3.2 Sensitivity Analysis

Sensitivity Analysis (SA), also known as parametric sensitivity, characterizes how the variation of an output state in a model can be related to the different sources of uncertainty found in its input parameters. SA can be helpful for multiple purposes, such as increasing the understanding of a model's behavior from the relationship input-output, assessing the robustness of a system in the presence of uncertain values, or finding sets of parameters that generate limit outputs to restrict their deviations beforehand.

Sensitivity Analysis (SA) can determine how target-dependent variables are affected by different sources of uncertainty on independent variables. For a given set of uncertain parameters and their bounded ranges, the SA goal is to characterize their effect on the overall uncertainty of the model. Previous mathematical research by M. Caracotsios and W.E. Stewart [4] describes an efficient method to apply Sensitivity Analysis to Initial Value Problems for Algebraic Equations. The relatively low computational cost and simple implementation make SA a popular method for parametric uncertainty quantification with many potential applications in optimization, model simplification, and experimental design.

The main objective is to evaluate the change over time of output ψ (any target state or function of the battery model) within a bounded set of time-invariant uncertain parameters θ. To characterize the output variation, the deviation on the vector of parameters can be defined as $\delta\theta$ such that $\theta = \hat{\theta} + \delta\theta$ with respect to the nominal value. Since the model state depends on its parameter $\psi = \psi(\theta)$. In a neighborhood of the optimal known deterministic solution $\hat{\psi}$, a Taylor series expansion can be used to quantify the deviation in these outputs according to [11]:

$$\delta\psi = L\delta\theta + \frac{1}{2}\delta\theta^T M\delta\theta + ... \tag{3.1}$$

Where $L(t) = \nabla_{\hat{\theta}}\, \psi(t) \in \mathbb{R}^{n_\theta}$ is the sensitivity vector and $\mathbf{M(t)} = \nabla^2_{\hat{\theta}}\, \psi(t) \in \mathbb{R}^{n_\theta \times n_\theta}$ is the sensitivity matrix which quantify the change in the desired output with respect to $\delta\theta$. This formulation defines a linearized solution to the UQ problem where the expansion only has to be accurate in a region close to the control solution. Even for highly nonlinear processes, the use of a small number of terms (1st or 2nd order expansions) can give satisfactory results [11].

3.2.1 Output probability distribution with 1st order approximation

As explained, the final goal of this research is to characterize the change in the output when the model parameters are uncertain. Thus, it is essential to understand how $\delta\psi$ affects the final value ψ, its deviation from the optimal result, and its probabilistic distribution. When a first-order expansion is used on the sensitivity analysis to define the deviation as $\boxed{\delta\psi = L\delta\theta}$, it is possible to compute an analytical expression of the worst-case outputs [12]. More importantly, when the uncertain parameters follow a multivariate normal distribution, it is possible to define the pdf of the output states analytically. Since most algorithms that estimate parameters from experimental data produce a multivariate normal distribution, this analytical formula is significant. For the specific case of Li-ion battery modeling, the past study on the identifiability of the parameters [2] gives independent normal distribution coefficients for the solid-phase diffusion and reaction rate constants. Therefore, the uncertain parameters θ are characterized here by a multivariate normal distribution:

$$f_{\mathbf{p.d.}}(\theta_1, \ldots, \theta_{n_\theta}) = \frac{1}{\sqrt{(2\pi)^{n_\theta}|\boldsymbol{V_\theta}|}} \cdot exp\left(-\frac{1}{2}[(\theta - \hat{\theta})^{\mathrm{T}} \boldsymbol{V_\theta}^{-1} (\theta - \hat{\theta})]\right)$$

Where θ is the vector of parameters for a specified uncertain event, $\hat{\theta}$ is the nominal set of parameters, and $\boldsymbol{V_\theta}$ is the variance/covariance matrix of this distribution. In this study, the parameters are considered independent, so the covariance matrix is a diagonal matrix defined by the variance of each parameter's pdf. For a first-order Taylor expansion on $\delta\psi$ given by the sensitivity vector $L(t)$, the estimated probability distribution of the output state is given by:

$$f_{p.d.}(\psi) = \frac{1}{\sqrt{2\pi V_\psi}} \cdot exp\left(-(\psi - \hat{\psi})^2/(2V_\psi)\right) \tag{3.2}$$

Which is a standard normal distribution for a single state where the variance is given by [12]:

$$V_\psi = L\boldsymbol{V_\theta}L^T \quad \longrightarrow \quad \sigma_\psi(t) = \sqrt{V_\psi(t)} = \sqrt{L(t)\boldsymbol{V_\theta}L(t)^T} \tag{3.3}$$

This distribution of ψ is a function of time according to the values of $L(t)$ that determine $V_\psi(t)$ and the nominal output $\psi(t)$. The proof to obtain (3.2) can be done using the first-order series

expansion that defines $\delta\psi = L\delta\theta$ and knowing that the dispersion in θ is characterized as a normal distribution which makes ψ normally distributed. This mathematical model is verified on the results section by comparing the theoretical pdf given by (3.2) against a Monte Carlo method using the whole nonlinear model for the multinormal distributed input parameters.

The first-order approximation of Sensitivity Analysis will be used in this project to quantify the uncertainty in any output state as it directly gives the output's mathematical probability distribution. To be able to define the pdf in (3.2) it is required to compute V_ψ which is found from the known variance matrix of the uncertain parameters V_θ and the sensitivity vector $L(t)$. Thus, the goal of this project is mainly to compute accurately $L(t) = \nabla_{\hat{\theta}}\, \psi(t) \in \mathbb{R}^{n_\theta}$. Once this vector is computed, the output values of interest can be accurately bounded without re-running the whole nonlinear model for every combination of uncertain coefficients. The sensitivity vector $L(t)$ can be computed by solving the system's differential-algebraic equations with the additional set of **sensitivity equations** or by using **finite differences**.

3.2.2 Sensitivity Equations for differential systems

There are two common approaches to computing sensitivity analysis numerically: Forward Sensitivity Analysis and Adjoint Sensitivity Analysis. The latter is a more complex alternative as it requires defining additional functions and computing the DAE solution separately to know the states at any point. The former is used to compute local sensitivities simultaneously with the actual ODE solver and is the one used to define the Sensitivity Equations.

An efficient method for sensitivity analysis of nonlinear IVP is based on implementing Sensitivity Equations (SEs). These equations are linear and can be solved with the local Jacobian of the states' system. Consider an ODE system of the form $\dot{x}(t) = f(t, x(t), \theta)$ with known initial values. The sensitivity equation to determine **the elements of** $L(t)$ with respect to θ is:

$$\dot{\mathbf{L}} = \mathbf{J}_x\mathbf{L} + \mathbf{J}_\theta \qquad \text{where} \qquad \mathbf{J}_x = \frac{df}{dx} \in \mathbb{R}^{n_x \times n_x}, \quad \mathbf{J}_\theta = \frac{df}{d\theta} \in \mathbb{R}^{n_x \times n_\theta} \tag{3.4}$$

In this notation $\mathbf{L}$ is a matrix that contains in every row the vector $L = \nabla_{\hat{\theta}}\, \psi(t)$ for each state ψ of the whole vector x that define the ODE states. The linearity of (3.4) and the possible integration into the original ODE model (since they share the Jacobian) make sensitivity equations a really efficient method to perform SA. The time-varying element $L(t)$ can be computed by adding this sensitivity equation to the solver system. To do so the sensitivity equation is split for each θ_j into column-wise vectors of $\mathbf{L}$ that represent new system sensitivity states. The number of states is extended by $n_f \cdot n_\theta$ and the Jacobian is repeated for each uncertain parameter in vector θ.

3.2.2.1 Derivation of the sensitivity equations

Sensitivity equation (3.4) can be obtained following the notation developed by M. Caracotsios and W. E. Stewart [4]. Define a dynamic system where x is the vector of state variables and θ is the vector of time-independent uncertain parameters given by the set of ODEs or DAEs $F(t, x, \dot{x}, \theta) = 0$ with initial values $x(t_0) = x_0(\theta)$ and $\dot{x}(t_0) = \dot{x}_0(\theta)$. The fully implicit form $F = 0$ is used to describe more generic DAE systems. Deriving the system of differential state equations F with respect to one time-invariant uncertain parameters θ_j:

$$\frac{\partial F(t,x,\dot{x},\theta)}{\partial \theta_j} = \frac{\partial F(t,x,\dot{x},\theta)}{\partial x(t)} \cdot \frac{\partial x(t)}{\partial \theta_j} + \frac{\partial F(t,x,\dot{x},\theta)}{\partial \dot{x}(t)} \cdot \frac{\partial \dot{x}(t)}{\partial \theta_j} + \frac{\partial F(t,x,\dot{x},\theta)}{\partial \theta_j} = 0 \tag{3.5}$$

Which is directly equivalent to the sensitivity equation in (3.4) considering the definition of the column vector $L_{\theta_j}(t) = \frac{\partial x}{\partial \theta_j}|_t$ and defining the Jacobians of the system of equations F as $\mathbf{J}_x = \frac{dF}{dx} \in \mathbb{R}^{n_x \times n_x}$, $\mathbf{J}_{\dot{x}} = \frac{dF}{d\dot{x}} \in \mathbb{R}^{n_x \times n_x}$ and $J_{\theta_j} = \frac{dF}{d\theta_j} \in \mathbb{R}^{n_x \times 1}$.

The final DAE sensitivity equation is:

$$\mathbf{J}_x \cdot L_{\theta_j} + \mathbf{J}_{\dot{x}} \cdot \dot{L}_{\theta_j} + J_{\theta_j} = 0 \tag{3.6}$$

In matrix forms, the jacobian matrices and sensitivity vector are:

$$\mathbf{J}_u = \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \dots & \frac{\partial f_1}{\partial u_k} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \dots & \frac{\partial f_2}{\partial u_k} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_k}{\partial u_1} & \frac{\partial f_k}{\partial u_2} & \dots & \frac{\partial f_k}{\partial u_k} \end{pmatrix}, \quad J_{\theta_j} = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_j} \\ \frac{\partial f_2}{\partial \theta_j} \\ \vdots \\ \frac{\partial f_k}{\partial \theta_j} \end{pmatrix}, \quad L_{\theta_j} = \begin{pmatrix} \frac{\partial u_1}{\partial \theta_j} \\ \frac{\partial u_2}{\partial \theta_j} \\ \vdots \\ \frac{\partial u_k}{\partial \theta_j} \end{pmatrix}$$

For the specific case where the system is an ODE without any algebraic equations, the matrix $\mathbf{J}_{\dot{x}}$ is equivalent to the identity matrix of dimension n_x, which is coherent with one of the column elements of the original Eq. (3.4) (considering the slight difference in the Jacobian definitions caused by the explicit equation form of the ODE function f).

For the PETLION model, $L(t)$ can be computed by integrating equation (3.6) into the DAE model. The addition will define a new set of states (one for each original state) that consider the sensitivity against a single parameter. There will be $n_\psi \cdot n_\theta$ sensitivity total equations added to the system. Since Initial Values define the problem, it is also necessary on sensitivity equations to correctly define the initial conditions for $L(t = 0)$ and $\dot{L}(t)$. As stated in [1]: "Accurate, consistent initialization for DAEs is an important part of maintaining the numerical stability in a simulation code". This initialization is crucial to avoid deviations in further points and ensure

convergence of the solver algorithms that might fail otherwise.

3.2.2.2 Initial conditions in sensitivity equations

The usage of sensitivity equations allows seeing this problem as an extended system of DAEs, which means that the initial conditions can be determined as one would for the original system. This is based on solving the system $F(t_0, x, \dot{x}, \theta) = 0$. The initial sensitivity values are 0 for all differential state variables as the sensitivity effect has to begin null. The initial sensitivity of the algebraic states is found by solving $g(t_0, x, \theta) = 0$ with any numerical method. Finally the initial sensitivity derivatives are computed solving the system (3.6) in matrix form using the previously found values of $L(t_0)$ and isolating $\dot{L}(t_0)$. Also, the initial sensitivities for some of the uncertain parameters are known to be null a priory if they only affect differential terms. For example, initial sensitivities with respect to diffusion constants such as D_{sn} and D_{sp} are zero but with respect to k_n and k_p they are non-zero. The expanded DAE system now has initial conditions based on the union of the original states x_0 and $\dot{x}_0$ and the initial sensitivities for as many parameters as desired. With this expansion of the DAE system, it is possible to determine the uncertainty for each desired output. For this research, it is necessary to have consistent initial conditions for every continuous charging mode that has different DAEs. This implies that the sensitivity values on discrete points also have to be fit accordingly. The specific mathematical notation to do so is thoroughly described in 'Parametric sensitivity functions for hybrid discrete/continuous systems' [5].

3.2.2.3 Numerical implementation of sensitivity equations

To use built-in Julia solvers the DAE must be defined as `F(res, du, u, p, t)` which modifies the states' residuals in-place. This function is built by extracting the states `Y, YP` from PETLION and extending them using sensitivity equations. The issue with this methodology is that knowing the states outside the model requires some modifications on the software that make the PETLION solver much slower. Once the function is built it can be fitted to a DAE type for the solver using `fun = DAEFunction(F)`. The problem is defined using `prob = DAEProblem(fun, du0, u0, tspan, p, differential\_vars)` where $du0$ and $u0$ are the extended initial conditions, p contains the uncertain parameters, $tspan = (t_0, t_{end})$ and differential_vars is a *Boolean* vector to indicate which states are not algebraic. Finally, the Sundials IDA built-in solver can be called using `solve(prob, IDA())`[2].

[2]DAEFunction() and DAEProblem() are functions of the DifferentialEquations.jl library.

3.2.3 Finite Differences to compute the sensitivity vector

The sensitivity vector and matrix can also be approximated using numerical methods such as finite differences. Knowing that $L(t) = \nabla_{\hat{\theta}}\ \psi(t)$ accounts for the output variability, this first derivative can be estimated with multiple accuracy approaches to finite differences. Since this research aims to provide precise results while ensuring fast computation, it is crucial to use a numerical method that considers the trade-off between accuracy and computational cost. With this goal, the finite differences approach chosen is **second-order accurate central differences**[3],

$$L(t) = \frac{\frac{1}{2}\psi(\theta + \Delta\theta) - \frac{1}{2}\psi(\theta - \Delta\theta)}{\Delta\theta}, \tag{3.7}$$

which is more precise and has lower numerical noise (oscillations) than usual first-order accuracy forward finite differences.

This method is chosen for its accuracy and computational speed since every complete simulation of the model only has to be run $2n_\theta$ times (two deviations for every parameter). It is verified on Sec. 4.2.3 that using higher orders of differentiation does not give significantly higher accuracy but is more computationally expensive. The selection of the time step $\Delta\theta$ is vital in this method since it could alter the results. A tiny step has significant issues with accuracy because it is closer to the machine precision *eps*, which causes round-off errors. A big step does not give precise results since it has to be small enough to fulfill the local linearity assumption of the Taylor expansion in the finite differences approximation.

3.2.3.1 Numerical implementation of Central Finite Differences

An algorithm is defined to compare the accuracy of several finite differences defining $L(t) = \frac{1}{\Delta\theta}\sum_{i=0}^{n} \alpha_i \psi(\theta + \beta_i \Delta\theta)$ where n is the order of FD, α are the coefficients and β are the indices of each term. For the second-order accurate CFD case in Eq. (3.7): `coef. = [1/2, -1/2]` and `indices = [1, -1]`. The methodology used employs Julia `@threads` to compute the $n \cdot n_\theta$ summation terms in parallel. There are exactly four threads available in this simulations given the computer used. To ensure that there are no issues with allocations, four copies are created of p (the structure that contains the parameters and specifications of the model in PETLION).

3.2.4 Logarithmic sensitivity equations for small parameters

In this project, the set of uncertain parameters has various orders of magnitude ranging from the ambient temperature $T_{amb} \sim 10^2$ to the diffusion coefficients $D_i \sim 10^{-14}$. Implementing directly finite differences in these small parameters can lead to some inaccuracies as the deviation $\Delta\theta$

[3] Multiple finite differences options and their coefficients are described in Wikipedia

gets close to the machine precision **eps** $\sim 10^{-16}$. In these cases, a common alternative to the standard sensitivity vector definition is a logarithmic derivative:

$$L(t) = \nabla_{\hat{\theta}}\, \psi(t) = \nabla_{\log_{10}\hat{\theta}}\, \psi(t) \cdot \nabla_{\hat{\theta}}\, \log_{10}\theta = \nabla_{\log_{10}\hat{\theta}}\, \psi(t) \cdot \frac{1}{\ln 10 \cdot \hat{\theta}} = \frac{L_2(t)}{\ln 10 \cdot \hat{\theta}}$$

Here, the vector $L_2(t) = \nabla_{\log_{10}\hat{\theta}}\, \psi(t)$ defines the derivative of the states against the logarithm of the parameters which are values much more tractable given their order of magnitude. This mathematical notation is used when computing sensitivity components for uncertain parameters of a low order of magnitude, giving better precision and less numerical noise.

3.3 Methodology for implementing Sensitivity Analysis

The methodology used to apply the mathematical background to battery models starts from a simple *deterministic* simulation. This simulation serves as the basis for all the desired computations, such as deviation ranges, back-off boundaries, and new safety constraints. Since the project's goal is to quantify the output uncertainty using sensitivity analysis, the value of the sensitivity vector $L(t)$ is found first to compute the actual deviation of any state later. In summary, the methodology used in this project for complete uncertainty quantification of mixed continuous-discrete simulations is:

1. Run any desired operational mode $CC, CV, CPo, CT, ...$ with nominal parameters
2. Choose the uncertain parameters that affect this model and their normal distributions
3. Choose a success range α for the percentage of accuracy desired in the output dispersion
4. For every or all **states of interest** of the simulated model:
   (a) Compute the state's parametric sensitivities $L(t)$ using Central Finite Differences
   (b) Compute the standard deviation using the definition of σ_ψ in (3.3)
   (c) Use the standard deviations equivalent to α to determine new state limits
5. Check new boundaries (time constraints, maximum values, or safety limits) for the deviated model. Determine new back-off from results of interest against past deterministic results.

The procedure is carried out in the final results chapter to quantify the effect of parametric sensitivities on lithium-ion battery operational modes. For efficient and simplified computation, all uncertain model states are found simultaneously as **states of interest**. This methodology gives two *exact* copies of the original model that represent new upper and lower limit states. The exact implementation of these steps using the Julia programming language is shown in Ap. (B).

Chapter 4

Preliminary Results

Preliminary results include four simple test cases used to verify the functionality of the proposed method and better understand the results with smaller examples. The first case shown in Sec. 4.1 implements the sensitivity equations on a simple chemical reaction. The second case uses different methods to compute the sensitivity vector for a basic Li-ion battery charge. The third case, in Sec. 4.3, quantifies the probabilistic state distribution in PETLION based on the previously found sensitivity vector. The last case determines the parametric sensitivity and output deviations for multiple standard uncertain parameters in batteries.

4.1 Chemical reaction example

Since the equations that define the PETLION model are complex and the state variables are highly related between them. A simpler chemical example was used first to have a faster and easier approach to the sensitivity equations' implementation in Julia. The reaction studied is:

$$A \xrightarrow{\text{k1}} B \underset{k_3}{\overset{k_2}{\rightleftharpoons}} C$$

The goal is to simulate the concentration of species A, B and C in a constant volume batch chemical reactor with isomerization kinetics. The initial concentrations of the tank are $C_A(0) = 1\,\text{M}$ and $C_B(0) = C_C(0) = 0\,\text{M}$. The rate constants are $k_1 = 1000s^{-1}$, $k_2 = k_3 = 1s^{-1}$ which implies that A will react really fast and B and C reach an equilibrium point later on.

The set of uncertain parameters chosen for this sensitivity analysis problem is given by the vector $\theta = [C_{A,0}, k_1, k_2, k_3]$, including all the time-invariant parameters that affect the reaction.

4.1.1 ODE approach

The basic equations that describe the 3 states of the chemical reaction are given usually by differential equations that express the change in every concentration using the rate constants that affect it: $\{\frac{dC_A}{dt} = -k_1 C_A,\ \frac{dC_B}{dt} = k_1 C_A - k_2 C_B + k_3 C_C,\ \frac{dC_C}{dt} = k_2 C_B - k_3 C_C\}$

4.1.2 DAE approach

Using algebraic equations allows building physical constraints into the system while keeping the number of equations that the ODE had. In this case, we want to explicitly conserve the mass in the system: $C_A(t) + C_B(t) + C_C(t) = C_{A,0}$. We can replace the last equation of the system with the conservation law. The equation can now be written in a fully implicit form as:

$$F(t, C(t), \dot{C}(t)) = \begin{bmatrix} -k_1 C_A - \frac{dC_A}{dt} \\ k_1 C_A - k_2 C_B + k_3 C_C - \frac{dC_B}{dt} \\ C_A(t) + C_B(t) + C_C(t) - C_{A,0} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.1}$$

Keeping in mind that the PETLION model is based on a system of DAEs, the system of equations given by (4.1) is the one used to test the future implementation of the uncertainty quantification mathematical systems.

4.1.3 Evolution of the states in the reaction

The resolution of the ODE or DAE models defined above can be found with Julia using any of the available numerical integrating tools. Using the `DAEProblem()` function with the Sundials' `IDA()` solver gives the following results:

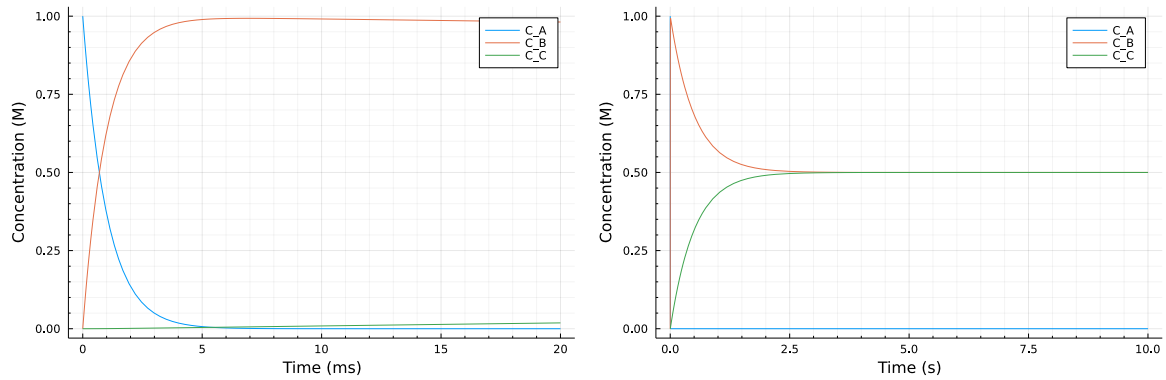

FIGURE 4.1: Evolution of the concentration of each substance in the basic chemical reaction example. A is the original reactant, and B and C are the final products that maintain equilibrium. Left shows the 'fast' start of the reaction (20 ms) and right shows the long-term states (10 s).

4.1.4 Sensitivity analysis

Sensitivity equations in Eq. (3.6) are used to find the sensitivity vector of this simple DAE system. The IDAS Julia solver then solves the complete system of equations as a regular one with the extended states. It is important to choose accurate initial conditions for the model's success. The results of the sensitivity analysis on the 3 states of the system $x(t) = [C_A(t), C_B(t), C_C(t)]$ for the 4 parameters $\theta = [C_{A,0}, k_1, k_2, k_3]$ is:

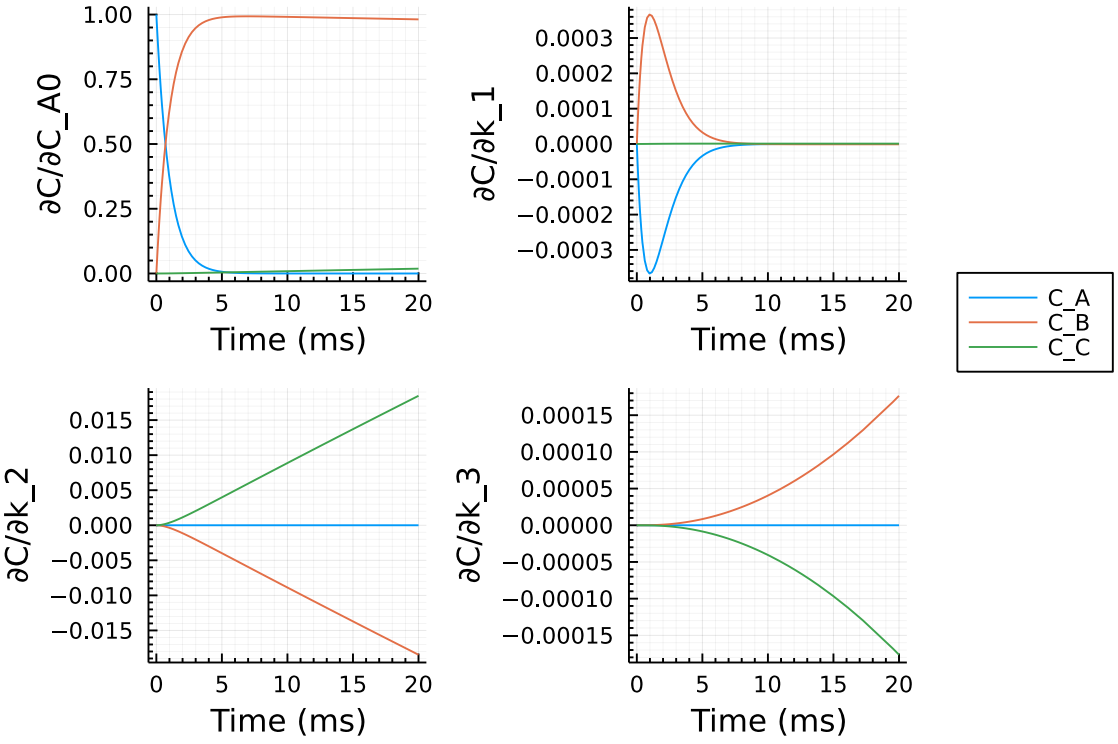

FIGURE 4.2: Initial evolution ($20ms$) of the parametric sensitivity of each substance in the chemical reaction example against the desired parameters in $\theta = [C_{A,0}, k_1, k_2, k_3]$. The parametric sensitivities for this basic chemical reaction can be computed in ~ 66 ms.

The results seen in Fig. 4.2 can be logically interpreted as parametric sensitivities considering the behavior of the original equation. For example, the initial concentration of the reactant A directly affects the concentration C_A at $t = 0$ as it represents its value while C_B starts with 0 sensitivity and proliferates for the fast reaction until it is also ~ 1.0 (the concentration of B will depend entirely on $C_{A,0}$ in a short time). Similarly, the reaction rate constant k_1 highly affects the concentrations of A and B in the first ms of the simulation only and is zero otherwise since its value does not affect future concentrations. Of the three constant rate values, k_2 has the highest sensitivity value, which shows that it is more relevant to reaching the equilibrium, as expected. With this better understanding of sensitivity equations and analysis, it is now easier to carry out the SA of PETLION.

4.2 Sensitivity Analysis of PETLION

Multiple approaches have been tested to design the function that will conduct sensitivity analysis over the battery charging model. A standard charging process is used as the base of sensitivity analysis preliminary cases to test quickly and efficiently. For the remaining of this section, all software implemented and results refer to the default Li-ion battery on PETLION {cathode: LCO, anode: LiC6, Fickian solid diffusion, no aging, voltage bounds $[2.5, 4.3\,\mathrm{V}]$, 1D temperature modeling}. The battery operation used for initial testing of results is **fully charging battery from 0% SOC at constant current I = 1C (C-rate)**, which means that the battery is completely charged in 1 hour[1]. The evolution of some battery states during this charge are:

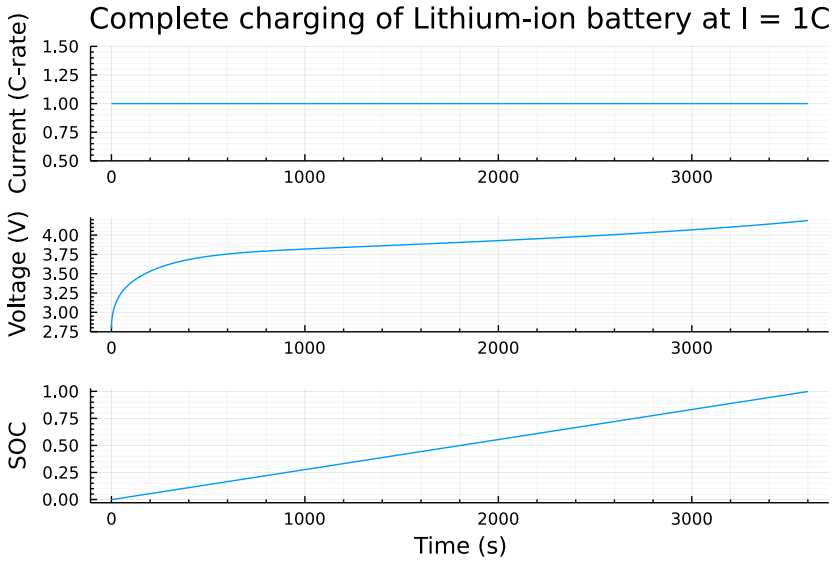

FIGURE 4.3: Charging default PETLION Li-ion Battery with 1D temperature modeling from state of charge $SOC_0 = 0\%$ at constant current $I = 1C$ until $SOC_{end} = 100\%$

The sections that follow, showing initial attempts and operations to implement sensitivity analysis, all refer to the computation of the parametric sensitivities of **this charging process**. The set of uncertain parameters is defined only by the ambient temperature $\theta = \{T_{amb}\}$ for the study of the preliminary results. Where $T_{amb} \sim N(298.15K, 1K) \sim N(25\,^{\circ}\mathrm{C}, 1\,^{\circ}\mathrm{C})$.

4.2.1 SA using Sundials

The first attempt to implement sensitivity analysis on the complete PETLION model was based on extending the usage of the Sundials software (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) [8]. This implementation could be straightforward since the current solver that PETLION uses for the DAE system is Sundials' IDA [9], and according to the documentation,

[1]The C-rate is a measure relative to the capacity of the battery such that an I = 2C charge is equivalent to the current required to charge the battery in 30 minutes and I = 0.5C charges the battery in 2 hours.

there is an extension called IDAS that allows automatic computation of sensitivities on every state for multiple parameters [16]. IDAS finds the sensitivity values by integrating the sensitivity equation for $L(t)$ into the DAE system based on (3.6). Changing the solver of PETLION from IDA to IDAS requires specifying the initial conditions for L_0 and $\dot{L}_0$.

This solution was not possible for implementation on Julia since there are some discrepancies between the functionalities of the original Sundials package (based on C language) and the equivalent package Sundials.jl. The Julia version has been adapted from the original Sundials C API and provides almost complete library coverage, but it has not been possible to wrap the sensitivity part. This option remains a potential alternative for the future whenever the Julia version of the software is completed to integrate sensitivity analysis into DAEs solver methods.

4.2.2 SA using sensitivity equations

Since the IDAS software was not compatible with the Julia language, the next best option is to implement the sensitivity equations manually following the mathematical notation given in (3.6). With this method, the number of equations is increased in the DAE so that the standard solver IDA of Sundials solves the expanded system of equations. The issue with PETLION is that the solver is integrated into the model, so the states and equations are not directly accessible. To be able to do this extension of the DAE, Marc Berliner helped in developing a simplified version of the PETLION code that had a main function compatible with IDA solver of the form `f_petlion!(res, t, Y, YP, p; kw...)` which is equivalent to a fully-implicit DAE form where `res` is the in-place evaluation of the function $F(t, x, \dot{x}, \theta)$ used in (3.6). Here the states are built into the variable Y, the derivatives into YP, the set of parameters stored in p as with the standard PETLION model, and $kw...$ stores extra model information for the run. Using the same procedure implemented for the simple chemical equation example, shown in the code A.2, it was possible to implement the sensitivity equations into the PETLION code.

Initial conditions

The initial conditions were defined following the mathematical notation from Sec. 3.2.2.2 so that the system of DAEs was fulfilled at the start point. To do so, Y_0 and YP_0 are initially guessed with the PETLION function `Y0, YP0 = PETLION.guess_init(p; SOC = 0)`. The state I is modified in $Y0$ to show the operating mode chosen, which is $I = 1C$. The algebraic initial conditions are set by solving the initial system $g(y, t) = 0$ and using also $YP0 = 0$ as the algebraic states have null sensitivity derivatives at the start. Finally, the initial values for the differential terms are found solving $f(y, t) - y' = 0$ at the start time.

A similar procedure is applied to the initial conditions of the sensitivity terms, the extended part of the DAE system. Initial sensitivity values are zero for all differential terms and computed from the system of equations for the algebraic states. The initial derivative terms are null for the algebraic states and can be computed from the system for the differential terms. The system solved is the final sensitivity equation Eq. (3.6).

Jacobian matrices

To be able to use the sensitivity equations it is necessary to define the jacobian matrices $\mathbf{J}_x = \frac{dF}{dx} \in \mathbb{R}^{n_x \times n_x}$, $\mathbf{J}_{\dot{x}} = \frac{dF}{d\dot{x}} \in \mathbb{R}^{n_x \times n_x}$ and $\mathbf{J}_\theta = \frac{dF}{d\theta} \in \mathbb{R}^{n_x \times n_\theta}$. These matrices are computed numerically in Julia. There are multiple functions designed for this purpose and the one used in this case was `ForwardDiff.jacobian()` from ForwardDiff.jl which returns $J(x)$ evaluated at a single point x. This is an accurate method to compute the Jacobian matrices but it is also computationally expensive as it is computing the complete matrix when most of the cases will just require a reduced amount of columns (most values on the states are 0). Also the Jacobian matrices are required in the sensitivity equation to compute some matrix-vector products so many of the terms will not be used at the end. These are possible aspects of this implementation that can bring computational inefficiency and could be further upgraded.

Preliminary results

This approach with the simplified PETLION code and sensitivity equations worked successfully in measuring the sensitivity of the output. The code was initially tested with uncertainties only on the ambient temperature modeled as a normal distribution with 1^{o}C of standard deviation and mean value $T_{amb} = 25\,^{\circ}\text{C} = 298.15\,\text{K}$. For this input uncertainty, the resulting sensitivity on some of the outputs is given in Fig. 4.4.

As expected, the current sensitivity shown in Fig. 4.4 is 0.0 since the model is pre-setting this value as an input and thus is not altered by the uncertainty. The voltage sensitivity is inversely proportional and grows with time, whereas the temperature sensitivity is directly proportional and grows fast at the beginning of the charging process. Also, the battery's temperature is highly sensitive to the ambient temperature, which is reasonable because these parameters are directly linked. The temperature in Fig. 4.4 is measured as a weighted average of all the discretized grid points of the battery model.

The solution of the extended DAE system after initialization takes **5651s ≈ 1h 34mins**[2] for the standard system with ten discretization points in all directions and setting the solver

[2] All tests are performed on a 2016 MacBook Pro 2.9 GHz 2-Core Intel i5 computer with 8 GB of RAM. Also, all simulations use Julia 1.7.2 kernel.

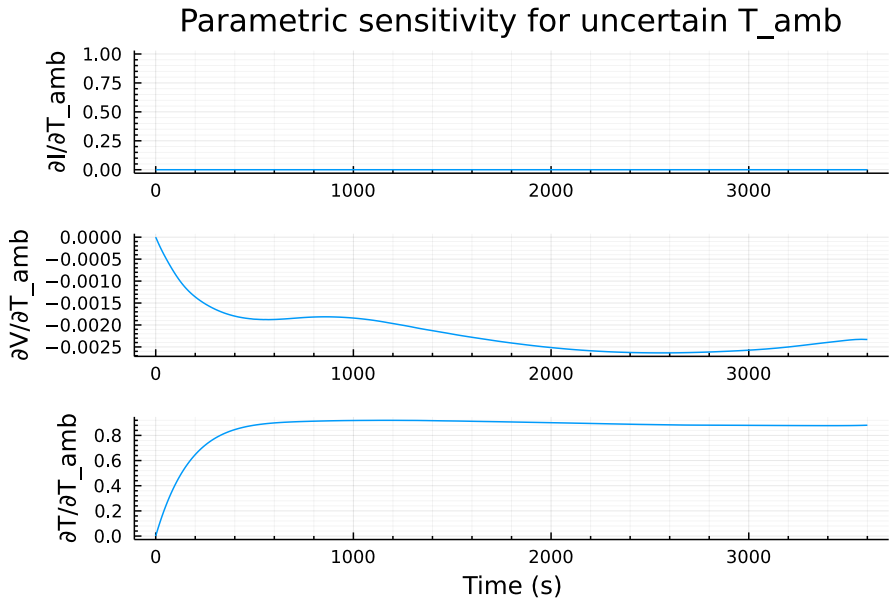

FIGURE 4.4: Parametric sensitivities for uncertain ambient temperature. Process charging default PETLION Lithium-ion Battery from SOC = 0 at constant current I = 1C. The sensitivity is computed for the states: current (I), voltage (V) and average temperature (T).

with 10^{-4} relative tolerance and 10^{-6} absolute tolerance. This result represents a substantial computational cost, especially when compared to the time required to solve the DAE system without the sensitivity extension, which is **92 s ≈ 1m 32s**. The time required to solve the extended program is >60x the initial time extending by far the expected 2x since the sensitivities are only computed for one extra parameter doubling the number of equations solved.

4.2.2.1 Issues with this method

After successfully finding sensitivities for the uncertain ambient temperature, some other parameters of interest were tested. The method was unsuccessful when applying uncertainties to the diffusion and equilibrium parameters. The Jacobian estimator used with Automatic Differentiation uses dual numbers to compute slopes. When other parameters are uncertain, they are defined as Dual numbers for the computation. This classification creates a compatibility issue since PETLION has built-in specifications that limit the type of its parameters. As a result, it is not possible to numerically compute the Jacobian matrices for D_{sn}, D_{sp}, k_n, k_p.

Independently on the ability to compute parametric sensitivities for other uncertain inputs, this method has a substantial computational cost compared to expectations. The method was initially designed here without focusing on speed and optimization, whereas an already built-in function to compute sensitivities on any language is based on multiple *tricks* that make them efficient. Every single computation of the Jacobian is an expensive procedure, and the sensitivity equations require computing three Jacobian matrices at every single point of the

DAE integration.

Improvement options

As suggested, there are multiple numerical approaches to optimize the computation of sensitivity equations, considering that the Jacobians are the biggest bottleneck:

- **Speeding up the Jacobian evaluation**: given the wide usage of Julia for mathematical modeling, linear algebra, and vastly complex systems, there are multiple optimization alternatives to compute these derivative matrices. In this original hard-coded approach, every Jacobian matrix is fully computed with ForwardDiff.jl at every point in time using. Alternative systems to optimize the computation and storage of these matrices are:
  - **Sparsity form of the Jacobian**: sparsity tools allow building and storing only the relevant values of the matrices. Many iterations of the Jacobian are used to determine which values are not important from the Dense matrix form using SparseDiffTools.jl. The sparse jacobian configuration can then be fed to the ForwardDiff.jl package to compute the matrices faster.
  - **JuMP graph coloring optimization**: the Jacobian can be evaluated using a sparse forward-mode automatic differentiation based on a graph coloring algorithm [15] that reduces the number of function evaluations. PETLION uses this method.
- **Matrix-vector direct computations**: a potential alternative to computing expensive Jacobian matrices on systems of ~ 350 equations is based on computing the desired product $J_x \cdot L(t)$ directly. This option requires fewer computations since multiple values on the vector and matrices can be null and lower allocations result in much lower memory storage for every iteration.
- ***Symbolics* approach**: the library `Symbolics.jl` allows defining a *complex problem* with multiple state variables and parameters as a symbolic result. The exact value of every output has a precise equation, depending on the original states, that uses basic operations as the addition and product. Symbolics is an interesting feature of Julia, which takes longer to compile and build up but is much faster afterward to find output states from the compiled model. This approach is tested on PETLION, but the results can also only be computed for the uncertain T_{amb} and not for other parameters.

4.2.3 Sensitivity Analysis using Central Finite Differences

As explained in the Mathematical Background in Sec. 3.2.3 it is possible to compute parametric sensitivities using Central Finite Differences. The parametric sensitivity vector $L(t)$ is given by a derivative and can thus be computed by running the whole nonlinear PETLION simulation $2 \cdot n_\theta$ times, slightly shifting the uncertain parameters to measure absolute deviation. A good feature of this approach is that it does not require the PETLION model to be simplified because the vectors of states Y and YP are not explicitly needed. The finite differences approach should be much more efficient than implementing the sensitivity equations and integrating them into the DAE model. Some checks are made below to verify that this approach is coherent with the expected result and has enough accuracy.

- **Informed choice of parameter finite step in differentiation**

  The choice of the parameter step size is crucial in implementing finite difference as it will affect the results. If the step chosen to compute the derivative is too small, there will be issues with accuracy because of machine round-off limitations ($eps \approx 2.22 \cdot 10^{-16}$). If the chosen step is too big, the parametric sensitivity found will not be accurate enough to fit the local approximation. The FD sensitivity code is initially run for multiple steps to check the alternatives and find a reasonable span. The results found are:

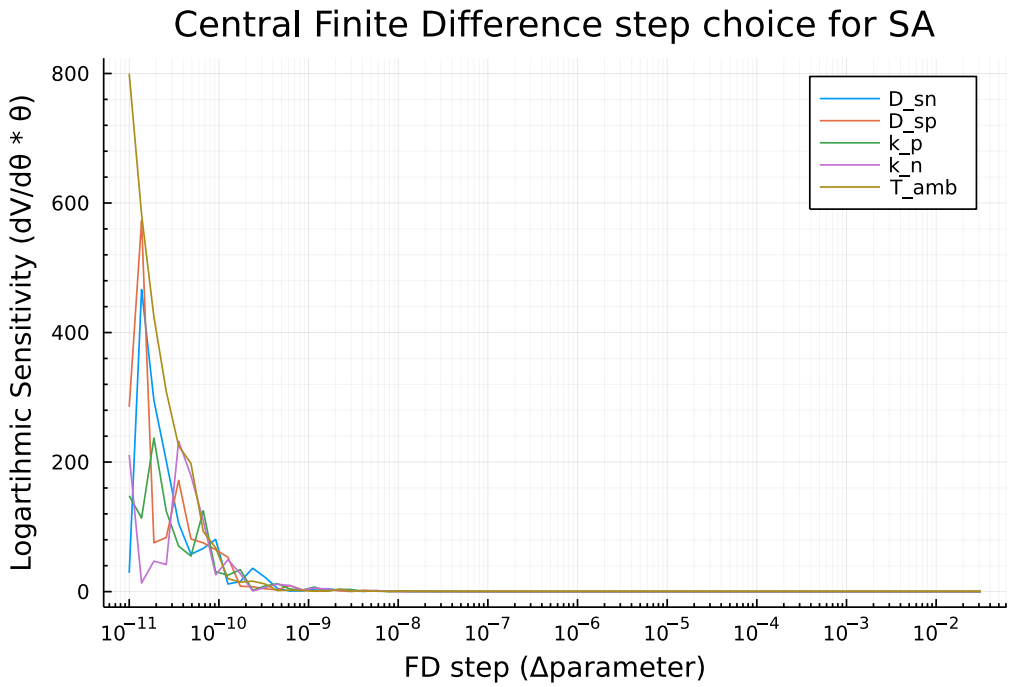

FIGURE 4.5: Parametric sensitivities of the output voltage on PETLION when charging a default Li-ion battery from SOC = 0% at I = 1C. The sensitivity is computed using second-order accurate CFD for multiple parameter steps ranging from 10^{-11} to 0.1. The step $\Delta parameter$ is the one used to model $\psi(\theta \pm \Delta\theta])$ on the finite differences Eq. (3.7).

From Fig. 4.5, we can deduce that any step coefficient in the range from 10^{-2} to 10^{-8} is a good fit for the model. The sensitivity results are stable in this range, justifying that the orders of magnitude are the accurate ones. This result is coherent with the expected minimum proper step being of the order of magnitude of the squared root of the machine precision error: $\sqrt{\epsilon} \approx 1.5 \cdot 10^{-8}$. The step chosen for all the FD methods in the whole project is $\Delta\theta = 10^{-3} \cdot \theta$, which is always applied proportional to the actual parameter.

- **Accuracy of chosen second-order Central Finite Differences vs. higher orders**
  The Central Finite Differences option is chosen to compute the numerical approximation of $L(t)$. The order chosen is second-order because it does not have a high computational cost. In this case, the whole nonlinear model is run exactly two times for every uncertain parameter in the model. For a reduced number of parameters, which is coherent with this study since the number of uncertain parameters is between one and five, the time needed to run the model will always be similar in order of magnitude to the time required for a single run. The relative error of different orders is compared to the eighth-order accurate CFD approximation (considered the most accurate option) to verify that the precision of the second-order model is enough. The relative error is shown in Fig. 4.6:

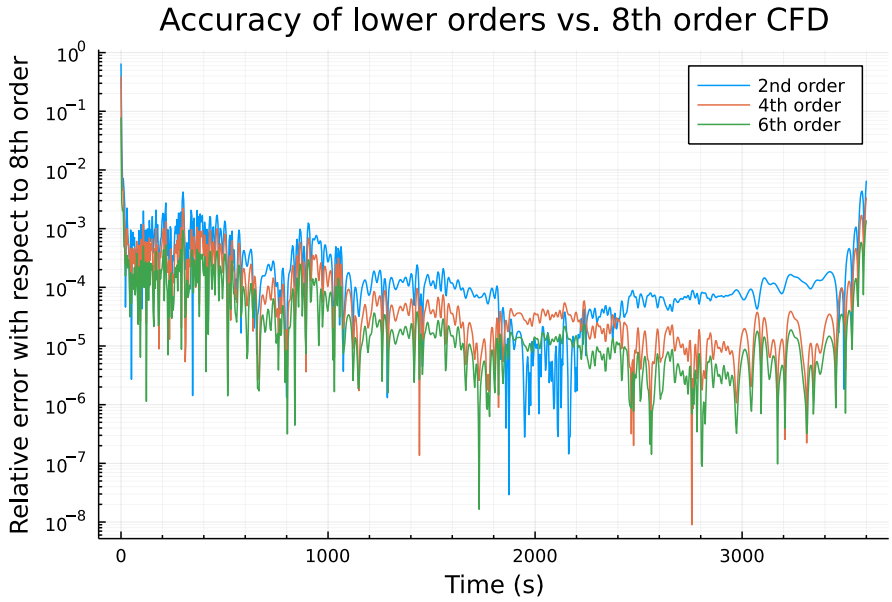

FIGURE 4.6: Accuracy of different Central Finite Difference methods with respect to 8th order CFD. Error is computed relatively as $||sens_8|-|sens_i||/|sens_8|$ where $sens$ is the parametric sensitivity $\partial V/\partial T_{amb}$.

The result is that the second-order CFD is accurate enough since the relative error compared to the eighth-order derivative is mostly below 10^{-3}. The initial relative error is higher since the voltage sensitivity is null at the start (as can be seen in Fig. 4.4).

- **Accuracy of Central Finite Differences against Sensitivity Equations**

  The accuracy of the CFD approximation for computing sensitivity parameters can be checked against the original approach with sensitivity equations based on (3.6). Computing the absolute error between the second-order accurate CFD sensitivity of $\partial V/\partial T_{amb}$ and the original sensitivity equations result shown in Fig. 4.4 gives:

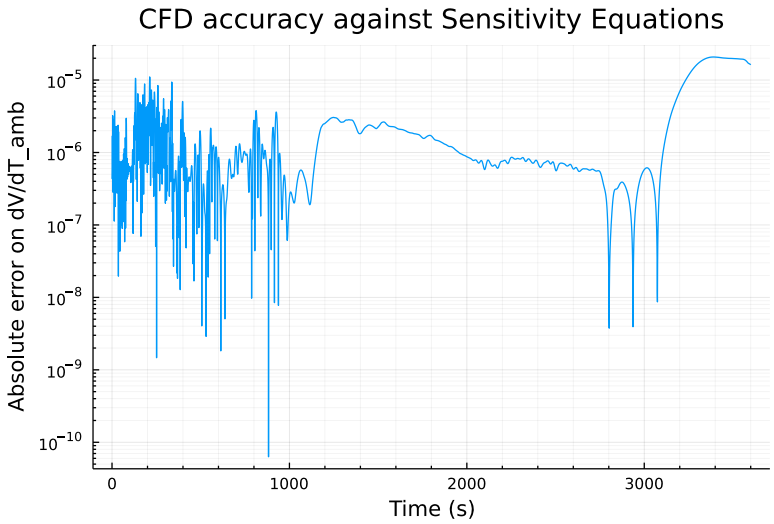

FIGURE 4.7: Absolute error of second-order accurate CFD with respect to original Sensitivity Equations approach for the simple charging path $I = 1C$. Error is computed as $||sens_{SE}| - |sens_{CFD}||$ where $sens$ is the parametric sensitivity $\partial V/\partial T_{amb}$.

  The absolute error of the CFD approximation shown in Fig. 4.7 is almost lower than 10^{-5} for the complete simulation, which is highly accurate. There is some oscillation noise in the initial error that can be related to the interpolator function that the CFD method employs to obtain the same comparable time points on each iteration[3]. The small oscillation of the parametric uncertainty is not alarming as the overall value is accurate for the whole simulation. The complete root-mean-square deviation is $rmse = 6.22\mu V$ (for a voltage range going from $2.75V$ to $4.20V$) proving that CFD is highly accurate.

4.3 Uncertainty Quantification of PETLION's states

The parametric sensitivities computed so far focus on finding the values that define the sensitivity vector $L(t)$. This vector contains the sensitivity of any output state concerning the uncertainty of the parameters that define the battery. Following the notation from the mathematical background, the output states' probability distribution can be explicitly determined when they are modeled as a first-order Taylor expansion $\boxed{\delta\psi = L\delta\theta}$. In this case, the theoretical

[3]The function to interpolate specific output time points in PETLION is Spline1D from the Dierckx.jl library

probability distribution of the output states is known and given by Eq. (3.2). The accuracy of this mathematical linear approximation can be tested against the results of running the whole nonlinear PETLION model with uncertain parameters using Monte Carlo sampling.

For the same charging method given by $I = 1C$, $SOC_0 = 0$, and uncertain parameter $T_{amb} \sim Normal(25\,°\text{C}, 1\,°\text{C})$ the distribution of any state such as the voltage can now be mathematically determined. The probability distribution function of the voltage against time can be found knowing the main curve of $V(t)$ and $L(t) = \frac{\partial V}{\partial T_{amb}}$:

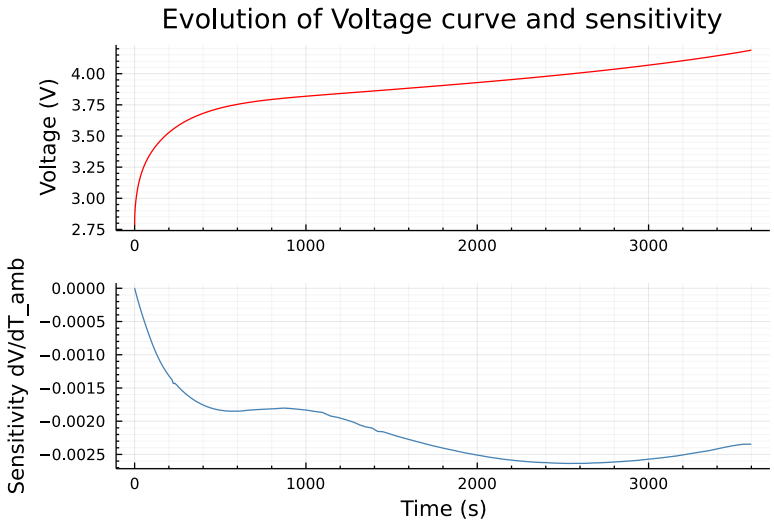

FIGURE 4.8: Nominal voltage and its sensitivity with respect to the variation on ambient temperature computed using second-order accurate Central Finite Differences for a complete $I = 1C$ charge.

Using these and the definition of $f_{p.d.}(\psi)$ and V_ψ we can determine the pdf curve at any time t:

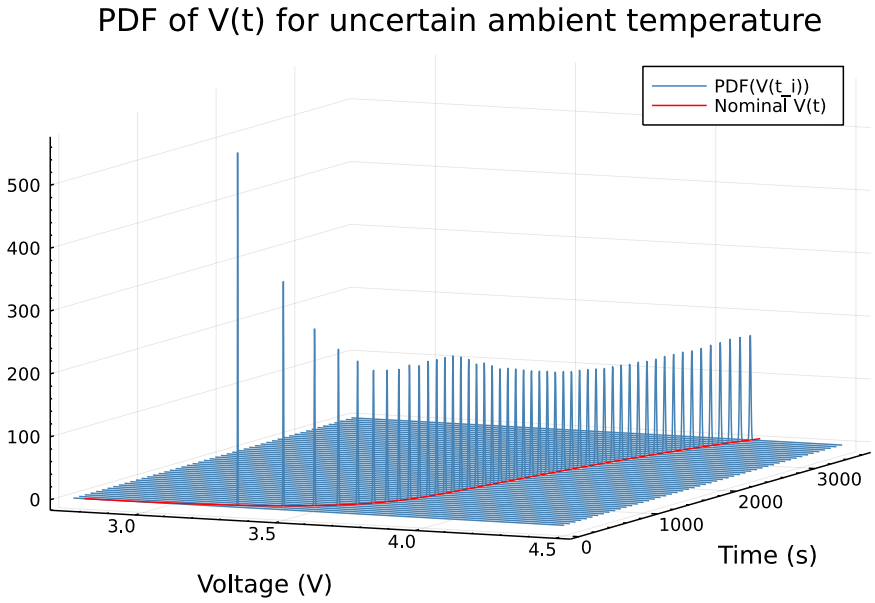

FIGURE 4.9: Mathematical probability distribution of the voltage when charging a default Li-ion battery in PETLION from SOC = 0% at I = 1C for an uncertain T_{amb}. The $pdf(V(t_i))$ is computed using second-order accurate Central Finite Differences for different points in time along the charging curve.

The results in Fig. 4.9 are a good indicator of the parametric sensitivity's evolution over time. This sensitivity variation is inversely related to the height of the waterfall's probability distributions so the dispersion of the voltage gets wider as the charging evolves in time. The progress of the pdf shown is consistent with the sensitivity results found from the sensitivity equation's method (Fig. 4.4) that are verified for the CFD approximation. The sensitivity $\partial V/\partial T_{amb}$ is initially small (pdf close to a delta function), then it grows to a local maximum around $t = 500s$, is slightly reduced and finally grows until the battery is almost fully charged. This preliminary results show that the voltage is susceptible to the changes in an uncertain ambient temperature. The output probability distribution of any state can be mathematically computed for the first-order approximation using the sensitivity vector computed with CFD.

4.3.1 Accuracy of Central Finite Differences against Monte Carlo sampling

We have verified so far that the numerical approximation using CFD is accurate to sensitivity equations. It is important to verify that the results obtained with this linear mathematical approximations fit the dispersion of the original nonlinear function. This can be checked using Monte Carlo. To determine the real dispersion of the voltage when the ambient temperature is uncertain the PETLION simulation is run for a set of $50k$ different ambient temperature samples drawn from $T_{amb} \sim N(25, 1\,°\text{C})$. The CFD sensitivity on the final voltage shows that $\hat{V}_{end} = 4.189\,\text{V}$ and its computed standard deviation is $\sigma_V = \sqrt{V_\psi(t)} = \sqrt{L(t)\boldsymbol{V_\theta} L(t)^T} = 0.002\,\text{V}$. The normal pdf given by this mathematical distribution $V \sim N(\hat{V}_{end}, \sigma_V)$ is displayed over the histogram of the 50k Monte Carlo nonlinear samples:

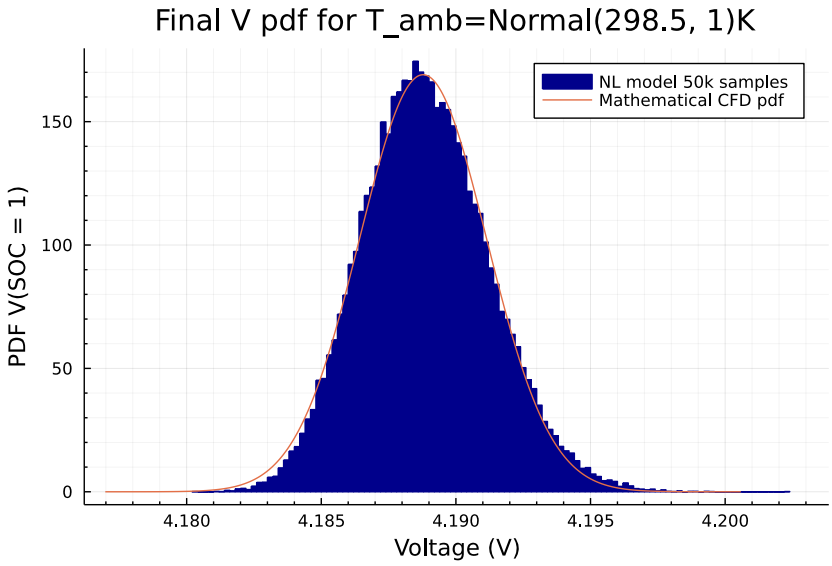

FIGURE 4.10: Probability distribution of the output voltage in the PELTION model for an uncertain ambient temperature when charging a default Li-ion battery from SOC = 0 at I = 1C. The pdf is computed using 2nd order CFD and 1st order SA and the histogram shows the distribution of 50k Monte Carlo samples with different ambient temperature conditions.

The results from Fig. 4.10 are encouraging as it seems that the linear approximation fits the nonlinear model with high precision for the basic CC charge and the chosen parametric uncertainties. To further check this accuracy, the percentage of samples that fit into every set of standard deviations from the mathematical mean can be computed:

TABLE 4.1: Accuracy of sensitivity analysis using CFD SA against original nonlinear PETLION model. Check of theoretical mathematical model mean and standard deviation of the final voltage. Check of the percentage of samples that fit into each span of standard deviations from the mean.

	Mathematical Approx.	MC samples
Mean ($\hat{V}$)	4.18878	4.18887
Std (σ)	0.00236	0.00237
1 std	68.269%	68.468%
2 std	95.450%	95.324%
3 std	99.730%	99.576%
4 std	99.994%	99.982%

This table proves the fitting accuracy for various ranges, which was visually apparent on the histogram dispersion. The mathematical values for the percentage of samples are well known for a normal distribution. Once the final nominal voltage and its standard deviation from the mean are known, the percentage of samples that fall into each standard deviation can be computed. The computed percentages are highly accurate to the mathematical value.

This result verifies that the mathematical approximations for Sensitivity Analysis and Central Finite Differences can accurately quantify the uncertainty propagation of the voltage in a continuous battery charge ($I = 1C$) over the PETLION model. The important conclusion is that this methodology is computationally efficient and could be applied to determine the dispersion of any output state for any parametric uncertainties. The computational time to find the sensitivity of the voltage against the ambient temperature is $\sim$ **0.33s** which is a significant improvement compared to the sensitivity equations alternative, which was more than 90 minutes. The direct approach of using Monte Carlo to sample fifty thousand different ambient temperature charges took $\sim$ **20 minutes**. This time difference indicates the benefit of using sensitivity analysis to quantify the output uncertainty instead of computing the actual output for every possible input uncertain parameter. In summary, computing the mathematical pdf from the second-order Central Finite Difference sensitivities is approx 3600x faster than finding the histogram dispersion from the direct samples.

4.3.2 Second-order expansion for Sensitivity Analysis

Following the mathematical background, it is known that the variation on any of the variables/states of the model when one or multiple parameters are uncertain can also be modelled as a second-order expansion: $\delta\psi = L\delta\theta + \frac{1}{2}\delta\theta^T M\delta\theta$. Although this approximation is more precise than the first-order approach, it is more computationally expensive since the sensitivity matrix $M(t)$ must also be computed.

In this case, though, even if the probability distribution of the uncertain parameters is known and follows a multivariate normal form, the analytical variation on the output is unknown. Other methods such as Monte Carlo can determine the probability distribution found in the results with a more accurate model. This repetition approach can also verify the analytical approach taken on the first-order simplified model.

Since the first-order linear approximation seems highly accurate, this project does not use the second-order approach. It could be an exciting method for future work and development.

4.3.3 SA by sampling original nonlinear model

A final insightful approach to quantify the effect of uncertainties in battery operations is to use direct sampling of the complete nonlinear DAEs model given by PETLION. This method is highly accurate as it gives the exact results for any distribution of the parameters. The issue is that the complete charging/discharging process has to be re-run for every set of uncertain parameters to precisely determine the output states. The method is computationally expensive as it requires running a similar simulation multiple thousand times. Once the output result is known for a representative set of samples, the probability distribution of the states of interest can be studied to determine how significant the dispersion is and even try to fit them into a well-known probabilistic distribution (in this case, an expected normal pdf).

Sampling the original simulation is always an alternative to finding parametric sensitivities in nonlinear models. In this case, the direct sampling of the complete PETLION model serves as the reference result to verify the accuracy of the mathematical methods tested, just as shown above in Fig. 4.10.

4.4 General parametric sensitivities of PETLION

4.4.1 Uncertain parameter distributions in lithium-ion batteries

The fundamental reason for using stochastic optimal control is that batteries are not deterministic, so uncertainty in their parameter definition must be considered. The variability of the desired parameters must be known to integrate stochastic analysis into the model. In this case, the parameters follow a normal distribution given by their mean and standard deviation values. The uncertain parameters used in this report are given by $\theta = \{T_{amb}, D_{sn}, D_{sp}, k_n, k_p\}$ which are considered relevant parameters in the optimal charging process [2]. The ambient temperature is modeled again as a normal distribution with $\hat{T}_{amb} = 25\,^{\circ}\text{C}$ and $\sigma_T = 1\,^{\circ}\text{C}$. The specifically identifiable distributions of the solid-phase diffusion and reaction rate constants are characterized in detail as nonlinear distributions using different literature estimations [2]:

TABLE 4.2: Means and standard deviations on a logarithmic basis for battery simulation values [2]

	$\log_{10}\mu_i$	$\log_{10}\sigma_{\mu,i}$
$D_{s,p}$	-15.26	0.2513
$D_{s,n}$	-11.63	0.2841
k_p	-12.49	0.3007
k_n	-9.736	0.3283

These distributions for the parameters are thoroughly studied in previous work and should be used, since they are given by the literature, to define the input parameters' distributions. The significant values for this study are the standard deviation results for every single parameter. These are used to build the variance matrix V_θ that defines the original model uncertainty as a diagonal matrix since all the parameters are independent. Following the previous methodology, the sensitivity vector is computed first for the desired parameters in vector θ, and then the output distribution of any state can be calculated.

4.4.2 Basic charge sensitivity analysis with multiple uncertain parameters

The sensitivity analysis is applied on the same basic complete charging simulation with $I = 1C$ and $SOC_0 = 0$. In this case the vector of uncertain parameters is $\theta = \{T_{amb}, D_{sn}, D_{sp}, k_n, k_p\}$. The second-order accurate Central Finite Differences method is used to find the sensitivity vector of the voltage $V(t)$ against these five parameters. The standard notation is used for the ambient temperature as before, but the logarithmic derivative is used to compute the sensitivities over the other parameters as they are of orders $k_i \sim 10^{-11}$ and $D_i \sim 10^{-14}$. The parametric voltage sensitivities found are given by:

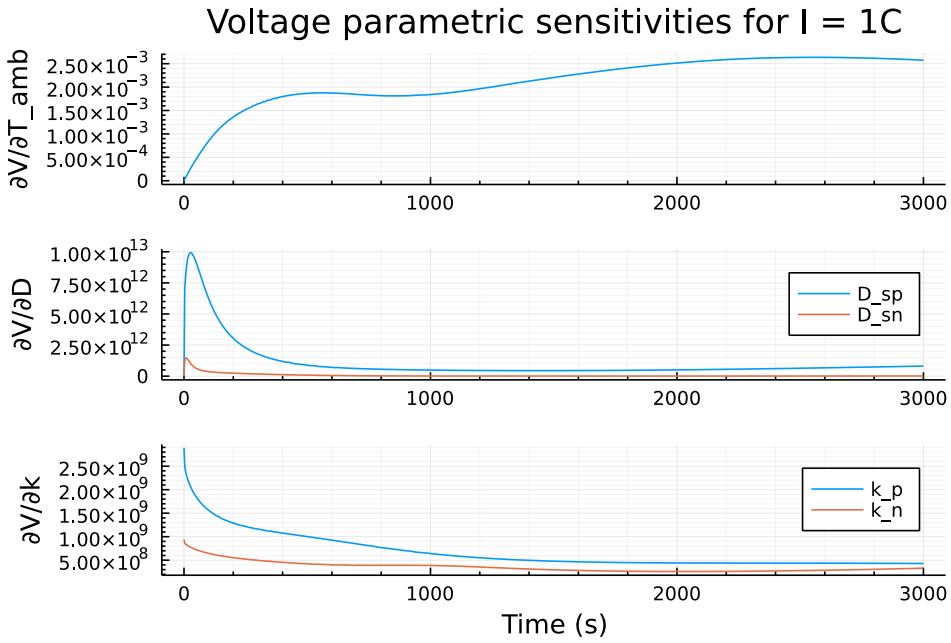

FIGURE 4.11: Sensitivity results of voltage for multiple uncertain parameters as a function of time. The sensitivities are computed for a simple full charge at CC I = 1C.

The results found in Fig. 4.11 show that computing the sensitivities of other parameters besides the ambient temperature is possible using CFD. The sensitivity vector has to be computed separately for each parameter, so the complete charging simulation is run $2n_\theta = 10$ times for second-order accuracy CFD. The same methodology can be directly used on other parameters of interest in PETLION, proving that the computation chosen is highly efficient since all the voltage sensitivities are found in $\sim$ **4.45 s**.

Even if the sensitivity values seem too large for the diffusion and reaction rate constants, they are coherent when applied to these small parameter values. They are highly accurate on order of magnitude compared to the effect of the ambient temperature given that the parameter's orders are $T_{amb} \sim 10^2$, $D_i \sim 10^{-14}$, and $k_i \sim 10^{-11}$. A good indicator for general uncertainty quantification to summarise how much influence one parameter has in the dispersion of the voltage is the integral of the sensitivity curve proportional to the parameter's value:

TABLE 4.3: Sensitivity of the voltage for I = 1C full charge given from higher to lower effect. Computed as the integral of the each parametric sensitivity curve and scaled for each mean value.

	Value $\hat{\theta}$	V sensitivity
T_{amb} [°C]	25	1902.67
k_n $[m^{5/2}/(mol^{1/2}s)]$	$5.031 \cdot 10^{-11}$	53.58
k_p $[m^{5/2}/(mol^{1/2}s)]$	$2.334 \cdot 10^{-11}$	46.94
$D_{s,p}$ $[m^2/s]$	$1.0 \cdot 10^{-14}$	31.60
$D_{s,n}$ $[m^2/s]$	$3.9 \cdot 10^{-14}$	7.24

The values in Fig. 4.3 quantify the uncertainty effect of each parameter separately on the evolution of the voltage in the Li-ion charge. The ambient temperature has the highest significance, followed by the rate constants and diffusion coefficients. These values are a good preliminary indicator of how many states will be affected by a deviation in any input parameter. The previous methodology can be repeated to quantify the output probability distribution, defining the pdf of any state using the sensitivity vector and the variance matrix V_θ.

Issues with the defined normal distributions

When trying to determine the standard deviation of the nominal voltage for these new uncertain parameters using their sensitivity curves, it is found that the literature-estimated multivariate normal distribution is not accurate. If the values specified in the literature to define the probabilistic distribution of θ are used, the final dispersion range of the voltage at any point in time gives non-physical results. This result confirms the initial stated issue that parameter estimation in literature is not accurate and usually delimits the constants in ranges too wide for real applications. Again, this validates the importance of stochastic uncertainty quantification for this set of parameters and other interesting constants in the battery model, which cannot usually be fixed deterministic values.

This case study concludes that the notation used in this project is consistent and can be edited to include multiple parameters quickly following the same procedure. The only issue is that literature estimates for the input multivariate normal distribution are not physically accurate and thus cannot be used for the following results of the project. For the final results, the only uncertain parameter modeled is the ambient temperature which can be set as desired. For further alternative work including these parameters, new distribution functions could be chosen with more narrow standard deviations that have a subtle effect on the dispersion of the final states.

Chapter 5

Results on continuous-discrete charging

The case studies performed as preliminary results prove that second-order accurate Central Finite Differences is a good estimator for the sensitivity vector. Also, First-order Sensitivity Analysis can accurately predict the distribution of the states at any point in time. These studies are implemented on PETLION for a continuous charging method at a constant current I = 1C. This new section aims to determine if the methodology can be efficiently extended to mixed continuous-discrete operations while maintaining high accuracy.

The methodology used to determine the uncertainty propagation from the input parameters on the PETLION model begins by computing the nominal simulation. The desired charging path is modeled in PETLION following the documentation and setting the initial states, discrete boundaries, and continuous operational modes. Since the literature-estimated probability distribution of the parameters of interest cannot be applied in this simulations, the only uncertain parameter used is again $T_{amb} \sim N(25\,^{\circ}\text{C}, 1\,^{\circ}\text{C})$. The sensitivity of the simulation can then be computed following the procedure of the previous sections. For this results section, the deviations from the nominal values are computed using **$\pm$3 standard deviations** from the mean. This boundary in a normal probability distribution represents 99.7% of the possible outputs, meaning that the upper and lower bounds found here limit this percentage of possible values.

5.1 Uncertainty Quantification on a CC-CV charge

A basic path for optimal charging is CC-CV which uses maximum constant current as long as the voltage is below the desired safety bound. If the voltage limit is hit before reaching the desired State of Charge, the battery can be completely charged following a CV mode at

the maximum bound. Lithium-ion batteries can be damaged when charged outside their voltage limits, leading to premature aging, so it is important to ensure that this constraint is satisfied. To test the effect of uncertain parameters in this charge path, a basic model is defined in PETLION with $SOC_0 = 0\%$, $I_0 = 4C$, $V_{max} = 4.1V$, and $SOC_{max} = 60\%$. Following the procedure of the preliminary results, the new expected bounds on any state with 99.7% significance can be determined. In this specific example, an exciting insight could be to know when the voltage limit of 4.1 V is hit if the ambient temperature is uncertain. The sensitivity of the voltage curve can be computed, and the standard deviation can be found. Since the standard deviation is a function of time, it is possible to plot the unbounded voltage for $I = 4C$ as a ribbon with its ± 3 std boundaries:

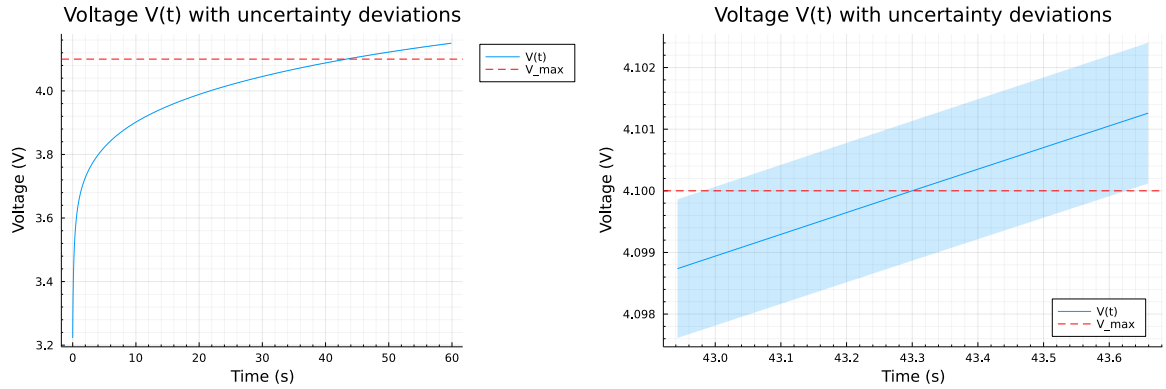

FIGURE 5.1: Nominal voltage for I = 4C charge and upper and lower bounds with 99.7% accuracy as a function of time using second-order accurate CFD. Left shows the complete evolution and right shows a zoom section where the upper and lower bounds actually hit the maximum limit of 4.1 V.

This initial analysis of the dispersion on the nominal voltage curve shows the significance of uncertainty analysis in battery modeling. The desired limit voltage is hit in a time-span $\Delta t \approx \pm 320\,\text{ms}$. Also, the real value of the voltage when the nominal limit happens can span in a range of $\Delta V \approx \pm 2\,\text{mV}$. This result shows that for a 1 °C standard deviation on the temperature, the uncertainty effect on voltage is small. This initial voltage example proves that any desired state can be bounded for specific accuracy constraints α to quantify uncertainty effects.

5.1.1 Sensitivity Analysis on a discrete step in CC-CV

It is essential to verify whether the mathematical notation and software methodology used for the preliminary results can also accurately determine the uncertainty propagation for a mixed continuous-discrete simulation. For this purpose, sensitivity analysis is applied to the complete CC-CV charging curve defined above. The primary approach starts by computing the complete curve with nominal parameters. The final methodology is used to design a function that can simultaneously compute the standard deviation of all the model's states. The output from this

function includes the upper and lower limit complete models instead of a single state. This new approach allows analyzing any state of interest afterward quickly and efficiently. This new system is highly efficient and works for any desired success range α using the required standard deviations from the nominal states.

The main CC-CV simulation output has deviated ± 3 std resulting in two limit models bounding 99.7% of the possible outputs. This deviation returns two new simulation outputs corresponding to the upper and lower boundaries. The computational cost of finding all complete model simulation outputs deviated for the lower and upper bounds is $\sim$ **59 s**. The most significant benefit is that all the states' deviations are already known with the current notation, so no more expensive computations are required afterward. Regarding fast charging using a CC-CV path, the time required to charge the stochastic battery goes from $1685s$ to $1738s$ considering 99.7% of probable states. The expected fast charge CC-CV time will happen in a span of 43s around the deterministic end time of $1711\,\mathrm{s} \approx 28\,\mathrm{min}$.

Once these limit models have been computed, any state or specification can be further studied. A typical state of interest in battery operations is temperature, which is a prominent surrogate for battery degradation. General operating modes will aim for low temperatures, avoiding fast degradation. Accordingly, the temperature should be kept below $80\,^{\circ}\mathrm{C}$ to avoid dangerous effects such as thermal runaway. Since the temperature is an essential factor in degradation, we can try to quantify the uncertain T_{amb} effect on the CC-CV battery's temperature. More specifically, we can analyze the average 1D cell temperature along time which is:

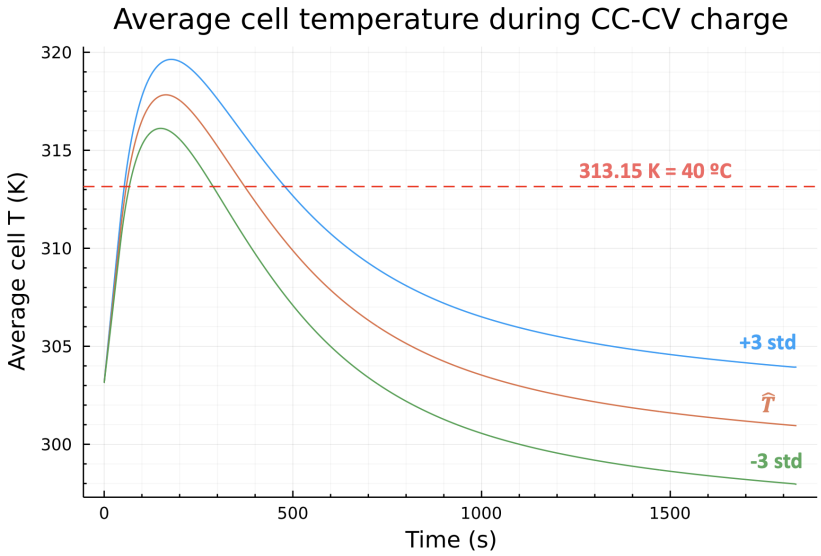

FIGURE 5.2: Average temperature of the battery cell during a CC-CV charge from 0% to 60% SOC. The lower and upper model bounds with 99.7% significance are found using CFD and linear SA. The $40\,^{\circ}\mathrm{C}$ line is plotted as a usual desired limit temperature in battery modeling to reduce degradation.

This figure shows how the average cell temperature is susceptible to the uncertainties in ambient temperature. The mixed continuous-discrete sensitivity analysis implementation was straightforward, using the information from the preliminary results. Initially, some issues with numerical errors appeared as noise and oscillations in the sensitivity curves. Given an accuracy issue on the PETLION library interpolation of values, this noise occurred in discrete steps. The interpolating function did not consider that the partial continuous operational modes equations differed after discrete bounds. This example helped detect this minor issue with the PETLION software for discrete steps, easily solved by changing the inner interpolating function to consider continuous parts separately.

5.1.2 Verifying results with MC sampling

Following the methodology of the preliminary results section, the actual accuracy of the mathematical approximation can be determined against a Monte Carlo sampling approach. As explained, to reduce battery degradation, we can determine how much time the battery charge is happening over 40 °C, a common upper bound on the temperature. From the limit curves and the nominal average temperature in Fig. 5.2, the time spent over 40 °C can be computed with a numerical integral. Plotting this mean value and standard deviation as the expected mathematical pdf over the Monte Carlo results for ten thousand samples:

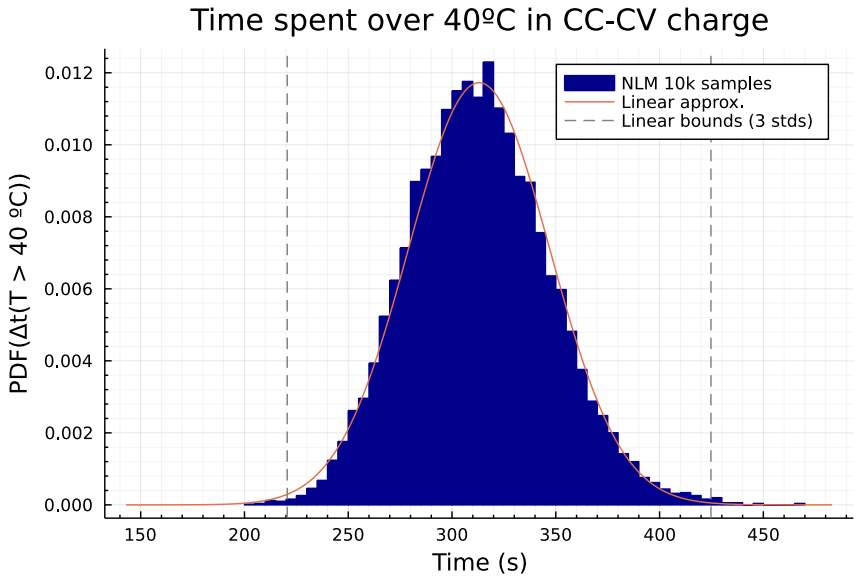

FIGURE 5.3: Probability distribution of the time that the CC-CV charging battery spends over 40 °C for an uncertain ambient temperature. The histogram is built from 10k Monte Carlo samples. The approximation pdf in red is built using second-order accurate CFD and linear SA.

Again, this result proves the significant accuracy and efficiency of the methodology used to approximate the output dispersion. For the 10k samples used, exactly **99.65%** fit inside the

upper and lower bounds found from the mathematical approximation in Fig. 5.2 against the expected 99.7%. Computationally, the Monte Carlo analysis for 10k samples requires ~ 10 **min** which is $10x$ more than the sensitivity analysis. This new result verifies the significance of linear sensitivity analysis to predict output probability distribution in a continuous-discrete charge such as CC-CV.

5.2 Charging along with non-measurable state bounds

An important advantage of PETLION is that it incorporates unconventional general operating modes such as CT, CPo, CCe, $CCss$, and $C\sigma$ previously described. The project has demonstrated successful approximations for battery charging using conventional methods of CC and CV. The same procedure can be extended using different mixed continuous-discrete operations to assess its significance further. The issue when implementing sensitivity analysis on any other charging paths is that other states are not measurable or controllable in real battery operations. Opposite from the current I, which is a directly *adjustable* input state and can be used to create a CC path. This new approach aims to find an alternative method to compute parametric sensitivities on battery charges for non-measurable boundary states such as the plating overpotential.

5.2.1 Results on an optimal CC-CPo-CV process

In addition to the main lithium-ion flows in battery reactions, other side elements interact in the cell, causing degradation during charge and discharge. Some anodic reactions cause lithium plating, which can limit the battery's capacity. These reactions occur when the anodic lithium plating overpotential is negative [3]. It is interesting thus to control whether the plating overpotential η_p becomes negative with stochastic uncertainties.

Previous fast control using PETLION in [3] suggests $CC-CPo-CV$ charge as an efficient method that would extend the battery lifetime. The protocol is based on $\{SOC_0 = 0\%,\ I_0 = 4C,\ V_{max} = 4.1V,\ \eta_{p_{min}} = 0V,\ SOC_{max} = 60\%\}$ and finds a fast charging method that avoids negative overpotential. The goal of this section is to determine how an uncertain ambient temperature $T_{amb} \sim N(25\,^{\circ}\text{C}, 1\,^{\circ}\text{C})$ can affect the output. The nominal charge results for this simulation without uncertainties are given in Fig. 5.4. The voltage and plating bounds are almost reached around the same time. When the plating boundary is hit, the full charging speed is reduced, and the voltage evolves until it reaches the maximum. The battery reaches the maximum 60% charge shortly after.

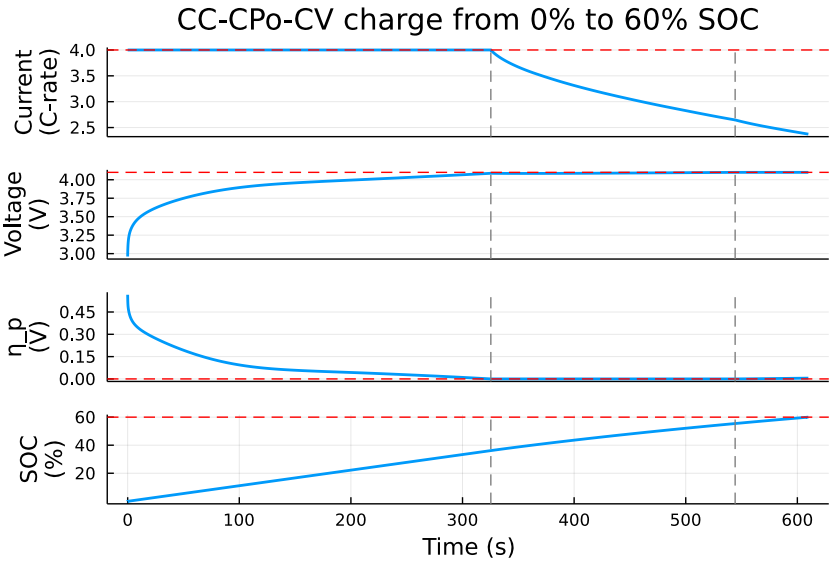

FIGURE 5.4: Output states $\{I,\ V,\ \eta_p,\ SOC\}$ for nominal CC-CPo-CV battery charge process. The plating overpotential boundary is hit at $t_1 = 325.38s$ and the voltage boundary is hit at $t_2 = 544.25s$.

Since the state of interest in this case is the plating overpotential, we can compute the dispersion of this state when the ambient temperature is uncertain. The evolution of the nominal η_p and the two limit simulations deviated by $\pm 3std$ are computed following the previous methodology:

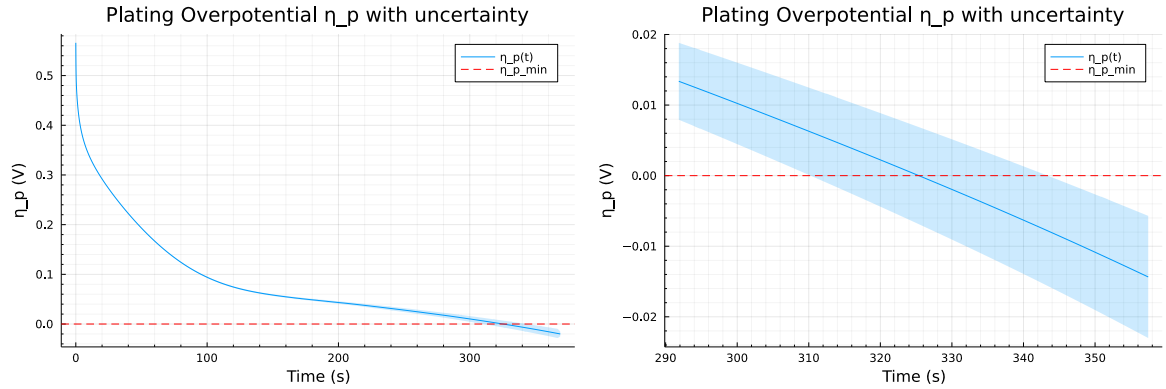

FIGURE 5.5: Nominal lithium plating overpotential for I = 4C charge. Upper and lower bounds with 99.7% accuracy as a function of time using second-order accurate CFD. The left shows the complete evolution, and the right shows a zoomed section where the limit states hit the negative boundary.

The results in Fig. 5.5 show that the actual point in time where this first bound is hit will deviate $\Delta t = \pm 16.6s$ from the expected deterministic point and can happen at $\Delta\eta_p = \pm 7mV$ difference. If the desired constraint was to avoid crossing over to negative potential values, the CPo mode should start at $\sim 310s$, the earliest bound. As explained, though, the lithium plating overpotential η_p is not a measurable state in batteries, so using it in deviated simulations to detect the actual points when this boundary is crossed would be unrealistic.

5.2.1.1 Charge state following measurable current curve I

As an alternative, the main current curve $\hat{I}(t)$ can be used as the reference in uncertain simulations. This method's benefit is that the current state can be used as an input for any simulation. The methodology is:

1. Computing the nominal simulation output starting at $\{I_0 = 4C, SOC_0 = 0\%\}$ and following the 3 desired discrete constraints $\{V_{max},\ \eta_{p_{min}},\ SOC_{max}\}$
2. Define from this main output states a function $\hat{I}(t)$
3. Compute the CFD parametric sensitivities by simulating a charge state that follows a current function $I = \hat{I}(t)$ with only 2 discrete constraints $\{V_{max},\ SOC_{max}\}$

This methodology gives three final model results: the nominal curve and the upper and lower bounds. From these models, we can now plot the evolution of any state such as η_p. As we can see from the dispersed output distribution in Fig. 5.6, the lithium plating overpotential reaches negative values when there are uncertainties in the ambient temperature. To test the accuracy of this dispersion and see if it fits 99.7% of the samples, we can also analyze this method using Monte Carlo. In this case, the samples used are also modeled following the nominal curve $I = \hat{I}(t)$. We can predict that using this curve instead of directly tuning the CPo curve will affect the distribution of the outputs and make the linear approximation less accurate as time progresses.

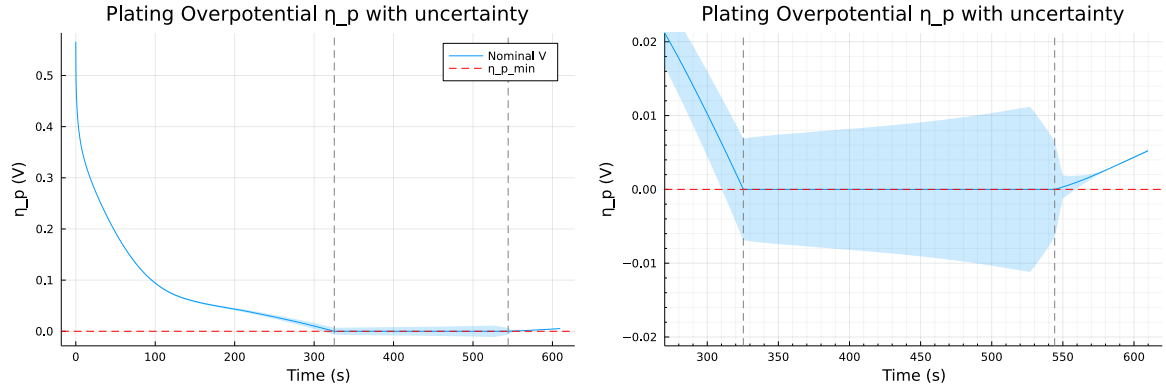

FIGURE 5.6: Nominal lithium plating overpotential for I = 4C charge. Upper and lower bounds with 99.7% accuracy as a function of time using second-order accurate CFD. Left shows the complete evolution and right shows a zoomed section where the limit states actually hit the negative boundary.

The mathematical linear approximation of the probability distribution can be plotted over the Monte Carlo samples of the lithium plating overpotential at different time points (Fig. 5.7). The fitting of the Monte Carlo samples for the plating overpotential is not as accurate as of the previous measurable results. This result was an expected limitation of using linear approximations in non-measurable states. Nonetheless, the expected probability distribution does serve

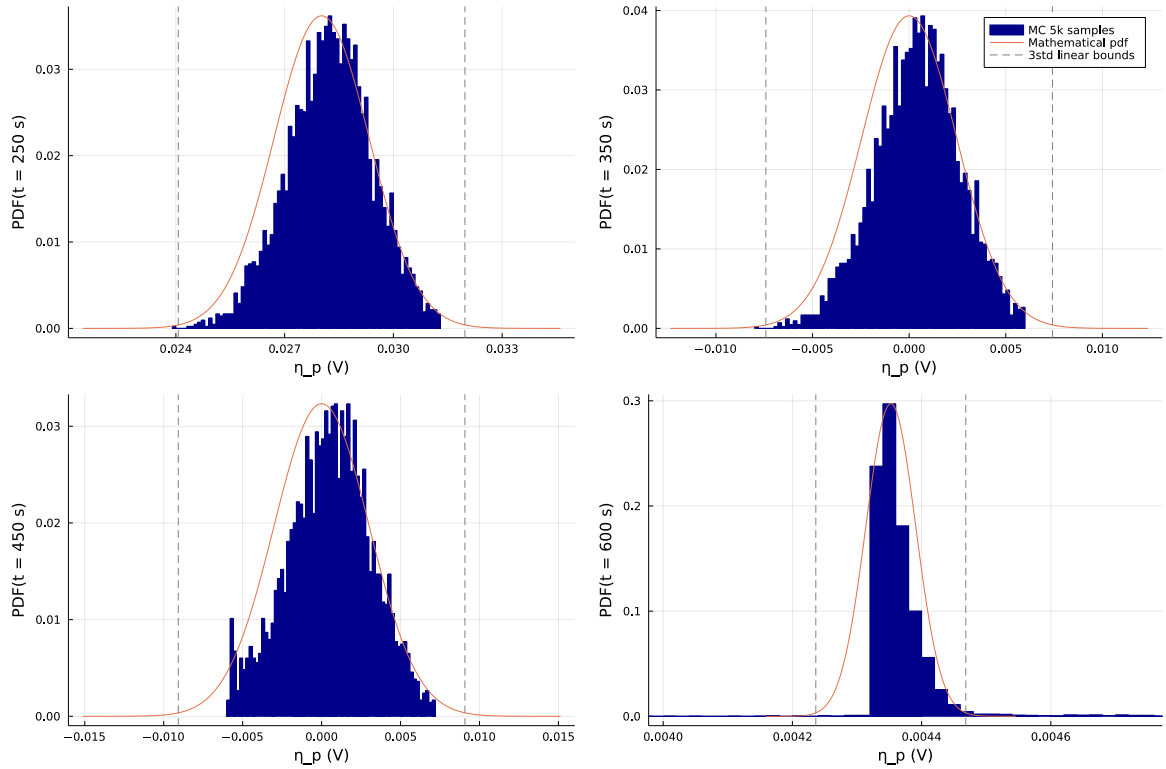

FIGURE 5.7: Comparing the mathematical probability distribution of the lithium plating overpotential for the CC-CPo-CV model against the Monte Carlo approach for 5k samples. Upper and lower bounds with 99.7% accuracy as a function of time using second-order accurate CFD. From top left to bottom right the distributions show the overpotential at times $t = \{250, 350, 450, 600\}s$.

as a realistic boundary at different times according to Fig. 5.7. The highly focused distribution at $600s$ is coherent with the CV charging mode. The plating is directly related to the applied voltage, so the possible dispersion is lower. Considering the voltage constraint can explain the fact that all the real distributions are slightly skewed to higher overpotential values. If we plot the voltage curve for the linear approximated 99.7% success limit states:

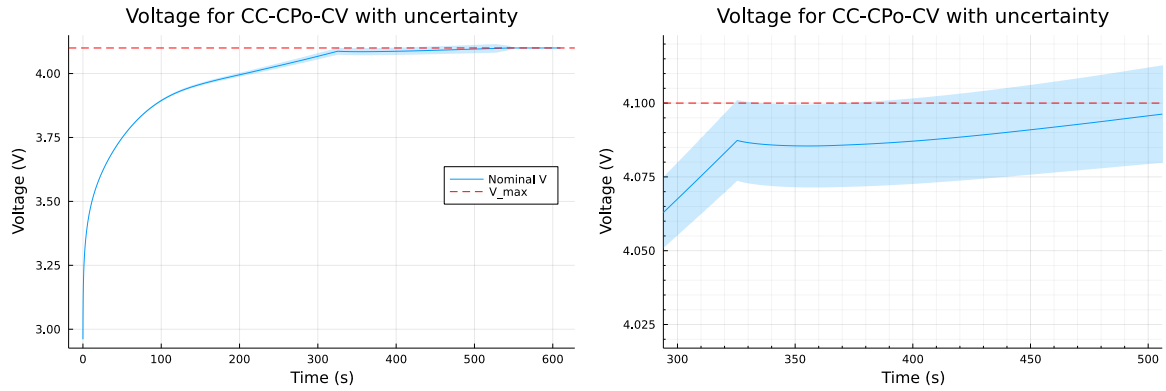

FIGURE 5.8: Voltage curve for CC-CPo-CV when simulation follows nominal current curve for $t < 545s$ and $V < V_{max}$. Upper and lower back-off bounds shown for 99.7% significance. Left shows the complete charge and right shows a zoomed section where the limit states hit V_{max}

We can see on Fig. 5.8 that the predicted upper limit for this methodology is reaching the

maximum voltage condition around $\sim 325s$. Some simulations will hit the voltage limit much earlier than the primary curve. More specifically, the voltage limit is hit at $+2.72\sigma_V$, which means that $\sim 0.33\%$ of the deviated simulations will never hit the plating constraint and follow a CC-CV pattern instead.

5.2.1.2 Charge state following measurable current considering V_{max}

The CV charge path goes at a much slower rate and delays the evolution of the other states. When this last operational mode begins, the lithium plating overpotential increases rapidly (as can be seen from the nominal curve in Fig. 5.6 for $t > 550s$). To better understand how this boundary is affecting the dispersion of the model, the deviated simulation that reaches higher voltages is regressed to the original $V > V_{max}$ point, and the charge is completed only at CV. This simulation gives new bounds on the plating overpotential:

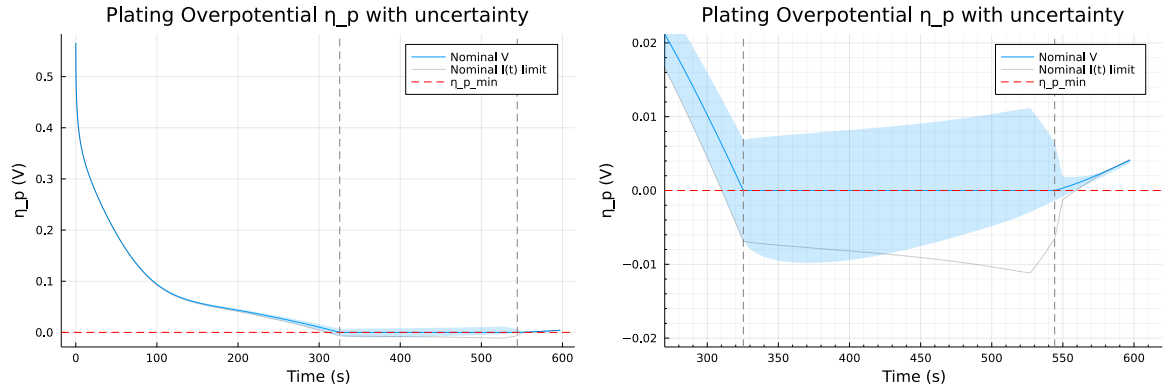

FIGURE 5.9: Plating overpotential curve for CC-CPo-CV considering that lower plating limit actually follows a CC-CV charge. Upper and lower bounds shown for 99.7% accuracy as a function of time using second-order CFD. Left shows the complete evolution and right shows a zoomed section.

This analysis shows that the lower boundary for 99.7% significance on the plating overpotential follows a CC-CV charge path. The maximum voltage limit is reached before the lithium plating overpotential becomes negative. This behavior can explain the Monte Carlo skewed distributions in Fig. 5.7 since the samples running at lower plating potentials are reaching the V_{max} limit earlier, which increases the value of the plating rapidly, moving the complete probability distribution curve to the right. This extended analysis explains the results of including stochastic uncertainties in a CC-CPo-CV charge.

To sum up, the usage of sensitivity analysis can provide great insight for understanding the effect of uncertain parameters. The complementary study of the deviations on both limits can accurately explain the new possible states for 99.7% of uncertain batteries. Also, the computational cost of the linear approximated limit states is $\sim 71s$ while the Monte Carlo for 5k samples takes $\sim 64min$. Coherently, more complex simulations take longer to run on PETLION: 10k

samples of the CC-CV charge required only $\sim 10min$. This issue makes stochastic uncertainty quantification with linear sensitivity analysis an even better alternative when modeling more and more complex methods. In the examples used so far, the computational time of sensitivity analysis is of the order of ~ 1 min while Monte Carlo sampling for several hundreds of replicas takes at least 10 min and is directly dependent on the number of samples.

5.3 Degradation effects on long-term battery charging

To assess further the computational cost of more complex optimal charging paths, we can include longer simulated battery operations. We have designed a PETLION simulation for long-term battery degradation caused by continuous charge-discharge cycles. This simulation uses $I_{max} = 2C$ and $V_{max} = 4.2V$ as general bounds. The lithium-ion battery is fully charged following CC-CV paths and discharged completely during 500 cycles. The deterministic simulation alone takes ~ 8 min. If we wanted to quantify uncertainty effects on these 500 cycles using Monte Carlo sampling, the computational time for only 1k samples would be of the order of ~ 6 days, clearly unreasonable. We can use linear sensitivity analysis to estimate the final uncertainties faster. Since this is a computationally intense simulation, we can use the original methodology to compute the standard deviation of a single state. Given that we want to study long-term degradation, a meaningful value is the State Of Health (SOH [%]). This state measures the maximum battery charge concerning its original total capacity and directly indicates its aging degradation.

For the deterministic simulation of 500 cycles, the final SOH was $76,5\%$. Using the methodology chosen with second-order CFD to approximate linear sensitivity analysis when the ambient temperature is uncertain, the final SOH is bounded for 99.7% of the batteries between 75.8% and 77.1% SOH. The computational time required to find this dispersion was ~ 84 min considering both Julia compilation and run time. Even if this is a considerably high time, it is still much faster than Monte Carlo and offers a very efficient approach to stochastic uncertainty quantification in highly complex battery simulations.

Chapter 6

Conclusions

This project integrates stochastic uncertainties in lithium-ion battery simulations by defining model parameters, such as the ambient temperature, using normal probability distributions. The propagation of these uncertainties onto the output states is quantified using linear sensitivity analysis and computed with second-order accurate Central Finite Differences. The main conclusion derived from this study is that the methodology designed can efficiently and accurately quantify stochastic uncertainty effects on some battery simulations. This methodology can be easily extended to any desired operational mode available in PETLION for multiple Li-ion battery materials, usage conditions, and uncertain parameters. The sensitivity results for each model can define safety specifications better and limit manufacturing parameters with tighter constraints.

The case studies presented demonstrate the proposed methodology's accuracy and computational efficiency on different charging paths. Stochastic optimal control is applied to complete CC-CV and CC-CPo-CV optimal charges for uncertain ambient temperature with computational times around 1 min. The cases studied also prove that the uncertainty in battery model parameters can cause significant state variation on optimal charging protocols. The states' dispersion is noticeable when performing stochastic analysis with probabilistic parameters, proving that a deterministic approach does not suffice. Using sensitivity analysis on optimal control also gives greater insight into how each parameter affects the evolution of the battery states.

Moreover, the example studies shown in this project verify the mathematical accuracy of sensitivity analysis for uncertainty quantification, especially for measurable states. Firstly, it is verified that a Taylor expansion on the value of states against the deviation of the model parameters is an accurate estimator of the state dispersion $\delta\psi$. Secondly, it is shown that a linear

first-order expansion can characterize the nonlinear model dispersion and that the mathematical expected normal distribution is fulfilled for measurable states. Finally, this work demonstrates that second-order Central Finite Differences can be an excellent numerical estimator for sensitivity analysis in measurable case studies. When using sensitivity analysis to quantify the uncertainty propagation on an optimal CC-CPo-CV charge, we find that discrete transitions between non-measurable operating modes can be incompatible with the linear approximation. This precision loss shows the limitations of linearized modeling for non-measurable states, especially using them as charging constraints. Nonetheless, even with lower accuracy on non-measurable states, the fitting of the linear approximation is high enough to bound the desired states successfully. Using the linear approximation, we can estimate the probability that the charging battery crosses the negative lithium overpotential boundary. We could likewise estimate the time spent below this boundary and this state's average value, correlated with long-term battery degradation metrics.

Overall, the benefits of implementing sensitivity analysis are clear, and they can lead to considerable advantages in battery modeling for increased understanding and safety. The most significant results found are the methodology and procedure implemented and verifying that they are successful. Sensitivity analysis can accurately predict the probability distribution of most states in battery modeling with efficient computational times. The methodology to integrate stochastic uncertainties in simulation is a powerful tool much faster than Monte Carlo sampling. The computational time of Monte Carlo compared to sensitivity analysis was 5x faster on a CC-CV charge, 54x for CC-CPo-CV, and 470x for the long term 500 cycles model (all times given proportional for five thousand samples in Monte Carlo for a single uncertain parameter).

6.1 Limitations of the current work

Some limitations were found with the stochastic uncertainty quantification methodology designed. A significant issue in this study is that the literature estimated parameter distributions for lithium-ion batteries give non-physical results when applied in the simulation. This problem indicates the significant uncertainty associated with battery modeling since the parameters can span several orders of magnitude according to different estimation methods. Further work should be done to define better the multivariate normal probability distribution of lithium-ion battery parameters. Knowing accurate distributions should be the essential requirement to make stochastic uncertainty quantification significant in real applications.

Lastly, a critical issue that should be noted is that the results shown here are based on specific test cases with some limited parameters, operating boundaries, and discrete constraints. Further testing of other optimal control charging paths is required to assess stochastic optimal control's general accuracy and significance. Possible new approaches that could extend the applications of stochastic optimal control include new uncertain parameters with various standard deviations, more output states of interest, and testing new mixed continuous-discrete charging protocols.

6.2 Future work or alternatives

This thesis serves as a base for stochastic uncertainty quantification in optimal control of lithium-ion batteries. It is an extension of previous studies that only used deterministic parameters. One essential application of sensitivity analysis in battery modeling could be its real-time usage on remote servers or onboard. Using this methodology onboard would require computing the sensitivity results in real-time (the methodology proposed can only find the deviated states after the complete simulation is known). If this real-time application is successful and if the software is fast enough, efficient, and accurate, it could be integrated into portable devices or electric vehicles to predict deviations on the fly. This application could potentially increase the safety of lithium-ion batteries and prevent irreversible damage to the cells. It would represent an exciting development of Advanced Battery Management Systems (ABMS).

An exciting path for future development that is left open in this project is the further study of sensitivity equations. This mathematical method for sensitivity analysis could be much more computationally efficient and lead to precise results. Some improvement suggestions in the preliminary case studies could be implemented in future research, such as defining the Jacobians using sparse matrices or employing a symbolic approach to the complete PETLION model. Sensitivity equations could give results in real-time and thus be integrated for the onboard applications previously described.

Lastly, besides linear sensitivity analysis, many other mathematical methods could be implemented to quantify uncertainty and carry out stochastic optimal control. One of the original ideas was using Polynomial Chaos Expansion (PCE), a popular alternative method for UQ. Some literature was investigated, but the mathematical theory's implementation was not trivial. Exploring the integration of Polynomial Chaos Expansions to propagate uncertainties in stochastic batteries remains an option for future work.

Appendix A

Chemical simple example

The chemical example analyzed was solved using multiple approaches as a 1st attempt to compute parametric sensitivities on an ODE. The easiest option is to use Julia built-in sensitivity libraries such as *DiffEqSensitivity* and *ForwardDiff*. These libraries are very useful to determine sensitivity of simple equations that can be expressed in semi-explicit form with a mass matrix for the derivative terms. It works perfectly for continuous functions but can't be applied to discrete-continuous processes such as PETLION, which is also built in a fully implicit form and optimized in a way that the DAEfunction isn't accessible to use this kind of libraries. The code for building Sensitivity Analysis using built-in libraries is:

Listing A.1: 'SA of basic chemistry reaction using Julia sensitivity open source libraries'

```
using DifferentialEquations, DiffEqSensitivity, ForwardDiff

function f(dC,C,p,t) # Definition of the DAE in semi-explicit form
    C_A, C_B, C_C = C
    C_A0,k_1,k_2,k_3 = p
    dC[1] = -k_1*C_A
    dC[2] = k_1*C_A - k_2*C_B + k_3*C_C
    dC[3] = sum(C) - (0 + 0 + C_A0)
    nothing
end

M = [1.0 0 0; 0 1.0 0;0 0 0] # Mass matrix
p = [1.0, 1000, 1, 1] # Parameters main values
vars = (:C_A0, :k_1, :k_2, :k_3) # Parameters names

u0 = [1.0,0.0,0.0] # Inital concentrations
tspan = (0.0,0.02); # Time span with main changes for rapid reaction 1
N = length(u0); M = length(p)

func = ODEFunction(f, mass_matrix = M)

# Solving DAE with Forward Sensitivity added
prob = ODEForwardSensitivityProblem(func, u0, tspan, p; sensealg=
    ForwardDiffSensitivity())
```

```
@time sol = solve(prob, Rodas5(autodiff=false); reltol=1e-8, abstol=1e-8)

# Solving DAE without Forward Sensitivity
prob_no_sens = ODEProblem(func, u0, tspan, p)
@time sol_no_sens = solve(prob_no_sens, Rodas5(autodiff=false); reltol=1e-8, abstol=1
    e-8)
```

From this first approach, the resolution of the basic DAE without sensitivity analysis takes **857** μ**s** and the one including parametric sensitivities takes **1341** μ**s**. This result has a coherent order of magnitude since the latter has to solve a slightly larder ODE but with the same specifications and general terms.

Once this approach has been tested and the results were known, the Sensitivity Equations had to be built manually for the DAE model (which is the one equivalent to the notation of the PETLION software). The complete code used is:

Listing A.2: 'SA of basic chemistry reaction using Sensitivity Equations to expand the DAE system'

```
# Goal - Complete the Sensitivity Analysis on the system of DAEs
using Sundials, DifferentialEquations, ForwardDiff, FiniteDiff, NLsolve, Plots

# Simulate the concentration of species A, B and C in ctt Volume batch: A -> B <-> C
## ************** DAE model ******************
# dCA/dt = - k1*CA
# dCB/dt = k1*CA - k2*CB + k3*CC
# CA(t) + CB(t) + CC(t) = CA(0) + CB(0) + CC(0)

CA_0, CB_0, CC_0 = 1.0, 0.0, 0.0; # concentrations [mol/L]
k1, k2, k3 = 1000.0, 1.0, 1.0;    # rate constants [1/s]
# Set of parameters and values that include uncertainties / do the SA:
θ = [CA_0; k1; k2; k3];

function F(du, u, p, t) # Basic DAE model with in-place residual output
    CA_0, k1, k2, k3 = p;
    CA, CB, CC = u
    dCA, dCB = du
    [-k1*CA - dCA, k1*CA - k2*CB + k3*CC - dCB, CA + CB + CC - (CA_0 + CB_0 + CC_0)]
end

function F_sens(res, du, u, p, t) # Extended DAE for sensitivities
    n_params = size(p,1); n_terms = Integer(size(u,1)/(n_params+1))

    y = u[1:n_terms]; yp = du[1:n_terms] # Function states x and dx
    res[1:n_terms] = F(yp, y, p, t) # Function residuals

    jac_Fx = ForwardDiff.jacobian(x -> F(yp, x, p, t), y)
    jac_Fxp = ForwardDiff.jacobian(x -> F(x, y, p, t), yp)
    jac_Fp = ForwardDiff.jacobian(x -> F(yp, y, x, t), p)

    for i in 1:n_params
        res[n_terms*i + 1:n_terms*(i+1)] = jac_Fx*u[n_terms*i + 1:n_terms*(i+1)] +
    jac_Fxp*du[n_terms*i + 1:n_terms*(i+1)] + jac_Fp[1:n_terms, i]
    end
end
```

```
# ********************************************************************************
# Initial values and initialization of the DAE problem and sensitivity analysis:
function g_inital!(fvec, CC)
    t0 = 0
    fvec[1] = F(zeros(3), (CA_0, CB_0, CC[1]), θ, t0)[3]
end

# 1) at t = 0 -> specify CA_0 and CB_0 and then GUESS for CC_0
# 2) To find initial values:
# 2.1) Solve g(y,t) = 0 with NLSolve (no derivative terms) - find y(0)
CC_initial = nlsolve(g_inital!, [0.1], method = :newton).zero[1]
# 2.2) Explicitly solve the system f(y,t) - y' = 0 for others to find y'(0)
dCA_0 = -k1*CA_0
dCB_0 = k1*CA_0 - k2*CB_0 + k3*CC_0
cCC_0 = CA_0 + CB_0 + CC_initial - (CA_0 + CB_0 + CC_0)

# 3) Initial values for the sensitivity parameters:
du0 = [dCA_0, dCB_0, cCC_0]; u0 = [CA_0, CB_0, CC_initial]; vt0 = 0;

# Declare n_eqs and n_params for use in general function:
n_terms = size(u0, 1); n_params = size(θ, 1);

# Define initial jacobian matrices:
jac_Fx_0 = ForwardDiff.jacobian(x -> F(du0, x, θ, t0), u0);
jac_Fxp_0 = ForwardDiff.jacobian(x -> F(x, u0, θ, t0), du0);
jac_Fp_0 = ForwardDiff.jacobian(x -> F(du0, u0, x, t0), θ);

# 3.1) Solve the initial sensitivity values for the last diff equation (not 0)
sens_0 = zeros(n_params*n_terms);
for i in 1:n_params
    sens_0[n_terms*i] = -jac_Fx_0[3,3]\jac_Fp_0[3, i];
end

# The initial value for (dC_A/dC_A0) has to be 1
sens_0[1] = 1; sens_0[3] = 0

# 3.2) Compute the values for the initial sensitivity derivatives:
# Solving the system of equations: ∂f/∂y * s_p + ∂f/∂y' * s_p' + ∂g/∂p = 0
# in matrix form -> dsens_0 = -jac_Fxp\(jac_Fp + jac_Fx*sens_0)
dsens_0 = zeros(n_params*n_terms);
for i in 1:n_params
    dsens_0[n_terms*(i-1) + 1:n_terms*i - 1] = (-jac_Fxp_0\(jac_Fp_0[1:n_terms, i] +
    jac_Fx_0*sens_0[n_terms*(i-1) + 1:n_terms*i]))[1:2];
end

# Initial values for extended DAE and solver time span:
y0 = vcat(u0, sens_0); yp0 = vcat(du0, dsens_0); tspan = (0.0, 0.02);

# Solving the extended DAE using ModelingToolkit functions
DAE_fun = DAEFunction(F_sens)
prob = DAEProblem(DAE_fun, yp0, y0, tspan, θ, differential_vars = [true, true, false,
     true, true, false, true, true, false, true, true, false, true, true, false])
@time sol = solve(prob, IDA())

# Plot results ...
```

The results from this self built approach to integrate sensitivity equations give identical

results to the ones simulated with the Julia libraries. The resulting code is much more complex because initial values for the concentrations and parametric sensitivity have to be exactly determined before solving the DAE. As can be seen, this code has the original DAE function F which depends on the 4 states required and then has the extended DAE system of equations given by F_sens which includes the sensitivity set of equations for every parameter of interest. Thus, converting the system from a three equation DAE to a fifteen equation DAE $n_x + n_\theta * n_x = 3 + 4 * 3 = 15$. The computational time required to solve this extended system is **6588** μ**s** which is still coherent compared to the more efficient Julia alternatives as it is in a similar order of magnitude but much less optimized.

The parametric sensitivity results for this simple chemical reaction in a short time ($20ms$) where given in the case study. Results for a longer reaction time ($10s$) are shown below:

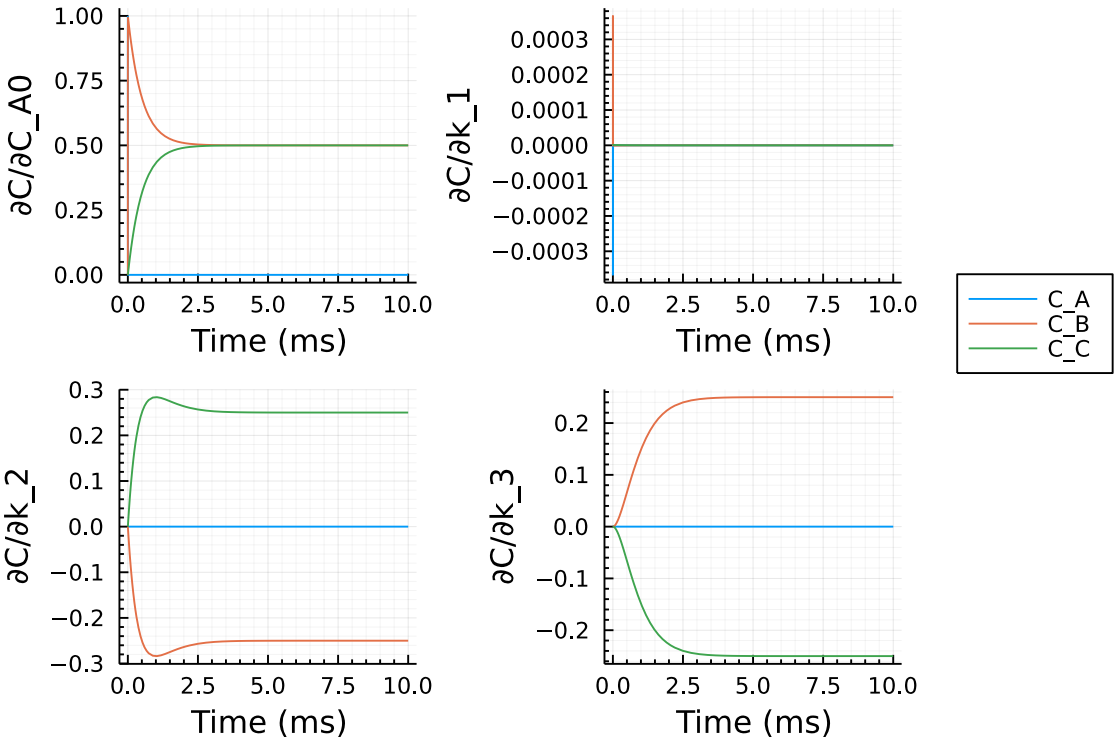

FIGURE A.1: Complete evolution ($10s$) of the parametric sensitivity of each substance in the chemical reaction example against the desired parameters in $\theta = [C_{A,0}, k_1, k_2, k_3]$.

Appendix B

Implementation of SA for PETLION

Following the notation from the Mathematical Background and Methodology section, the steps used to model any optimal charging protocol with uncertainties in PETLION is:

.....

1. Run any desired operational mode $CC, CV, CPo, CT, ...$ with nominal parameters
2. Choose the uncertain parameters that affect this model and their normal distributions
3. Choose a success range α for the percentage of accuracy desired in the output dispersion
4. For every or all **states of interest** of the simulated model:
   (a) Compute the state's parametric sensitivities $L(t)$ using Central Finite Differences
   (b) Compute the standard deviation using the definition of σ_ψ in (3.3)
   (c) Use the standard deviations equivalent to α to determine new state limits
5. Check new boundaries (time constraints, maximum values, etc.) for the deviated model. Determine new back-off from results of interest against original deterministic results.

.....

The numerical implementation used for these steps is described on this section. First of all, to run the desired operational mode with nominal parameters we must first set the parameters and constraints of the simulation using `p::PETLION.model`. Then the simulation is run using `simulate(p, I = 4, SOC = 0, V_max = 4.1, ...)` for any chosen operational modes and maximum and minimum boundaries. The uncertain parameters' names are defined in the vector `parameter_names = (:T_amb, :D_sn, :D_sp)` and their nominal values in `parameter_values`. These vectors will be passed to the following functions for uncertainty quantification. The desired success range is $\alpha = 99.7\%$ in these functions which is the probability range covered

when considering $\pm 3std$ from the nominal state. The functions designed to implement the steps (a), (b), and (c) for any state are given below.

B.1 Parametric sensitivities L(t) using CFD

The function to compute sensitivities for a single state of interest is built using the algorithm defined above for Central Finite Differences (coefficients $= [-1/2,\ 1/2]$, indices $= [-1,\ 1]$) with step size $\Delta = 10^{-3}$. The function is generalized to compute the sensitivity for multiple output states ψ depending on the requirements.

Listing B.1: 'Function to compute sensitivity vector of a single state for any set of uncertain parameters'

```
function compute_sensitivity(p_threads, parameter_names, parameter_values, time_points, output,
    simulate_fun::Function = my_simulate, x...; kw...)
    # Chosen step for CFD and its indexes and coefficients for second-order accuracy
    step_size = 10 (-3); indexes = [-1 1]; coefficients = [-1/2 1/2]
    sol_dummy = simulate_fun(parameter_names, parameter_values, p_threads[1], time_points,
                             x...; kw...)
    sol_sample = similar(getfield(sol_dummy, Symbol(output)))
    sol_sens = [sol_sample for i in 1:length(parameter_names)*length(indexes)]
    I = CartesianIndices((1:length(parameter_names), 1:length(indexes)))
    Threads.@threads for iter in 1:length(I)
        Δ_parameter = parameter_values[I[iter][1]]*step_size
        p_dummies = copy(parameter_values)
        p_dummies[I[iter][1]] = p_dummies[I[iter][1]] + indexes[I[iter][2]]*Δ_parameter
        sol_dummy = simulate_fun(parameter_names, p_dummies, p_threads[Threads.threadid()], time_
    points, x...; kw...)
        sol_sens[iter] = coefficients[I[iter][2]].*getfield(sol_dummy, Symbol(output))
    end
    sol_sens_dict = Dict{Symbol,Any}()
    for i in 1:length(parameter_names)
        sol_sens_dict[parameter_names[i]] = (sol_sens[i] + sol_sens[length(parameter_names)+i])./(
    parameter_values[i]*step_size)
    end
    return sol_sens_dict
end
```

B.2 Standard deviation of the output state

Once the sensitivity vector $L(t)$ is known for the state of interest it can be used to compute the final deviation. As explained on the mathematical background, the standard deviation of the state depends on the vector $L(t)$ and the matrix of variances of the uncertain parameters V_θ. The matrix of standard deviations is defined using $\sigma_i = 0.5\% \cdot i$ for any parameter used except for the temperature which has $\sigma_{T_{amb}} = 1\,°\text{C}$. The matrix could be specifically defined for any desired dispersion. The sensitivity vector for multiple parameters at every time point is used separately to find $\sigma_\psi(t)$ which is saved as `std_output`.

Listing B.2: 'Function to compute the standard deviation of any state given the sensitivity results from the previous function'

```
using LinearAlgebra
```

```
## 1st sensitivity analysis attempt for 1 vector of sensitivities:
function compute_std(parameter_names, parameter_values, sensitivity_dict)
    # Create covariance matrix according to used uncertain parameter
    θ_1_names = parameter_names; nominal_θ = parameter_values;
    std_vector = [(value*0.005)^2 for value in parameter_values] # Variance: σ*σ diagonal matrix
    temp_index = findall(parameter_names .== :T_amb)[1] # std is 1C in temperature
    std_vector[temp_index] = 1.0
    V_θ_1 = zeros(length(parameter_names), length(parameter_names))
    V_θ_1[diagind(V_θ_1)] .= std_vector

    # Compute std with covariance matrix and sensitivity results for every time point
    std_output = float.(sensitivity_dict[parameter_names[1]])
    for time_i in 1:length(std_output)
        if length(std_output[1]) == 1
            L = [sensitivity_dict[name][time_i] for name in parameter_names]
            V_ψ = L'*V_θ_1*L
            std_output[time_i] = sqrt(abs(V_ψ))
        else
            for space_i in 1:length(std_output[1])
                L = [sensitivity_dict[name][time_i][space_i] for name in parameter_names]
                V_ψ = L'*V_θ_1*L
                std_output[time_i][space_i] = sqrt(abs(V_ψ))
            end
        end
    end
    return std_output
end
```

B.3 Determine new state limits

Finally, once the standard deviation of the desired state is known we can use it to study the probabilistic distribution of that variable in time with any desired accuracy α. In this case, since the probability significance chosen is always 99.7% the number of standard deviations from the mean required are 3. To analyze the upper and lower bounds of this deviation in the chosen state we can just plot the nominal curve and ribbon edges using the results from `std_output` found before.

B.4 Final optimal approach and implementation

Lastly, as explained on the results section, this methodology is optimized to compute the sensitivities of all the output states simultaneously. Since the deviated model has to be run 2 times for CFD, the resulting states can be computed directly from that same model without having to call the `compute_sensitivity` function multiple times. To do so, the previous functions are fused and applied to all the states `getfield(model.results[end].opts.var_keep,state) == true` which are all the states significant to any model outputs in PETLION.

Bibliography

[1] Berliner, Marc D et al. "Methods—PETLION: Open-Source Software for Millisecond-Scale Porous Electrode Theory-Based Lithium-Ion Battery Simulations". In: *Journal of The Electrochemical Society* 168.9 (2021), p. 090504.

[2] Berliner, Marc D et al. "Nonlinear Identifiability Analysis of the Porous Electrode Theory Model of Lithium-Ion Batteries". In: *Journal of The Electrochemical Society* 168.9 (2021).

[3] Berliner, Marc D. et al. "Fast Charging of Lithium-ion Batteries by Mathematical Reformulation as Mixed Continuous-Discrete Simulation". In: *2022 American Control Conference (ACC)*. accepted. IEEE. 2022.

[4] Caracotsios, Makis and Stewart, Warren E. "Sensitivity analysis of initial value problems with mixed odes and algebraic equations". In: *Computers and Chemical Engineering* 9.4 (1985), pp. 359–365.

[5] Galan, S., Feehery, William, and Barton, Paul. "Parametric sensitivity functions for hybrid discrete/continuous systems". In: *Applied Numerical Mathematics* 31 (1999), pp. 17–47.

[6] Grey, Clare P. and Hall, David S. "Prospects for lithium-ion batteries and beyond—a 2030 vision". In: *Nature Communications* 11 (2020).

[7] Hannan, M.A. et al. "Review of energy storage systems for electric vehicle applications: Issues and challenges". In: *Renewable and Sustainable Energy Reviews* 69 (2017), pp. 771–789.

[8] Hindmarsh, Alan C et al. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3 (2005), pp. 363–396.

[9] Hindmarsh, Alan C. et al. *User Documentation for IDA*. v6.1.1. 2021.

[10] Liu, Kai et al. "Materials for lithium-ion battery safety". In: *Science Advances* 4.6 (2018).

[11] Ma, D. L., Chung, S. H., and Braatz, R. D. "Worst-case performance analysis of optimal batch control trajectories". In: (1999), pp. 3256–3261.

[12] Nagy, Zoltan and Braatz, Richard. "Distributional uncertainty analysis using power series and polynomial chaos expansions". In: *Journal of Process Control* 17 (Mar. 2007), pp. 229–240.

[13] Newman, J and Tiedmann, W. "Porous-Electrode Theory with Battery Applications". In: *AlChE Journal* 21.1 (1975), pp. 25–41.

[14] Rackauckas, Christopher and Nie, Qing. "Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of Open Research Software* 5.1 (2017).

[15] Revels, Jarrett, Lubin, Miles, and Papamarkou, Theodore. *Forward-Mode Automatic Differentiation in Julia*. 2016.

[16] Serban, Radu et al. *User Documentation for IDAS*. v5.1.1. 2021.

[17] Tamilselvi, S. et al. "A Review on Battery Modelling Techniques". In: *Sustainability* 13.18 (2021).

[18] Torchio, M et al. "LIONSIMBA: A Matlab framework based on a finite volume model suitable for Li-ion battery design, simulation, and control". In: *Journal of The Electrochemical Society* 163.7 (2016), A1192–A1205.

[19] Van den Bossche, Peter et al. "SUBAT: An assessment of sustainable battery technology". In: *Journal of Power Sources* 162.2 (2006), pp. 913–919.

[20] Various authors. *The Julia Programming Language*. 2021. URL: https://julialang.org/ (visited on 05/10/2022).