UNIVERSITÄT
DES
SAARLANDES

Mastering Satellite Operation: On Model-based and Data-driven Optimal Battery-Aware Scheduling

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften (Dr.-Ing.) der Fakultät für Mathematik und Informatik der Universität des Saarlandes

vorgelegt von
GILLES NIES

Saarbrücken, 2021

Dean of the faculty	*Prof. Dr. Jürgen Steimle*
Day of colloquium	*22.06.2022*
Chair of the committee	*Prof. Dr. Verena Wolf*
Reviewers	*Prof. Dr. Holger Hermanns* *Prof. Dr. Boudewijn Haverkort* *Asst. Prof. Dr. Morteza Lahijanian*
Academic assistant	*Dr. Alexander Gress*

Abstract

Rechargeable batteries as a power source have become omnipresent. Especially the lithium-ion variant powers virtually every contemporary portable device, it constitutes the central energy source in the domains of e-mobility, autonomous drones and robots, most earth-orbiting spacecraft like satellites and many more.

In this work, we take the perspective of the kinetic battery model, an intuitive analytic model, that keeps track of a battery's charge as it is strained with a sequence of loads over time. We provide the mathematical foundation of the battery model, extend it with capacity limits and realistic charging behavior, as well as uncertainty in its initial state and the load it is strained with. In addition, we show how to estimate the non-measurable state of a kinetic battery via measurable quantities, like voltage and current, using Kalman filters.

We derive efficient energy budget analysis algorithms in the form of discretization and analytical bisection schemes, deduce their efficiency, empirically analyze their performance and identify their individual strengths and weaknesses.

We show how our techniques can be used during design time and in-flight, in the context of two nanosatellite missions GOMX–1 and GOMX–3. For the former, we derive probabilistic workload models to analyze the mission's yearly energy budget, while for the latter, we bridge the gap from analysis to synthesis by leveraging model checking techniques of the formal methods community in order to produce battery-aware optimal schedules to be executed by the satellite. We demonstrate the adequacy of our scheduling approach via three in-orbit test runs on GOMX–3, and show a close correspondence between predicted workload and battery state compared to logged telemetry data such as voltage and current. Finally, in an attempt to continuously perpetuate the schedule synthesis, we extend the scheduling workflow with Kalman filters in order to additionally harness these in-flight telemetry data so as to regularly adjust the battery state estimation.

Zusammenfassung

Wiederaufladbare Akkumulatoren als Energiequelle sind mittlerweile omnipräsent. Insbesondere die Lithium-Ionen Variante versorgt fast jedes moderne, portable Gerät mit Energie. Sie ist in einer Vielzahl von Bereichen integral, insbesondere in den Bereichen der E-Mobilität, der autonomen Dronen und Roboter, sowie der Erdsatelliten.

Das Augenmerk dieser Arbeit richtet sich auf das kinetische Batteriemodell. Dieses intuitive, analytische Modell, beschreibt die zeitliche Entwicklung des Ladezustands eines Akkumulators, der einer Sequenz elektronischer Lasten unterzogen wird. Das mathematische Fundament des Modells wird erweitert um Kapazitätsgrenzen und realistisches Ladeverhalten, sowie um den Umgang mit Unsicherheit im Kontext des initialen Ladezustands und der anliegenden Last. Darüber hinaus, schlagen wir ein auf dem Kalman-Filter basierendes Schätzverfahren vor, welches den unmessbaren Ladezustand anhand messbarer Quantitäten, wie Spannung und Stromfluss, approximiert.

Wir leiten effiziente Algorithmen her, die der Bestimmung von Energiebedarf und Energiebilanz eines batteriebetriebenen Geräts dienen. Diese beruhen auf Diskretisierungs- sowie auf analytischen Intervallhalbierungsverfahren. Es folgt eine Analyse der theoretischen und praktischen Effizienz, sowie die Identifizierung etwaiger individueller Stärken und Schwächen der Paradigmen.

Anhand zweier konkreter Fallstudien zeigen wir, wie unsere Verfahren die Planung sowie den laufenden Betrieb einer Nanosatellitenmission erleichtern und verbessern. Für die GOMX–1 Mission wird ein probabilistisches Arbeitslastmodell eines Nanosatelliten aus Flugtelemetriedaten inferiert, anhand dessen die jährliche Energiebilanz des Satelliten hergeleitet werden kann. Im Kontext der zweiten Mission, GOMX–3, entwerfen wir ein Verfahren zur Synthese energieoptimaler Ablaufpläne, die in der Praxis durch den Satelliten ausgeführt werden. Zu diesem Zeck werden Methoden aus dem Bereich der Modellprüfung herangezogen und mit den Analyseverfahren des Batteriemodells verzahnt. Anhand dreier konkreter Testläufe, lässt sich eine starke Korrespondenz zwischen Telemetriedaten und vorhergesagter Quantitäten erkennen, welche die Angemessenheit unserer Syntheseverfahren untermauert. Zur kontinuierlichen Fortsetzung und Verlängerung der synthetisierten Pläne über die gesamte Missionsdauer, wird schließlich das Verfahren durch einen weiteren Zwischenschritt ergänzt. Dieser dient durch das Verwenden von Kalman-Filtern dazu, den vorhergesagten Ladezustand des Akkumulators anhand besagter Telemetriedaten regelmäßig anzupassen, um eventuelle, durch das Batteriemodell hervorgerufene Abweichungstendenzen auszugleichen.

Contents

CHAPTER

1 Introduction

Batteries as a power source have become more or less omnipresent. Especially Lithium-ion battery technology is built into almost every contemporary portable device like notebooks, smartphones, smart watches, wireless headphones and its use is only increasing, i.e. electric cars and bikes, unmanned flying drones or multicopter aircraft, cordless vacuum cleaners, autonomous vacuuming robots, video gaming handhelds and even entire video gaming consoles. The technology has become an integral part of more and more homes and provides stability in our otherwise rather volatile power grids by interposing large collections of battery cells, serving as energy buffers. For most spacecraft that orbit any astronomical body in our solar system, such batteries serve as rechargeable power source as such objects are bound to enter eclipse at some point, and thus have no direct power supply. It is almost safe to say, that the use of batteries will grow in the future rather than decline, ever-increasing the need for estimating, analyzing and planning with the remaining battery charge in all kinds of devices and situations.

In the space domain, historically a very conservative domain, due to the immense cost of space missions, a revolution has been looming. With the standard of nanosatellites and CubeSats in 1999, space access was rendered financially feasible for universities, spin-offs thereof and small start-up enterprises. Proportionally to the decreasing cost of launching a satellite into space, also the inhibitions of using modern, more experimental methods decreased, making the space domain a scientifically lucrative playground for state-of-the-art automation, analysis and planning techniques.

In this thesis, we take the perspective of analytic battery models as well as the rigorous mathematical analysis thereof, and cross-fertilize this domain with the modelling formalisms and verification techniques of the formal methods community, in order to develop automatic, efficient and precise analysis and scheduling procedures particularly tailored to nanosatellite missions.

Specifically, we examine the kinetic battery model, a simple, intuitive, yet precise analytic battery model and extend it appropriately to fit the space domain. These extensions include the incorporation of capacity limits while charging, uncertainties with regards to the initial battery state as well as the load a battery is strained with and the consolidation of the formal battery model with real-life measurements of said battery, in order to correct for any bias or drift the model may

exhibit. We derive the theoretical foundations of efficient but accurate analysis algorithms to estimate depletions risks and empirically as well as theoretically investigate their computational efficiency and accuracy.

With sophisticated energy budget analysis techniques in place, we proceed to the formal methods world, so as to first leverage familiar modelling formalisms to capture the energy-relevant aspects of a satellite mission over time while in orbit. Later we use the associated model checking techniques to synthesize optimal plans for the mission, while strictly adhering to side constraints, mostly but not exclusively about the battery state. In general, most battery models are inherently nonlinear, and hence cannot be directly expressed by modelling formalisms that come with efficient analysis techniques. Hence, we resort to a simple linear battery model in the schedule synthesis part, that is completed to a feedback loop via a validation step involving the tailored analysis algorithms of the more realistic kinetic battery model. This validation step is able to reject and exclude a synthesized schedule, that was optimal with respect to the linear model, yet turned out to be too energetically expensive with respect to the kinetic battery model. In case a schedule is rejected, re-synthesis is triggered.

In orbit, where direct contact to a spacecraft, and hence also direct influence on its behavior via its operators is rather sparse. Hence, in order to optimize for productivity of the satellite, an operator must be sufficiently confident in the accuracy of the involved models. However, no model is without flaws, and may be biased or expose drifts with respect to reality. Detection of such drifts may only occur as late as with the next spacecraft contact, and its associated telemetry data downlink. A reaction to counteract a drift may be delayed by even more than one orbit, as a satellite may have already left the ground station's field of vision. The scheduling procedure is thus ameliorated with a battery state estimation step that consolidates model-based data with the most recent telemetry data as soon as they arrive. Consequently, from the latest battery state estimate, a new optimal plan is synthesized, and made available to the spacecraft at the earliest time. This approach effectively leads to a continuous perpetual generation of schedules.

1.1 Contribution

The contribution of this thesis is essentially threefold:

- extensions to the kinetic battery model,
- three algorithms that effectively implement energy budget analyses in terms of the depletion risk of a certain system given a load model,
- practical real-world applications that showcase how to use the extended battery model in conjuction with formal methods to realize battery-aware scheduling of nanosatellites, thereby showcasing the practical applicability of the contributions above.

Extensions To The Battery Model:

- We extended the kinetic battery model with *capacity limits*, and thereby with realistic charging behavior. It takes the form of a capacity threshold beyond which a battery may not be charged gain any more available charge. We propose a separate set of differential equations that characterizes such a

so-called saturated battery. This simple concept brings a perhaps surprising amount of challenges with it. Among them, finding the exact time point of saturation, a problem which turns out to be non-elementary. We therefore propose an algorithm to bound this time point from above and below, and realize approximative dynamics of a kinetic battery with capacity limits using this algorithm.

- Orthogonally, we extend a kinetic battery with *uncertainty in the initial initial battery state* as well as *load noise*. This concept makes the introduction of so-called state of charge distributions instead of single battery states necessary. Based on these distributions, we rigorously derive analytic expressions for approximations of successor distributions given a noisy task and an initial battery state distribution.

- We extend the load model from piecewise linear loads to markovian processes we refer to as *Markov Task Processes*. These processes effectively extend sequences of constant loads with jump probabilities, i.e. multiple successors that may be selected at random. We show how to capture the entire behavior of such a task process up to a certain time horizon, as well as its effect on the battery.

Algorithms:

- We introduce *two discretization approaches* that implement a gridding of the battery state space, first as a static scheme, then as an adaptive scheme that only focuses on a relevant neighborhood of the actual support of the battery state distribution. Both approaches provide a remedy to the computationally impractical analytic expressions derived before.

- In an attempt to avoid discretization entirely, we introduce the so-called *percentile propagation* algorithm. The approach is based on the fact that one can derive bounds on the overall depletion risk of an entire battery state distribution, by only investigating a certain percentile battery state of it. Iterating this step in a manner similar to a bisection scheme, one can derive arbitrarily close bounds on the true depletion risk, with the precision being configurable a priori. The applicability of the scheme is tied to certain preconditions, however, which we identify.

Practical Applications:

- We showcase the value of the above contributions during design time of a nanosatellite mission on the case of the GOMX–1 satellite, for which we performed an *energy budget analysis* to conclude that the onboard batteries where over-dimensioned by the factor of 8.

- We designed and implemented a *fully automatic battery-aware scheduling pipeline* for the GOMX–3 satellite mission, and were the first to showcase the applicability of the synthesized optimal schedules via three different in-flight test runs on GOMX–3. These successfull test runs demonstrate the value of formal methods in conjunction with formal energy storage models for in-flight operation of satellite missions.

- We derived the groundwork for a prototype of a *receding-horizon version of the battery-aware scheduling pipeline*, an extension which is now being used successfully with the GOMX–4 mission [38].

1.2 Chapter Origins

The results of this thesis are partially based on the following published results:

- Holger Hermanns, Jan Krčál, Gilles Nies, Recharging Probably Keeps Batteries Alive, in Cyber Physical Systems. Design, Modeling, and Evaluation, Lecture Notes in Computer Science, Volume 9361, pages 83–98, 2015.
- Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies, Marvin Stenger: Battery-Aware Scheduling in Low Orbit: The GOMX–3 Case, in Formal Methods, Lecture Notes in Computer Science, vol 9995, pages 559–576, 2016. **This work received a best paper award.**
- Holger Hermanns, Jan Krčál, Gilles Nies, How Is Your Satellite Doing? Battery Kinetics with Recharging and Uncertainty, in Special Issue On Quantitative Evaluation Of Systems, Leibniz Transactions on Embedded Systems, Vol. 4 No. 1, pages 04:1–04:28 February 2017.
- Gilles Nies, Marvin Stenger, Jan Krčál, Holger Hermanns, Morten Bisgaard, David Gerhardt, Boudewijn Haverkort, Marijn Jongerden, Kim G. Larsen, Erik R. Wognsen, Mastering Operational Limitations Of LEO Satellites: The GOMX–3 Approach, in Acta Astronautica, Volume 151, pages 726–735, 2018.
- Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies and Marvin Stenger, Battery-Aware Scheduling In Low Orbit: The GOMX–3 Case, In Formal Aspects of Computing, Issue 31, pages 261–285, 2019.
- Holger Hermanns and Gilles Nies, Quantification of Battery Depletion Risk Made Efficient, in NASA Formal Methods, Lecture Notes in Computer Science, Volume 13260, pages 156–174, 2022

1.3 Outline

Chapter 2 provides a short overview of the family of analytical battery models, as well as a quick introduction of Timed Automata, its Priced extensions and the prevalent tool for model checking networks thereof, UPPAAL CORA, and features a quick dive into control theory with an introduction of Kalman filters.

In **Chapter 3** the theory and the extensions of the kinetic battery model are developed in detail, while in **Chapter 4** corresponding analysis algorithms are derived and their efficiency is investigated.

We proceed by showcasing the interplay and application of battery models and model checking in **Chapter 5**, which features real-world case studies of CubeSats GOMX–1 and GOMX–3. For the former, we purely take an energy budget analysis perspective, while for the latter we bridge the gap to battery-aware synthesis of optimal plans for short time horizons, and the continuous perpetuation thereof culminating in a receding-horizon scheduling approach.

Chapter 6 concludes the thesis.

CHAPTER

2 Preliminaries

2.1 Battery Models

Batteries in-the-wild exhibit two non-linear effects widely considered to be the most important ones to capture: the *rate capacity effect* and the *recovery effect*. In the sequel we introduce the *kinetic battery model* KiBaM as the simplest model capturing these effects, and place it in the context of other candidates to model a battery. A thorough analysis of these battery models and their respective characteristics is found in the literature [20].

The Linear Model. Also called the *ideal battery*, this model views a battery as one well of capacity cap that is decreased proportionally to a load ℓ that is imposed on the battery. Thus, the lifetime of a full battery under load ℓ can naturally be expressed by

$$\frac{\mathsf{cap}}{\ell}.$$

While easy to handle and to extend, the linear battery model neither captures the recovery of batteries nor the rate-capacity effect.

Peukert Model. An extension of the ideal battery is provided by *Peukert's law*. Here, parameters a and b characterize the lifetime of a full battery under load ℓ as a/ℓ^b. For $a = \mathsf{cap}$ and $b = 1$, this corresponds to an ideal battery, although parameters fitted through experiments generally result in a being a bit smaller than cap and b being slightly larger than 1. Peukert's law captures the rate capacity effect, but neglects the recovery effect.

The Electrochemical Model. Together with its accompanying simulation tool DUALFOIL [8], the highly complex and parameterizable electro-chemical battery model is, in its own right, widely considered as the reference "reality" to check the faithfulness and accuracy of other models. It is comprised of 6 coupled, non-linear differential equations and it requires over 50 battery-type specific parameters in order to be faithfully run.

The Diffusion Model. The *diffusion model* describes the ion concentration along the width of a battery as a continuum. A full battery exhibits equal concentration along the battery, while a discharge causes a decrease of the concentration near the discharging electrode. This, in turn, causes a gradient that causes the ions to diffuse towards the electrode. Thus, during periods of rest the ion concentration tends to equilibrate along the width of a battery, inducing a recovery. During periods of high discharge, the diffusion cannot keep up causing premature depletion; the rate capacity effect. The model allows for the derivation of analytical expressions of the battery lifetime as well and exhibits a very high degree of precision against the electro-chemical model.

The Kinetic Battery Model. The *kinetic battery model* (KiBaM) can be viewed as a discretized diffusion model by dividing the stored charge into two parts, the *available* charge and the *bound* charge and can actually be proven to be a first-order approximation of the diffusion model. When the battery is strained only the available charge is consumed instantly, while the bound charge is slowly converted to available charge by diffusion. This diffusion between available and bound charge can take place in either direction depending on the amount of both types of energy stored in the battery. Both non-linear effects are captured for the exact same reason as for the diffusion model: the relatively slow conversion of bound charge into available charge or vice versa. Due to its simplicity, intuitiveness and accuracy relative to the more complex diffusion model and, by extension, also relative to the DUALFOIL electro-chemical model simulator, we focus on the kinetic battery model in this thesis. We provide a thorough and rigorous analysis of the KiBaM in Chapter 3.

2.2 Kalman Filter

Kalman filtering is an algorithm that, given possibly noisy time series of measurements, estimates the internal state of a dynamical system, related to these measurements. Essentially, a Kalman filter forms a kind of feedback loop in two steps, a *prediction* and a *correction* step. In the prediction step, the Kalman filter projects the $k-1$-st state estimate ahead using the dynamical model underlying it, into a preliminary k-th state estimate. In the correction step, it incorporates the k-th measured datapoint as feedback into its preliminary prediction and corrects it accordingly, to get the definitive k-th state estimate.

Formally, the Kalman filter addresses the general problem of estimating a state $x \in \mathbb{R}^n$ of a discrete time, possibly controlled process given by the difference equation:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{2.1}$$

with measurements $y \in \mathbb{R}^m$ given by

$$y_k = H_k x_k + v_k,$$

and where

F_k is the (time-discrete) $n \times n$ *state transition matrix* of the k-th step induced by the underlying dynamical process.

B_k is the $n \times l$ *control matrix* of the k-th step, relating (optional) control input to state variables.

u_k is the k-th *control variable*, with $u_k \in \mathbb{R}^l$.

w_k represents *process noise* in the k-th step, where $w_k \sim \mathcal{N}(0, Q_k)$. Q_k is the *process noise covariance* in the k-th step.

H_k is the $m \times n$ *measurement matrix* of the k-th step, relating state with measurements.

v_k represents *measurement noise* in the k-th step, where $v_k \sim \mathcal{N}(0, R_k)$. R_k is the *measurement noise covariance* in the k-th step.

The Kalman filter, being a recursive scheme, needs to be initialized with an initial estimate of the internal state x_0 as well as with an initial state variance matrix P_0 reflecting the confidence in the initial state. Intuitively speaking, the higher this initial state variance is chosen, the more a Kalman filter will trust the measurements in the beginning stages of the estimation. Process and measurement noise in each step are assumed to be pairwise mutually independent. From the difference equation it becomes apparent that Kalman filters do not need the entire history in order to start estimating state. It rather only needs the systems previous state and the current measurement, making it memory efficient and fast (and thus even suitable for hard real time applications).

The computational details of how a Kalman filter recursively estimates state are well documented and understood [22] and beyond the scope of this thesis. It should however be mentioned, that the Kalman filter is the *optimal linear filter* if the model matches the real system exactly and if the noise is uncorrelated white noise with known covariances.

2.3 (Priced) Timed Automata

The model of *Timed Automata* (TA) [2] has been established as a standard modelling formalism for real time systems. Timed Automata extend finite state machines with non-negative real-valued variables called *clocks* in order to capture timing constraints. Thus, a timed automaton is an annotated directed graph over a set of clocks C with vertex set L (called *locations*) and edge set E. Edges and locations are decorated with conjunctions of clock constraints of the form $c \bowtie k$ where $c \in C$, $k \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. For edges such constraints are called *guards*, for locations they are called *invariants*. Edges are additionally decorated with *reset sets* of clocks. Intuitively, taking an edge causes an instantaneous change of location and a reset to 0 for each clock in the reset set. However an edge may only be taken if its guard and the target location's invariant evaluate to true. If this is not the case the current location remains active, if it's invariant permits, and clocks increase continuously with the same rate, thus modelling the passing of time.

In order to reason about resources, TAs are enriched with non-negative integer *costs* and non-negative *cost rates* in the form of annotations for edges and locations, respectively [4]. The result are *Priced Timed Automata* (PTA). The intuition is that cost accumulates continuously proportional to the sojourn time of locations and increases discretely upon taking an edge as specified by the respective annotations.

Definition 1 — *Priced Timed Automata.* Let C be a set of clocks and $\mathbb{B}(C)$ be the set of all clock constraints as described above. A priced timed automaton is a tuple $(L, E, \ell_0, \mathsf{inv}, \mathsf{price})$ where L is a set of locations, $E \subseteq L \times \mathbb{B}(C) \times 2^C \times L$ is a set of edges, ℓ_0 is the initial location, $\mathsf{inv} : L \to \mathbb{B}(C)$ assigns invariants to locations, and $\mathsf{price} : L \cup E \to \mathbb{N}$ assigns costs and cost rates to locations and edges, respectively.

We omit the formal semantics of PTA, and instead refer to the literature for a complete development [4].

A common problem to consider in the context of PTA is that of computing the minimum cost to reach a certain set of target states in a given PTA. This so-called *cost-optimal reachability analysis* (CORA) receives dedicated attention in the literature [3, 23] and is well-known to the community. The CORA is implemented in a number of tools, most prominently UPPAAL CORA [6]. As input UPPAAL CORA accepts networks of PTAs extended by discrete variables, and thus allows for modular formalisation of individual components. The set of goal states is characterised by queries, i.e. temporal formulae over the variables declared in the network of PTAs. These usually fall into the category of so-called time-bounded reachability queries. The tool's query language is a subset of *Timed Computation Tree Logic* (TCTL) [24] and thus consists of path and state formulae. A state formula is a simple expression over the variables of the system and is evaluated using a state's valuation of said variables. Path formulae come in different flavours, yet for this work only reachability path formulae are relevant. Reachability properties are of the form $\exists \Diamond \varphi$ and ask whether there *exists* ($\exists$) a path that *eventually* ($\Diamond$) reaches a state which satisfies the state formula φ. The tool has a predefined variable `time` capturing the global time of a whole system, so there is no need for a user to explicitly manage a global clock.

CHAPTER

3 The Kinetic Battery Model

In this chapter we rigorously introduce the kinetic battery model (KiBaM), as found in the literature. We continue with the concept of depletion (Section 3.2) and proceed by augmenting the KiBaM with the highly non-symmetric concept of capacity limits and battery saturation in Section 3.3. Unfortunately, this concept eventually involves transcendental numbers when trying to determine the battery saturation time point, which makes its exact determination computationally impossible. We instead introduce approximations of saturation time points (Section 3.4). In Section 3.5 the KiBaM is further extended with uncertainties in the initial state of charge (SoC) and the loads the battery is strained with. This leads to a stochastic interpretation of the KiBaM with capacity limits, for which we show how to efficiently compute successor battery state distributions. With the base operations on the KiBaM in place, we proceed in Section 3.6 with the introduction of a Markovian load process model that includes probabilistic branching, arbitrary noisy loads but deterministic timing aspects, and show how to compute the resulting battery state distribution up to an arbitrary time horizon. To finalize the chapter, Section 3.7 outlines how measurements of voltage and current can be combined with the KiBaM to avoid accumulating inaccuracies over time in battery state estimates.

3.1 The Kinetic Battery Model

The *Kinetic Battery Model* (KiBaM) is an energy storage model that models the state of charge (SoC) of a battery by splitting it into two disjoint portions, namely

- the available charge $\mathrm{A}(t)$, the portion of stored charge that is directly available to be consumed or replenished.
- the bound charge $\mathrm{B}(t)$, the portion of stored charge that is chemically bound inside the battery, and is not considered to be immediately available.

These quantities can be considered unitless and abstract for the moment. The battery is strained by a *load* $\ell(t)$ that represents charging and discharging if $\ell(t) < 0$ and $\ell(t) > 0$, respectively. If there is no load (i.e. $\ell = 0$) we speak of a *resting* period.

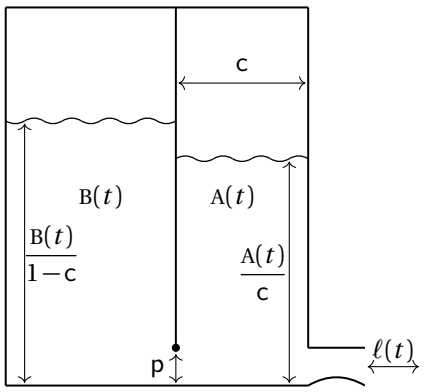


Figure 3.1: The two-wells depiction of the KiBaM

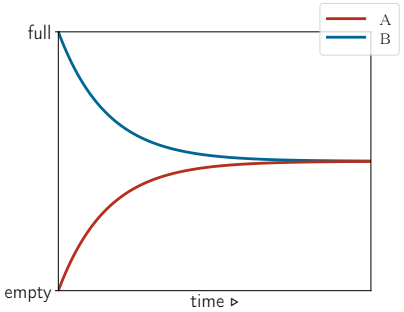
The principle behind the KiBaM is that bound charge is converted into available charge over time (or vice-versa) via diffusion, which is why the KiBaM is often depicted as two interconnected wells holding fluid (see Figure 3.1).

The model is characterized by two parameters, the first of which being the *width* parameter $c \in]0,1[$. It corresponds to the width of the available charge well, while $1-c$ is the width of the bound charge well. The second parameter, the *diffusion rate* parameter $p > 0$, is the factor of proportionality of the difference in fluid levels of both wells, namely $A(t)/c$ and $B(t)/(1-c)$, and thus governs the rate with which bound charge is converted to available charge and vice-versa.

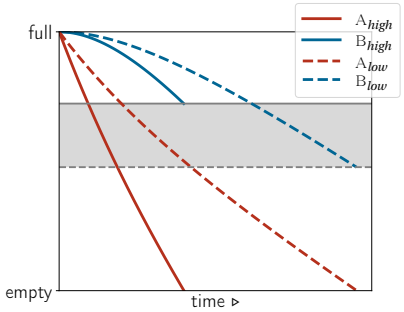
KiBaM ODE System. Mathematically, the KiBaM state of charge evolves as indicated by two coupled differential equations:

$$\begin{aligned} \dot{A}(t) &= -\ell(t) + p\left(\frac{B(t)}{1-c} - \frac{A(t)}{c}\right), \\ \dot{B}(t) &= \qquad\quad p\left(\frac{A(t)}{c} - \frac{B(t)}{1-c}\right). \end{aligned} \tag{3.1}$$

The dynamics of the KiBaM account for a couple of non-linear effects that can be observed on real-world batteries, most notably the *recovery effect* and the *rate-capacity effect*. The recovery effect can be summarized as follows. If a resting period follows a discharging period, the available charge well recovers to a certain degree, via the diffusion of bound charge to available charge. Note that the effect is symmetric: After a charging period, the available charge well seems to deteriorate, although in this case the meaning of the word recovery is contradictory. The rate-capacity effect describes another simple phenomenon. The faster the discharge, the smaller the battery's effective capacity. This means, the more a (discharging) load surpasses the bound-to-available charge conversion rate, the worse the battery performs. Also this effect is somewhat symmetric when considering a charging scenario. In the end, both effects are rooted in the fact that charge conversion is taking place. An illustration of both phenomena is given in Figure 3.2.



The recovery effect. When the battery is not strained by any load, both charge levels approach each other, due to diffusion. The available charge seems to recover.



The rate-capacity effect. A high discharging load (solid lines) leaves a lot more bound charge in a battery at the time of depletion, compared to when a low discharging load is applied (dashed lines).

Figure 3.2: Illustrations of the recovery effect and the rate-capacity effect captured by the KiBaM.

Solving The KiBaM ODE System

It is possible to derive a solution of the ODEs at time t, after applying a constant load ℓ for t time units, for instance by using Laplace transforms. We can express it as a vector valued linear map, taking the initial available and bound charge a_0 and b_0 as argument:

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} \mathrm{d}_{aa}(t) & \mathrm{d}_{ab}(t) & \mathrm{d}_{a\ell}(t) \\ \mathrm{d}_{ba}(t) & \mathrm{d}_{bb}(t) & \mathrm{d}_{b\ell}(t) \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ b_0 \\ \ell(t) \end{bmatrix} \tag{3.2}$$

with

$$\begin{aligned} \mathrm{d}_{aa} &= (1-\mathsf{c})\mathrm{e}^{-kt} + \mathsf{c}, & \mathrm{d}_{ba} &= 1 - \mathrm{d}_{aa} \\ \mathrm{d}_{ab} &= -\mathsf{c}\cdot\mathrm{e}^{-kt} + \mathsf{c}, & \mathrm{d}_{bb} &= 1 - \mathrm{d}_{ab} \\ \mathrm{d}_{a\ell} &= \frac{(1-\mathsf{c})\cdot(\mathrm{e}^{-kt}-1)}{k} - t\cdot\mathsf{c}, & \mathrm{d}_{b\ell} &= -t - \mathrm{d}_{a\ell}, \end{aligned}$$

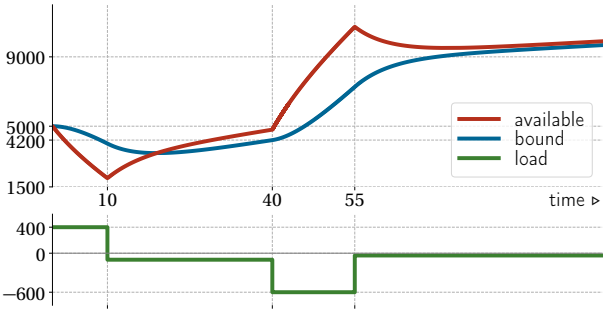
where $k = \mathsf{p}/(\mathsf{c}\cdot(1-\mathsf{c}))$ and where we omit the time argument t of the matrix coefficients if it is clear from the context.

The coefficients $\mathrm{d}_{a\ell}$ and $\mathrm{d}_{b\ell}$ of ℓ do not sum to 1, because the non-zero load ℓ makes the total power in the battery change. The above map is a vector valued reformulation of equations found in the literature [29]. Here the KiBaM ODE solution is given by

$$\begin{aligned} a_t &= a_0\mathrm{e}^{-kt} + \frac{(s_0 k\mathsf{c}-\ell)(1-\mathrm{e}^{-kt})}{k} - \frac{\ell\mathsf{c}(kt-1+\mathrm{e}^{-kt})}{k} \\ b_t &= b_0\mathrm{e}^{-kt} + s_0(1-\mathsf{c})(1-\mathrm{e}^{-kt}) \quad - \frac{\ell(1-\mathsf{c})(kt-1+\mathrm{e}^{-kt})}{k} \end{aligned}$$

where $s_0 = a_0 + b_0$. Regrouping the terms according to a_0, b_0 and ℓ leads to the vector valued formulation.

Example 1. We illustrate the evolution of the KiBaM state of charge as time passes under the assumption of symmetry of charging and discharging below.



The initial available charge decreases heavily due to the load 400 but the restricted diffusion makes the bound charge decrease only slowly up to time 10; after that the battery undergoes a mild recharge, and so on. At all times the bound charge approaches the available charge by a speed proportional to the difference of the two values.

Definition 2 — *SoCs.* Throughout this document we use $[a;b]$ to denote a state of charge as a row vector and denote the set of SoCs by $\mathbb{S} := \mathbb{R}^2$.

We introduce other basic concepts and notation.

Definition 3. Following convention, we interpret arithmetic or comparison operations on SoCs to be componentwise, hence for $\star \in \{+,-,\cdot,/\}$ we have

$$[a_0;b_0] \star x := \begin{cases} [a_0 \star a_1; b_0 \star b_1] & \text{if } x = [a_1;b_1] \\ [a_0 \star \mathrm{k}; b_0 \star \mathrm{k}] & \text{if } x = \mathrm{k} \in \mathbb{R} \end{cases}$$

and for $\triangleright \in \{<,\leq,=,>,\geq\}$ we have

$$[a_0;b_0] \triangleright x := \begin{cases} a_0 \triangleright a_1 \wedge b_0 \triangleright b_1 & \text{if } x = [a_1;b_1] \\ a_0 \triangleright \mathrm{k} \wedge b_0 \triangleright \mathrm{k} & \text{if } x = \mathrm{k} \in \mathbb{R} \end{cases}.$$

Note that componentwise comparisons do not correspond to lexicographic orders.

In addition, we define the order $<_{\mathbb{K}}$ when we want to impose a strict order on one of the components.

Definition 4 — *The order* $<_{\mathbb{K}}$. For two SoCs $[a_0;b_0],[a_1;b_1] \in \mathbb{S}$ we define

$$[a_0;b_0] <_{\mathbb{K}} [a_1;b_1] := \begin{cases} a_0 < a_1 \wedge b_0 \leq b_1 \\ \vee \\ a_0 \leq a_1 \wedge b_0 < b_1 \end{cases}.$$

Definition 5 — *Equilibrium.* We say that a SoC $[a;b] \in \mathbb{S}$ is *in equilibrium* iff no diffusion is taking place, i.e.

$$\frac{a}{\mathsf{c}} = \frac{b}{1-\mathsf{c}}.$$

The load model we want to investigate is described by *tasks* and sequences thereof.

Definition 6 — *Tasks.* A *task* (Δ, ℓ) is a pair of a positive time duration $\Delta > 0$ and a load ℓ with which the battery is strained throughout that time duration, i.e. for $0 \leq t < \Delta$, we have $\ell(t) = \ell$. We denote the set of tasks by $\mathbb{T} := \mathbb{R}_{>0} \times \mathbb{R}$.

A sequence of tasks thus induces a piecewise constant load sequence.

Definition 7 — *The operator* $\mathbb{K}$. We define the KiBaM successor SoC in terms of an operator $\mathbb{K} : \mathbb{T} \times \mathbb{S} \to \mathbb{S}$. For an initial SoC $[a_0; b_0]$ and a task (Δ, ℓ) we define

$$\mathbb{K}_{(\Delta,\ell)}[a_0; b_0] := [a_\Delta; b_\Delta]$$

according to Equation 3.2.
A task sequence $(T_i)_{i=0}^N \in \mathbb{T}^+$ can be handled iteratively by functional composition, i.e.

$$\mathbb{K}_{(T_i)_{i=0}^N} := \mathbb{K}_{T_0} \circ \cdots \circ \mathbb{K}_{T_N}.$$

We are now able to establish that $\mathbb{K}$ is monotonous with respect to the initial SoC, given a fixed task T, or in different terms, $\mathbb{K}$ preserves the order $\leq$.

Lemma 1 — $\mathbb{K}$ *preserves* $<_{\mathbb{K}}$. Let $S_0, S_1 \in \mathbb{S}$ be two SoCs and $T \in \mathbb{T}$ be a task. We have that

$$S_0 <_{\mathbb{K}} S_1 \Longrightarrow \mathbb{K}_T S_0 <_{\mathbb{K}} \mathbb{K}_T S_1.$$

Proof:

We first conclude that the partial derivatives of $\mathbb{K}$

$$\frac{\partial \mathrm{A}}{\partial a} = \mathrm{d}_{aa}, \quad \frac{\partial \mathrm{A}}{\partial b} = \mathrm{d}_{ab}$$
$$\frac{\partial \mathrm{B}}{\partial a} = \mathrm{d}_{ba}, \quad \frac{\partial \mathrm{B}}{\partial b} = \mathrm{d}_{bb}$$

are constant in $[a;b]$. Using simple arithmetic in conjunction with the fact that $\ln \mathrm{e}^x = x$ and $\ln 1 = 0$, as well as the facts that Δ and p are positive and $0 < \mathsf{c} < 1$, it is easy to show that the partial derivatives are all strictly positive, and thus $\mathbb{K}$ is strictly monotonically increasing with respect to the initial SoC. The claim follows immediately. ■

Note that the above lemma is independent of the load ℓ. It simply argues that two SoCs do not swap order when the same task is applied. In other words, the larger of two battery states cannot be "overtaken" by the smaller SoC, when the same task is applied, which is a natural and intuitive property of any reasonable battery model.

***Corollary* 1** ($\mathbb{K}$ preserves $\leq$)**.** Let $S_0, S_1 \in \mathbb{S}$ be two SoCs and $T \in \mathbb{T}$ be a task. We have that

$$S_0 \leq S_1 \Longrightarrow \mathbb{K}_T S_0 \leq \mathbb{K}_T S_1.$$

Proof:

By Lemma 1 and the fact that $\mathbb{K}$ is deterministic. ■

A similarly intuitive and important result is that the KiBaM exhibits a lower state of charge the higher the load.

Lemma 2. Let Δ be a positive duration, $[a;b] \in \mathbb{S}$ be a SoC and ℓ_0, ℓ_1 be two loads. We have that

$$\ell_0 \geq \ell_1 \Longrightarrow \mathbb{K}_{(\Delta,\ell_0)}[a;b] \leq \mathbb{K}_{(\Delta,\ell_1)}[a;b].$$

Proof:

The partial derivatives of $\mathbb{K}$ with respect to the load variable ℓ are

$$\frac{\partial \mathrm{A}}{\partial \ell} = \mathrm{c}\mathrm{e}^{-k\Delta} - \mathrm{c} \quad \text{and} \quad \frac{\partial \mathrm{B}}{\partial \ell} = (\mathrm{c}-1)\mathrm{e}^{-k\Delta}(\mathrm{e}^{k\Delta} - 1).$$

Using $\ln 1 = 0$ and $\ln \mathrm{e}^x = x$, as well as the facts that c, k and Δ are non-negative, we find that both expressions are strictly negative, thereby proving the claim. ■

From this, we are now able to derive a crucial property of $\mathbb{K}$.

Lemma 3 — *Threshold monotonicity.* Let $\bowtie \in \{<,>\}$, $(\Delta,\ell) \in \mathbb{T}$ be a task with $0 \bowtie \ell$, $\kappa \in \mathbb{R}$ a *threshold* and $[a_0;b_0]$ be an initial SoC with $[a_0;b_0] \bowtie [\mathrm{c};1-\mathrm{c}]\cdot\kappa$. If there is a $\delta \in]0,\Delta]$ with $[\mathrm{c}\kappa;b_\delta] = \mathbb{K}_{(\delta,\ell)}[a_0;b_0]$, then

1. $b_\delta \bowtie (1-c)\kappa$, and
2. $a_{\delta'} \bowtie c\,\kappa$ for all $\delta < \delta' \leq \Delta$.

Intuitively, $\mathbb{K}$ satisfies the following version of monotonicity: Given a mutual threshold on both charge levels, we have that (1) the available charge reaches it before the bound charge does, and (2) the available charge will not cross it again during the same task.

Example 2. We notice that neither the available nor the bound charge are monotonic with respect to the time variable in the standard sense. In Exam-

ple 1, the bound charge is not monotonic on the interval $[10,40]$, the available charge is not monotonic on $[55,100]$. However, for instance, on $[10,40]$, the available charge is the first to exceed the value 4200 and never falls below that boundary again during that task.

From Lemmas 1, 2 and 3, we derive the following intuitive property which we will need later on. We use the $\bullet$ symbol for *wildcards*, meaning that we do not really care about its value.

***Corollary* 2.** Let $\triangleright \in \{<,>\}$, ℓ be a load with $\ell \triangleright 0$, $S_0, S_1 \in \mathbb{S}$ be two SoCs, and $\kappa \in \mathbb{R}$ be a threshold such that $S_0 \triangleright_{\mathbb{K}} S_1 \triangleright [\mathsf{c}; 1-\mathsf{c}] \cdot \kappa$.
If there is $\delta_0 \in]0,\Delta]$ with $\mathbb{K}_{(\delta_0,\ell)} S_0 = [\mathsf{c}\kappa; \bullet]$, there is also $\delta_1 \in]0,\Delta]$ with $\mathbb{K}_{(\delta_1,\ell)} S_1 = [\mathsf{c}\kappa; \bullet]$ and $\delta_1 \triangleright \delta_0$,

The above lemma says that among two SoCs, the larger one reaches a threshold first while charging, but last while discharging.

3.2 Depletion

So far, the operator $\mathbb{K}$ is defined on any given SoC and its evolution potentially spans the whole range of SoCs, including the negatives. While this makes sense mathematically, it is counter intuitive in the context of a battery state. This means, according to the KiBaM as is, batteries have no concept of depletion, nor of capacity limits and hence are objects of essentially arbitrary capacity. In this section, we introduce the concept of depletion.

We define the region of critically low SoCs in terms of a battery depletion level depl, which in turn induces depletion thresholds on available and bound charge quantities by $[\underline{a}; \underline{b}] := [\mathsf{c}; 1-\mathsf{c}] \cdot \mathsf{depl}$. Intuitively it makes sense that depl is non-negative, however there is no mathematical reason to enforce this, since the KiBaM system is defined on the negatives as well.

Definition 8 — *Safe SoCs and depletion.* A SoC S is *safe* iff $S > [\underline{a}; \underline{b}]$, and we denote the set of safe SoCs by $\underline{\mathbb{S}} := \mathbb{R}_{>\underline{a}} \times \mathbb{R}_{>\underline{b}}$. A SoC is *depleted* if its available charge is lower than the depletion threshold, i.e. if $a \leq \underline{a}$.

Note that depletion is defined only on the available charge dimension of a SoC, which is evident based on the naming of both SoC quantities. There is no need to discriminate between depleted SoCs, as none of them can support any further discharging tasks. Hence, we designate $\bot := [\underline{a}; \underline{b}]$, to be the *canonical* depletion SoC, and denote with $\underline{\mathbb{S}}_\bot$ the set of safe SoCs including depletion.

Mathematically, we extend the $\mathbb{K}$ operator with depletion limits as follows.

Definition 9 — *The operator* $\underline{\mathbb{K}}$. We define the operator $\underline{\mathbb{K}} : \mathbb{T} \times \underline{\mathbb{S}}_\bot \to \underline{\mathbb{S}}_\bot$ for

each task T as:

$$\underline{\mathbb{K}}_T S \coloneqq \begin{cases} \bot, & \text{if } S = \bot \\ \bot, & \text{if } \mathbb{K}_T S \text{ is depleted} \\ \mathbb{K}_T S, & \text{otherwise} \end{cases}$$

The operator $\underline{\mathbb{K}}$ inherits all notational aspects of $\mathbb{K}$.

Definition 10 — *Depleting tasks.* We call T a *depleting task* if for an initial safe SoC S, we have $\underline{\mathbb{K}}_T S = \bot$.

***Corollary* 3.** For safe initial SoCs and each depleting task, the available charge will drop below the depletion threshold before the bound charge does.

Proof:

By Corollary 2. ■

***Corollary* 4** (All's well that ends well)**.** For a safe initial SoC S and a task $(\Delta, \ell) \in \mathbb{T}$ the following holds: If $\underline{\mathbb{K}}_{(\Delta,\ell)} S \neq \bot$ then for $0 < \delta < \Delta$ we also have $\underline{\mathbb{K}}_{(\delta,\ell)} S \neq \bot$.

Proof:

If any intermediate state would be depleted, then by Lemma 3, the final SoC would be depleted as well. $\lightning$ ■

In essence, Corollary 4 says that, when tracking a SoC along a sequence of tasks, we need only care about the SoCs at the end of each task. If at the end of each task the SoC is safe, it must have been safe at any intermediate time point.

We end this section by lifting a few lemmas from before to the depletion case, like the fact that $\underline{\mathbb{K}}$ preserves $\leq$ (and $<_{\mathbb{K}}$ if the resulting SoCs are safe).

***Corollary* 5.** Let $S_0, S_1 \in \underline{\mathbb{S}}$ be two safe SoCs and $T \in \mathbb{T}$ be a task. We have that

$$S_0 <_{\mathbb{K}} S_1 \wedge \underline{\mathbb{K}}_T S_0 \neq \bot \neq \underline{\mathbb{K}}_T S_1 \implies \underline{\mathbb{K}}_T S_0 <_{\mathbb{K}} \underline{\mathbb{K}}_T S_1.$$

Proof:

By the fact that $\underline{\mathbb{K}}$ degenerates to $\mathbb{K}$ in this case and the fact that $\mathbb{K}$ preserves $<_{\mathbb{K}}$(Lemma 1). ■

The above corollary says that if T is not a depleting task for neither S_0 nor S_1, then $\underline{\mathbb{K}}$ preserves $<_{\mathbb{K}}$.

Lemma 4 — $\underline{\mathbb{K}}$ *preserves* $\leq$. Let $S_0, S_1 \in \underline{\mathbb{S}}_\bot$ be two SoCs and $T \in \mathbb{T}$ be a task.

We have that

$$S_0 \leq S_1 \Longrightarrow \underline{\mathbb{K}}_T S_0 \leq \underline{\mathbb{K}}_T S_1.$$

Proof:

We need to consider three cases.

$S_0 = \bot = S_1$: We have $\underline{\mathbb{K}}_T S_0 = \underline{\mathbb{K}}_T S_1 = \bot$ by definition, and the claim follows.

$S_0 = \bot, S_1 \in \underline{\mathbb{S}}$: Then $\underline{\mathbb{K}}_T S_0 = \bot$ and $\underline{\mathbb{K}}_T S_1 = \bot$ or $\underline{\mathbb{K}}_T S_0 \in \underline{\mathbb{S}}$. In either case, the conclusion of the implication is fulfilled.

$S_0, S_1 \in \underline{\mathbb{S}}$: Then, if $\underline{\mathbb{K}}_T S_1 = \bot$ then also $\underline{\mathbb{K}}_T S_0 = \bot$ because of Corollary 1, and the claim follows. If $\underline{\mathbb{K}}_T S_0 = \bot$ but $\underline{\mathbb{K}}_T S_1 \in \underline{\mathbb{S}}$ then the claim follows immediately. Lastly, if $\underline{\mathbb{K}}_T S_0, \underline{\mathbb{K}}_T S_1 \in \underline{\mathbb{S}}$ then the claim follows by Corollary 1, since $\underline{\mathbb{K}}$ reduces to $\mathbb{K}$. ■

Lemma 5. Let Δ be a positive duration, $[a;b] \in \underline{\mathbb{S}}_\bot$ be a SoC and ℓ_0, ℓ_1 be two loads. We have that

$$\ell_0 \geq \ell_1 \Longrightarrow \underline{\mathbb{K}}_{(\Delta,\ell_0)}[a;b] \leq \underline{\mathbb{K}}_{(\Delta,\ell_1)}[a;b].$$

Proof:

The proof relies on the same case distinction as in the proof of the previous lemma, and is analogous in nature. ■

3.3 Capacity Limits

Real-world batteries are evidently not infinite energy storage devices. The KiBaM does not reflect this, since the $\underline{\mathbb{K}}$-operator can attain arbitrarily large values. Hence, we naturally have to enforce capacity limits. Let $\mathsf{cap} \in \mathbb{R}_{>\mathsf{depl}}$ be the capacity limit. Then $[\overline{a};\overline{b}] := [\mathsf{c};(1-\mathsf{c})] \cdot \mathsf{cap}$ are the induced limits on available and bound charge. Conceptually, we don't want a battery to be able to be charged beyond its capacity limits. Again, there's no mathematical need to define cap to be non-negative, although it absolutely makes sense intuitively to view this threshold as non-negative.

Definition 11 — *Saturation.* We call a SoC S *saturated* and *over-saturated*, iff $a = \overline{a}$ and $a > \overline{a}$, respectively. We denote with $\overline{\mathbb{S}}$ the safe saturated SoCs, with $\vec{\mathbb{S}}$ the safe over-saturated SoCs, with $\underline{\overline{\mathbb{S}}}$ we denote safe, but not over-saturated SoCs and finally with $\underline{\overline{\mathbb{S}}}_\bot$ we denote $\underline{\overline{\mathbb{S}}}$ including depletion. Formally, we define

$$\overline{\mathbb{S}} := \{[\overline{a};b] \mid b \in \mathbb{R}\}, \qquad \vec{\mathbb{S}} := \{[a;b] \mid a > \overline{a} \wedge b \in \mathbb{R}\},$$
$$\underline{\overline{\mathbb{S}}} := \underline{\mathbb{S}} \setminus \vec{\mathbb{S}}, \qquad \underline{\overline{\mathbb{S}}}_\bot := \underline{\overline{\mathbb{S}}} \cup \{\bot\}.$$

Definition 12 — *Saturating Tasks.* We say that task T is *saturxating*, iff $\mathbb{K}_T S$ is saturated or over-saturated.

We conclude by deducing the following intuitive fact about the KiBaM with capacity limits from previous results.

***Corollary* 6.** For unsaturated SoCs $S \in \underline{\mathbb{S}} \setminus \overline{\mathbb{S}}$ and a corresponding saturating task T we have that the available charge will reach the saturation threshold before the bound charge does.

Proof:

By Corollary 2. ■

Staying Saturated

Charging and discharging are not fully symmetric: A battery with empty available charge can no longer power its task, contrary to a battery with full available charge that *continues to operate*. We thus need to consider its further charging behavior.

When the available charge is at its capacity $\overline{a} = \mathsf{c} \cdot \mathsf{cap}$ and is still further charged by a *sufficiently* high charging current, its value stays constant and only the bound charge increases due to diffusion.

$$\begin{aligned} \dot{\mathrm{A}}(t) &= 0, \\ \dot{\mathrm{B}}(t) &= \mathsf{p}\left(\mathsf{cap} - \frac{\mathrm{B}(t)}{1-\mathsf{c}}\right) \end{aligned} \tag{3.3}$$

The differential equations above describe the behavior of the battery at time t only if the charging load to the available charge well is *sufficient* to compensate the diffusion, i.e. for each t, we have

$$-\ell(t) \geq \dot{\mathrm{B}}(t).$$

Since $\ell(t)$ is constant during a task, say ℓ, and the diffusion is decreasing over time, the charging load is sufficient throughout the whole task if and only if it is sufficient at the very beginning of a task, i.e.

$$-\ell \geq \dot{\mathrm{B}}(0).$$

Hence, for an initial bound charge b_0 we define the condition whether the charging current is sufficient by

$$\ell \leq \ell_{\mathsf{sat}}(b_0) \coloneqq \mathsf{p}\left(\frac{b_0}{1-\mathsf{c}} - \mathsf{cap}\right) \tag{3.4}$$

Later on we will need this expression solved for b_0, so as to find the least bound charge level that manages to compensate the diffusion with respect to a certain charging load ℓ.

Definition 13. The *least* bound charge of a saturated SoC that compensates the diffusion given a charging load ℓ is given by

$$\begin{aligned} b_\ell^{\mathsf{sat}} &:= \left(\frac{\ell}{\mathsf{p}} + \mathsf{cap}\right) \cdot (1-\mathsf{c}) \\ &= \frac{\ell}{\mathsf{p}}(1-\mathsf{c}) + \mathsf{cap} \cdot (1-\mathsf{c}) \\ &= \frac{\ell}{\mathsf{p}}(1-\mathsf{c}) + \overline{b} \end{aligned}$$

By solving the ODE (3.3) and using Inequality (3.4), we obtain the dynamics of a saturated battery that is further charged by a sufficient charging load in the form of a simple function.

Lemma 6. Let $\Delta > 0$ and b_0 such that $\ell \leq \ell_{\mathsf{sat}}(b_0)$. A battery with a saturated SoC $[\overline{a}; b_0]$ attains after the task (Δ, ℓ) the state of charge $[\overline{a}; \mathrm{B}_\Delta^{\mathsf{sat}}(b_0)]$ where

$$\mathrm{B}_\Delta^{\mathsf{sat}}(b_0) = \mathrm{e}^{-\mathsf{c}k\Delta} \cdot b_0 + \left(1 - \mathrm{e}^{-\mathsf{c}k\Delta}\right) \cdot \overline{b} \tag{3.5}$$

and k again stands for $\mathsf{p}/(\mathsf{c} \cdot (1-\mathsf{c}))$.

Proof:

By solving the ODE system 3.3 we conclude that B takes the form of $\mathrm{B}^{\mathsf{sat}}$. ■

Note that the resulting bound charge evolution of a saturated battery $\mathrm{B}_\Delta^{\mathsf{sat}}(b_0)$ does not further depend on ℓ, i.e. one cannot make the battery charge faster by increasing the charging load. Furthermore, for a fixed b_0, the curve of $\Delta \mapsto \mathrm{B}_\Delta^{\mathsf{sat}}(b_0)$ is a negative exponential starting from the point b_0 with the full capacity $\overline{b}$ of the bound charge being its limit. Thus, Lemma 6 also reveals that the bound charge in finite time never reaches its capacity and there is no need to describe this situation separately.

We lift the concept of Lemma 6 to an operator on SoCs.

Definition 14. For a positive duration $\Delta > 0$ and a SoC $[a; b]$ we define

$$\mathbb{K}_\Delta^{\mathsf{sat}}[a; b] := [a; \mathrm{B}_\Delta^{\mathsf{sat}}(b)]$$

We conclude with showing that $\mathbb{K}^{\mathsf{sat}}$ is monotonically increasing in b.

Lemma 7 — $\mathbb{K}^{\mathsf{sat}}$ *preserves* $\leq$. Given $S_0, S_1 \in \mathbb{S}$ and a duration $\Delta > 0$ we have

$$S_0 \leq S_1 \Longrightarrow \mathbb{K}_\Delta^{\mathsf{sat}} S_0 \leq \mathbb{K}_\Delta^{\mathsf{sat}} S_1$$

Proof:

> The first component stays constant, so it suffices to show that $\mathrm{B}_{\Delta}^{\mathsf{sat}}$ is monotonically increasing. The derivative of $\mathrm{B}_{\Delta}^{\mathsf{sat}}(b)$ with respect to b is $\mathrm{e}^{-\mathsf{c}k\Delta}$, which is positive, thereby establishing the claim. ■

Saturating The Battery

Each non-saturated and safe SoC can eventually be saturated under $\mathbb{K}$ (and therefore $\underline{\mathbb{K}}$) via indefinite charging. We are interested in the time point of saturation, since conceptually the dynamics of the battery will change from this point onwards.

Hence, we want $\overline{\Delta}$ such that the first component of $\mathbb{K}_{(\overline{\Delta},\ell)}S$ is exactly $\overline{a}$. Formally, we want to solve the equation

$$a_0 \cdot \mathrm{d}_{aa}(\overline{\Delta}) + b_0 \cdot \mathrm{d}_{ab}(\overline{\Delta}) + \ell \cdot \mathrm{d}_{a\ell}(\overline{\Delta}) = \overline{a} \tag{3.6}$$

for $\overline{\Delta}$.

It turns out that this is an instance of the so-called *product logarithm*: Given a real number $\mathrm{k} \in \mathbb{R}$ solve the equation

$$x\mathrm{e}^{x} = \mathrm{k}$$

for x. The solution of such equations can be brought into closed form using the *Lambert W function* $\mathscr{W}$, i.e. the solution of $x\mathrm{e}^{x} = \mathrm{k}$ is $\mathscr{W}(\mathrm{k})$.

One can bring Equation 3.6 into a related form

$$u \cdot \mathrm{e}^{-k\overline{\Delta}} + v \cdot \overline{\Delta} + w = \overline{a}$$

where

$$\begin{aligned} u &= a_0(1-\mathsf{c}) - b_0\mathsf{c} + (\mathsf{c}+1)\cdot\frac{\ell}{k} \\ v &= -\ell\mathsf{c} \\ w &= \overline{a} - a_0\mathsf{c} - b_0\mathsf{c} - (1-\mathsf{c})\cdot\frac{\ell}{k}. \end{aligned}$$

It turns out that solving equations of the form $\mathrm{e}^{x} + x = \mathrm{k}$ for x, meaning equations where the variable of interest appears in an exponential as well as a linear term, can be reduced to the product logarithm itself. Generally, the solution to $x = q + r\mathrm{e}^{sx}$ is given by $q - \frac{1}{s}\mathscr{W}(-rs\mathrm{e}^{qs})$ Thus, the solution to Equation 3.6 is

$$-\mathscr{W}\left(\frac{u}{v}\mathrm{e}^{-\frac{w}{v}}\right) - \frac{w}{v}.$$

Unfortunately, the Lambert W function can not be expressed using elementary functions, meaning the solution is a *transcendental* number, thus its exact computation is impossible. It can however be approximated using numerical methods [7]. Also, the $\mathscr{W}$ function generally attains multiple solutions, it is thus a multi-valued function. Whenever we refer the solution to Equation 3.6 we implicitly mean the largest of these solutions.

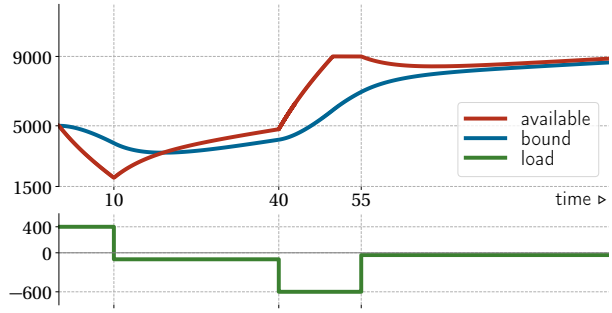
All the previous building blocks allow us to express the SoC of a KiBaM after powering a given task (Δ, ℓ) when considering capacity limits.

Definition 15. We define the operator $\overline{\underline{\mathbb{K}}} : \mathbb{T} \times \overline{\underline{\mathbb{S}}}_\perp \to \overline{\underline{\mathbb{S}}}_\perp$ for each task $T = (\Delta, \ell)$ and each SoC S as:

$$\overline{\underline{\mathbb{K}}}_T S := \begin{cases} \mathbb{K}^{\mathsf{sat}}_\delta \circ \underline{\mathbb{K}}_{(\overline{\Delta},\ell)} S & \text{if } \mathbb{K}_T S \text{ is over-saturated} \\ \underline{\mathbb{K}}_T S & \text{otherwise} \end{cases}$$

where $\overline{\Delta}$ is the largest solution of Equation 3.6 and $\delta := \Delta - \overline{\Delta}$.

Example 3. If we put a capacity limit of $\overline{a} = 9000$ to the previous scenario, the battery ends up with a slightly smaller charge at time 100. The computation of the final SoC changes only in the interval $[40, 55]$. Here, instead of $\mathbb{K}_{(15,-600)}$, we apply $\mathbb{K}_{(\overline{\Delta},-600)}$ for the first $\overline{\Delta} \approx 7.8$ time units, followed by $\mathbb{K}^{\mathsf{sat}}_{15-\overline{\Delta}}$.



We conclude again that the operator $\overline{\underline{\mathbb{K}}}$ respects the order $\leq$.

Lemma 8 — $\overline{\underline{\mathbb{K}}}$ *preserves* $\leq$. Given $S_0, S_1 \in \overline{\underline{\mathbb{S}}}_\perp$ and a task $T \in \mathbb{T}$ we have

$$S_0 \leq S_1 \Longrightarrow \overline{\underline{\mathbb{K}}}_T S_0 \leq \overline{\underline{\mathbb{K}}}_T S_1.$$

Proof:

The only interesting case is the case where $T = (\Delta, \ell)$ is a saturating task. Every other case is already established by Lemma 4. Let T be a saturating (and thus charging) task. According to Lemma 2 with $\kappa := \mathsf{cap}$ we have that S_1 will be saturated (with an unsaturated bound charge) exactly at time $\overline{\Delta}_1$, meaning $\underline{\mathbb{K}}_{(\overline{\Delta}_1,\ell)} S_1 = [\overline{a}; b_1^{\mathsf{mid}}]$. We also have by Corollary 1 that S_0 is not yet saturated, thus $\underline{\mathbb{K}}_{(\overline{\Delta}_0,\ell)} S_0 = [a; b_0^{\mathsf{mid}}]$, with $a < \overline{a}$ as well as $b_0^{\mathsf{mid}} \leq b_1^{\mathsf{mid}}$.

Case 1: Let us first assume that S_0 is not saturated during the remainder of the task, or right at its end Δ, such that the dynamics are governed by $\mathbb{K}$. We now need to show that

$$\mathbb{K}_{(\delta,\ell)}[a; b_0^{\mathsf{mid}}] \leq \mathbb{K}^{\mathsf{sat}}_\delta[\overline{a}; b_1^{\mathsf{mid}}],$$

with $\delta := \Delta - \overline{\Delta}_1$. We do so by showing the stronger result

$$\mathbb{K}_{(\delta,\ell)}[a; b_1^{\mathsf{mid}}] =: [a_\Delta; b_1^{\mathsf{end}}] \leq \mathbb{K}^{\mathsf{sat}}_\delta[\overline{a}; b_1^{\mathsf{mid}}],$$

which is valid because of $b_0^{\mathsf{mid}} \leq b_1^{\mathsf{mid}}$ and the application of Corollary 1. By assumption we already have $a_\Delta < \overline{a}$, thus we continue to show $b_1^{\mathsf{end}} \leq \mathrm{B}_\delta^{\mathsf{sat}}(b_1^{\mathsf{mid}})$. We fall back to the ODE formulation of the two scenarios to show that the derivative of the "staying saturated" scenario is larger at every remaining time point $t \in [\delta, \Delta]$, which is sufficient, because we start at the same bound charge level b_1^{mid}. Hence we show

$$\mathsf{p}\left(\frac{\mathrm{A}(t)}{\mathsf{c}} - \frac{\mathrm{B}(t)}{1-\mathsf{c}}\right) \leq \mathsf{p}\left(\frac{\overline{a}}{\mathsf{c}} - \frac{\mathrm{B}(t)}{1-\mathsf{c}}\right).$$

This is equivalent to showing $\mathrm{A}(t) \leq \overline{a}$, which is always fulfilled.

Case 2: Let us now assume that S_0 is saturated during the remainder of the task, say at time $\overline{\Delta}_0$. By exchanging Δ by $\overline{\Delta}_0$ in the above argument we arrive at

$$\overline{\underline{\mathbb{K}}}_{(\overline{\Delta}_0,\ell)} S_0 =: [\overline{a}; b_{\overline{\Delta}_0,0}] \leq [\overline{a}; b_{\overline{\Delta}_0,1}] := \overline{\underline{\mathbb{K}}}_{(\overline{\Delta}_0,\ell)} S_1.$$

Both SoCs remain saturated for the remainder of the task, i.e. their dynamics are governed by $\mathbb{K}^{\mathsf{sat}}$. However, $\mathbb{K}^{\mathsf{sat}}$ preserves the order $\leq$ by Lemma 7, which establishes the claim. ■

Lemma 9. Let Δ be a positive duration, $[a;b] \in \overline{\mathbb{S}}_\perp$ be a SoC and ℓ_0, ℓ_1 be two loads. We have that

$$\ell_0 \geq \ell_1 \Longrightarrow \underline{\mathbb{K}}_{(\Delta,\ell_0)}[a;b] \leq \underline{\mathbb{K}}_{(\Delta,\ell_1)}[a;b].$$

Proof:

Like for the previous lemma, we focus on the interesting case of saturation. Let us thus assume that (Δ, ℓ_1) is a saturating task and let $\overline{\Delta}_1$ be the saturation time point. By Lemma 5 we have that

$$\mathbb{K}_{(\overline{\Delta}_1,\ell_1)}[a;b] =: [\overline{a}; b_1^{\mathsf{mid}}] \geq [a_0^{\mathsf{mid}}; b_0^{\mathsf{mid}}] := \mathbb{K}_{(\overline{\Delta}_1,\ell_0)}[a;b]$$

By definition of $\overline{\mathbb{K}}$, for the remainder of the task $\delta := \Delta - \overline{\Delta}_1$ the dynamics of $[\overline{a}; b_1^{\mathsf{mid}}]$ are governed by $\mathbb{K}^{\mathsf{sat}}$.

Case 1: Let us assume first that (Δ, ℓ_0) is not a saturating task for $[a;b]$ (or it is saturated exactly at time Δ) such that its dynamics is governed by $\mathbb{K}$. Furthermore, let us instead of $[a_0^{\mathsf{mid}}; b_0^{\mathsf{mid}}]$ investigate the SoC $[a_0^{\mathsf{mid}}; b_1^{\mathsf{mid}}]$, which is valid since $\mathbb{K}$ is monotonically increasing in the SoC variables (Corollary 1). Like in the previous lemma, we fall back to the ODE formulation and show that the derivative of the bound charge level in the saturated scenario is larger at every remaining time point $t \in [\delta, \Delta]$, i.e.

$$\mathsf{p}\left(\frac{\mathrm{A}(t)}{\mathsf{c}} - \frac{\mathrm{B}(t)}{1-\mathsf{c}}\right) \leq \mathsf{p}\left(\frac{\overline{a}}{\mathsf{c}} - \frac{\mathrm{B}(t)}{1-\mathsf{c}}\right).$$

This boils down to showing $\overline{a} \geq \mathrm{A}(t)$, which is a given.

Case 2: Let us now assume that (Δ, ℓ_0) is saturating, say the saturation happens at time $\overline{\Delta}_0 \geq \overline{\Delta}_1$. By exchanging Δ by $\overline{\Delta}_0$ in the above argument we arrive at

$$\overline{\underline{\mathbb{K}}}_{(\overline{\Delta}_0,\ell_0)}[a;b] =: [\overline{a}; b_{\overline{\Delta}_0,\ell_0}] \leq [\overline{a}; b_{\overline{\Delta}_0,\ell_1}] := \overline{\underline{\mathbb{K}}}_{(\overline{\Delta}_0,\ell_1)}[a;b].$$

Both SoCs remain saturated for the remainder of the task, i.e. their dynamics are governed by $\mathbb{K}^{\mathsf{sat}}$. However, $\mathbb{K}^{\mathsf{sat}}$ is independent of the load and preserves the order $\leq$ by Lemma 7, which establishes the claim. ■

3.4 Approximation Of Saturation Time Points

Due to the non-elementary character of the saturation time point, we rely on approximations of it.

With a test of saturation, an initial lower and upper bound of the saturation time point, given by 0 and the task duration Δ itself, we have all the ingredients for an interval halving algorithm to approximate the saturation time points from below and above (see Algorithm 1).

Input : $[a_0; b_0]$, a saturating task (Δ, ℓ) and ε
Output : $[\overline{\Delta}_\uparrow, \overline{\Delta}_\downarrow]$ with $\overline{\Delta}_\downarrow - \overline{\Delta}_\uparrow \leq \varepsilon$ and $\overline{\Delta} \in [\overline{\Delta}_\uparrow, \overline{\Delta}_\downarrow]$

1 $\overline{\Delta}_\downarrow := \Delta$
2 $\overline{\Delta}_\uparrow := 0$
3 **while** $\overline{\Delta}_\downarrow - \overline{\Delta}_\uparrow \geq \varepsilon$ **do**
4 $\quad \Delta_{\mathsf{mid}} := (\overline{\Delta}_\downarrow + \overline{\Delta}_\uparrow)/2$
5 $\quad$ **if** $\mathbb{K}_{(\Delta_{\mathsf{mid}},\ell)}[a_0; b_0]$ *is over-saturated* **then**
6 $\quad\quad \overline{\Delta}_\downarrow := \Delta_{\mathsf{mid}}$
7 $\quad$ **else**
8 $\quad\quad \overline{\Delta}_\uparrow := \Delta_{\mathsf{mid}}$

Algorithm 1: A bisection algorithm to bound the saturation time point of a saturating task.

Termination. The algorithm terminates since the difference $\overline{\Delta}_\downarrow - \overline{\Delta}_\uparrow$ (Line 4) is halved with every iteration of the loop, eventually falsifying the loop condition.

Complexity. Given a precision ε in the order of $\mathcal{O}(2^{-n})$, The algorithm runs in time $\mathcal{O}(\log n)$, given that Line 4 halves the difference $\overline{\Delta}_\downarrow - \overline{\Delta}_\uparrow$, leading to a logarithmic number of (constant time) loop body executions.

Correctness. The correctness statement and its proof are typical for bisection or interval-having methods.

Lemma 10 — *Correctness of Algorithm 1.* For an initial SoC S and a saturating task (Δ, ℓ) we have the following properties of about the output interval

$[\overline{\Delta}_{\uparrow}, \overline{\Delta}_{\downarrow}]$:

1. $\overline{\Delta} \in \left[\overline{\Delta}_{\uparrow}, \overline{\Delta}_{\downarrow}\right]$,
2. the task $(\overline{\Delta}_{\uparrow}, \ell)$ is not saturating,
3. the task $(\overline{\Delta}_{\downarrow}, \ell)$ is saturating.

Proof:

> We first conclude that 2 and 3 follow immediately from the conditional in Line 5 as well as the two branches. From 2 and 3 as well as item 2 of Lemma 3 we conclude 1. ■

Approximating $\overline{\underline{\mathbb{K}}}$ Efficiently

The problematic part in computing $\overline{\underline{\mathbb{K}}}$ is the case $a_{\Delta} > \overline{a}$ when the upper limit is reached at time $\overline{\Delta} < \Delta$; we cannot compute such time $\overline{\Delta}$ *exactly*, as it is a transcendental number. This case also disallows *exact* closed-form expressions in the next section where we address the KiBaM enriched with capacity limits and random initial SoCs. For these reasons we introduce under- and over-approximations of the SoC using simple closed-form expressions based on Algorithm 1. Special cases of these approximations are employed in later sections to obtain elegant analytical expressions and also efficient algorithms.

We require the following infrastructure.

Definition 16 — *Target loads.* Let S be a safe SoC, Δ be a positive duration and $\tilde{a}$ be an available charge target level. We denote by $\ell^{\rightarrow}_{\tilde{a},\Delta} S$ the unique load such that the first component of $\mathbb{K}_{(\Delta,\ell^{\rightarrow}_{\tilde{a},\Delta} S)} S$ equals $\tilde{a}$, i.e. the solution of

$$\mathbb{K}_{(\Delta,\ell)} S = [\tilde{a}; \bullet]$$

for ℓ. Specifically, we denote by $\vec{\ell}_{\Delta} S$ the solution of this problem with $\tilde{a} \coloneqq \overline{a}$. We say that $\vec{\ell}$ is the *least saturating load* of S. Additionally, we define the the solution of the same problem for $\mathbb{K}$'s reverse operator $\mathbb{K}^{-1}$ and denote it by $\ell^{\leftarrow}_{\tilde{a},\Delta} S$. Formally, we solve

$$\mathbb{K}^{-1}_{(\Delta,\ell)} S = [\tilde{a}; \bullet]$$

for ℓ, and denote by $\overleftarrow{\ell}_{\Delta} S$ the solution of this problem with $\tilde{a} \coloneqq \overline{a}$.
Furthermore, we define with $\mathrm{B}^{\rightarrow}_{\tilde{a},\Delta} S$ and $\mathrm{B}^{\leftarrow}_{\tilde{a},\Delta} S$ the bound charge levels attained when applying $\mathbb{K}_{(\Delta,\ell^{\rightarrow}_{\tilde{a},\Delta} S)} S$ and $\mathbb{K}^{-1}_{(\Delta,\ell^{\leftarrow}_{\tilde{a},\Delta} S)} S$, respectively, and use the shorthand $\vec{\mathrm{B}}_{\Delta} S$ and $\overleftarrow{\mathrm{B}}_{\Delta} S$ when specifically using the target $\tilde{a} \coloneqq \overline{a}$.
We omit the SoC argument and the duration index whenever it is clear from the context.

Lemma 11. Given a certain target available charge level $\tilde{a}$, a duration $\Delta \in \mathbb{R}_{>0}$

and a SoC S, the target load $\ell^{\rightarrow}_{\tilde{a},\Delta} S$ is computed by

$$\ell^{\rightarrow}_{\tilde{a},\Delta} S = -\frac{\mathrm{d}_{aa}}{\mathrm{d}_{a\ell}} \cdot a - \frac{\mathrm{d}_{ab}}{\mathrm{d}_{a\ell}} \cdot b + \frac{\tilde{a}}{\mathrm{d}_{a\ell}}$$

and the inverse target load is computed by

$$\ell^{\leftarrow}_{\tilde{a},\Delta} S = \frac{-\mathrm{d}_{bb} \cdot a + \mathrm{d}_{ab} \cdot b + (\mathrm{d}_{aa}\mathrm{d}_{bb} - \mathrm{d}_{ab}\mathrm{d}_{ba}) \cdot \tilde{a}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{a\ell}\mathrm{d}_{bb}}$$

Proof:

By straight-forward solving of the equations

$$\mathbb{K}_{(\Delta,\ell)} S = [\tilde{a}; \bullet] \quad \text{and} \quad \mathbb{K}^{-1}_{(\Delta,\ell)} S = [\tilde{a}; \bullet]$$

for ℓ. ■

In a similar mindset, we are interested in exactly those SoCs for which a certain task T is saturating or depleting. We provide the more general definition of target boundary first.

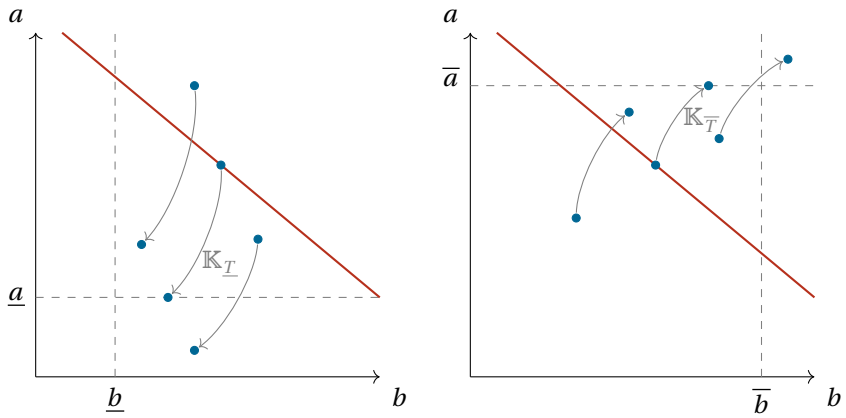
Definition 17 — *Target-, Saturation-, Depletion boundary.* Let $T \coloneqq (\Delta, \ell)$ be a task and $\tilde{a}$ be a target available charge level. The *target boundary* of T is defined by

$$\{S \mid \mathbb{K}_T S = [\tilde{a}; \bullet]\}$$

For $\tilde{a} \coloneqq \overline{a}$ we call the target boundary the *saturation boundary*, and for $\tilde{a} \coloneqq \underline{a}$ the boundary is called *depletion boundary* of T.

Let us illustrate the concept of the target boundary in a small example.

Example 4. The target boundaries are best visualized in the SoC space, i.e. in the two-dimensional space spanned by the available and bound charge. Assume a discharging task $\underline{T}$ and a charging task $\overline{T}$. The red line visualizes the depletion boundary of $\underline{T}$ (left) and the saturation boundary of $\overline{T}$ (right), respectively.



Any SoC above and to the right of the depletion boundary remains safe after $\underline{T}$, while SoCs below and to the left of the boundary are rendered unsafe. SoCs that are part of the boundary reach a SoC of the form $[\underline{a}; b]$, for some b. Analogously, SoCs above and to the right of the saturation boundary are over-saturated after $\overline{T}$, while SoCs below and to the left of the boundary remain unsaturated. SoCs that are situated exactly on the boundary end up saturated.

As the above example already indicates, the target boundaries are linear in the SoC space, which is captured by the following lemma.

Lemma 12. For a task $T \coloneqq (\Delta, \ell)$ and a target available charge level $\tilde{a}$, the target boundary is characterized by either linear function

$$a_T^{\tilde{a}}(b) = \frac{\tilde{a}}{\mathrm{d}_{aa}} - \frac{\mathrm{d}_{ab}}{\mathrm{d}_{aa}} b - \frac{\mathrm{d}_{a\ell}}{\mathrm{d}_{aa}} \ell,$$

or

$$b_T^{\tilde{a}}(a) = \frac{\tilde{a}}{\mathrm{d}_{ab}} - \frac{\mathrm{d}_{aa}}{\mathrm{d}_{ab}} a - \frac{\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}} \ell,$$

where we usually omit the subscript $\tilde{a}$ when it is implicitly clear in the context.

Proof:

We are interested in those SoCs S fulfilling $\mathbb{K}_T S = [\tilde{a}; \bullet]$, i.e.

$$\mathrm{d}_{aa} a + \mathrm{d}_{ab} b + \mathrm{d}_{a\ell} \ell = \tilde{a}$$

Solving this for a provides the first function $a_T(b)$, while solving it for b provides the second function $b_T(a)$. ■

Lemma 13. The target boundary is strictly monotonically decreasing in the SoC space.

Proof:

The derivative of $a_T(b)$ is

$$-\frac{\mathrm{d}_{ab}}{\mathrm{d}_{aa}} = \frac{\mathrm{c}e^{k\Delta} - \mathrm{c}}{\mathrm{c}e^{k\Delta} - \mathrm{c} + 1}.$$

Given that $\mathrm{c} \in [0,1]$ it is easy to show that this expression is strictly negative. ■

Lemma 14. Let $T \coloneqq (\Delta, \ell)$ be a task, $\tilde{a}$ be a target available charge level. The SoC S on T's target boundary that minimizes (maximizes) $\mathbb{K}_T S = [\tilde{a}; b_\Delta]$ is at the left (right) domain boundary.

Proof:

All SoCs on the target boundary reach a SoC $[\tilde{a}; b_\Delta]$. We optimize

$$b_\Delta = \mathrm{d}_{ba} a + \mathrm{d}_{bb} b + \mathrm{d}_{b\ell} \ell$$

given the constraint

$$\mathrm{d}_{aa} a + \mathrm{d}_{ab} b + \mathrm{d}_{a\ell} \ell = \tilde{a}.$$

In order to eliminate one variable in the objective function, we substitute a with $a_T(b)$, according to Lemma 12

$$b_\Delta = \mathrm{d}_{ba}\left(\frac{\tilde{a}}{\mathrm{d}_{aa}} - \frac{\mathrm{d}_{ab}}{\mathrm{d}_{aa}} b - \frac{\mathrm{d}_{a\ell}}{\mathrm{d}_{aa}} \ell\right) + \mathrm{d}_{bb} b + \mathrm{d}_{b\ell} \ell.$$

The derivative of b_Δ with respect to b yields

$$-\frac{\mathrm{d}_{ab}}{\mathrm{d}_{aa}} \mathrm{d}_{ba} + \mathrm{d}_{bb} = \frac{1}{\mathrm{c}e^{k\Delta} - \mathrm{c} + 1},$$

which is constant, and strictly positive for $\mathrm{c} \in]0,1[$, $\mathrm{p} > 0$ and $\Delta > 0$. This means that b_Δ is linear and is monotonically increasing with respect to b, thus proving the claim. ■

Additionally, we can prove that SoCs beyond the target boundary are larger (smaller) then each SoC hitting the target when charging (discharging).

Lemma 15. Let $\bowtie \in \{<, >\}$, $T \coloneqq (\Delta, \ell)$ be a task with $\ell \bowtie 0$, $\tilde{a}$ be a target available charge level and let S_0 be a SoC on T's target boundary. Then, for each SoC S_1 with $S_0 \bowtie_{\mathbb{K}} S_1$ we have $\mathbb{K}_T S_0 = [\tilde{a}; b] \bowtie_{\mathbb{K}} \mathbb{K}_T S_1$.

Proof:

By the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1). ■

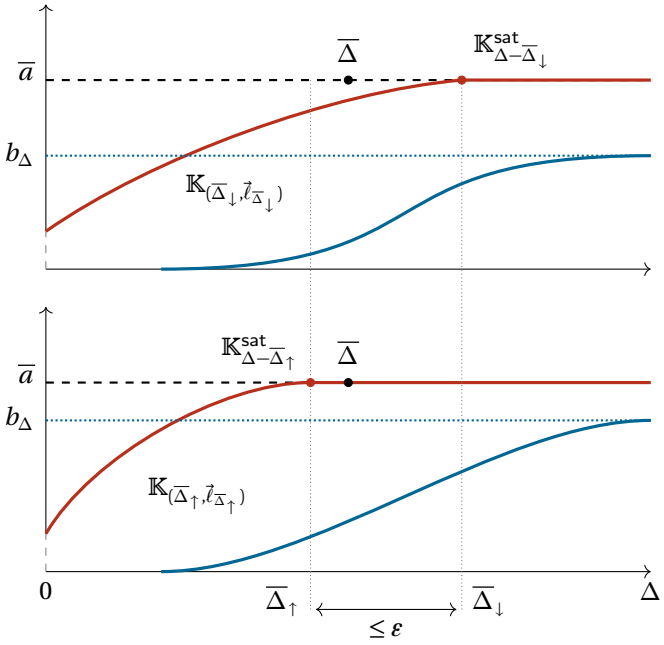


Figure 3.3: An illustration of the $\underline{\overline{\mathbb{K}}}^\uparrow$-operator (top) and the $\underline{\overline{\mathbb{K}}}^\downarrow$-operator (bottom). In the former scenario, the available charge curve (red) hits its saturation limit $\overline{a}$ exactly at $\overline{\Delta}_\downarrow$, i.e. after the actual saturation time point $\overline{\Delta}$, while in the latter scenario the saturation point is under-approximated to $\overline{\Delta}_\uparrow$. The bound charge curve (blue) in the $\underline{\overline{\mathbb{K}}}^\uparrow$ case is consistently below the bound charge curve in the $\underline{\overline{\mathbb{K}}}^\downarrow$ scenario.

Finally, we define the following operators that approximate $\underline{\overline{\mathbb{K}}}$.

Definition 18. We define the operators $\underline{\overline{\mathbb{K}}}^\uparrow, \underline{\overline{\mathbb{K}}}^\downarrow : \mathbb{T} \times \underline{\overline{\mathbb{S}}}_\perp \to \underline{\overline{\mathbb{S}}}_\perp$ by

$$\underline{\overline{\mathbb{K}}}^\downarrow_{(\Delta,\ell)} S := \begin{cases} \mathbb{K}^{\mathsf{sat}}_{\delta_\uparrow} \circ \mathbb{K}_{(\overline{\Delta}_\uparrow,\vec{\ell}_{\overline{\Delta}_\uparrow})} S, & \text{if } \mathbb{K}_{(\Delta,\ell)} S \text{ is over-saturated} \\ \underline{\mathbb{K}}_{(\Delta,\ell)} S & \text{otherwise} \end{cases}$$

and

$$\underline{\overline{\mathbb{K}}}^\uparrow_{(\Delta,\ell)} S := \begin{cases} \mathbb{K}^{\mathsf{sat}}_{\delta_\downarrow} \circ \mathbb{K}_{(\overline{\Delta}_\downarrow,\vec{\ell}_{\overline{\Delta}_\downarrow})} S, & \text{if } \mathbb{K}_{(\Delta,\ell)} S \text{ is over-saturated} \\ \underline{\mathbb{K}}_{(\Delta,\ell)} S & \text{otherwise} \end{cases},$$

where $\delta_\downarrow := \Delta - \overline{\Delta}_\downarrow$ and $\delta_\uparrow := \Delta - \overline{\Delta}_\uparrow$.

Figure 3.3 illustrates how we handle the saturation time point scenario of the $\underline{\overline{\mathbb{K}}}^\downarrow$- and $\underline{\overline{\mathbb{K}}}^\uparrow$-operator.

Indeed, $\underline{\overline{\mathbb{K}}}^\downarrow$ and $\underline{\overline{\mathbb{K}}}^\uparrow$ bound the actual KiBaM SoC evolution in the following sense.

Lemma 16. For any SoC $S \in \underline{\overline{\mathbb{S}}}_{\perp}$ and any task $T \in \mathbb{T}$ we have

$$\overline{\underline{\mathbb{K}}}^{\uparrow}_{T} S \leq \overline{\underline{\mathbb{K}}}_{T} S \leq \overline{\underline{\mathbb{K}}}^{\downarrow}_{T} S.$$

Proof:

We only show $\overline{\underline{\mathbb{K}}}^{\uparrow}_{T} S \leq \overline{\underline{\mathbb{K}}}_{T} S$, since $\overline{\underline{\mathbb{K}}}_{T} S \leq \overline{\underline{\mathbb{K}}}^{\downarrow}_{T} S$ is analogous.
The interesting case is again the case for which $T \coloneqq (\Delta, \ell)$ is a saturating task. Let $\overline{\Delta}$ be the saturation time point. By the correctness of Algorithm 1 (Lemma 10) we have $\overline{\Delta}_{\downarrow} \geq \overline{\Delta}$ and by definition we know that $(\overline{\Delta}_{\downarrow}, \vec{\ell}_{\overline{\Delta}_{\downarrow}})$ is a saturating task. In particular, since $\overline{\Delta}_{\downarrow}$ is a postponed saturation time point we know that $\vec{\ell}_{\overline{\Delta}_{\downarrow}}$ is a charging load of lower magnitude than ℓ itself, i.e. $\ell \leq \vec{\ell}_{\overline{\Delta}_{\downarrow}}$. Hence, Lemma 9 applies and we obtain

$$\overline{\underline{\mathbb{K}}}_{(\Delta, \vec{\ell}_{\overline{\Delta}_{\downarrow}})} S \leq \overline{\underline{\mathbb{K}}}_{(\Delta,\ell)} S$$

and we conclude the proof with the fact that $\overline{\underline{\mathbb{K}}}_{(\Delta, \vec{\ell}_{\overline{\Delta}_{\downarrow}})} S$ is by definition the same as $\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)} S$. ■

Sometimes it is necessary, be it for complexity reasons or ease of implementation, to not run Algorithm 1 which is inherently iterative in nature, but instead rely on easy analytical solutions, at the cost of accuracy.

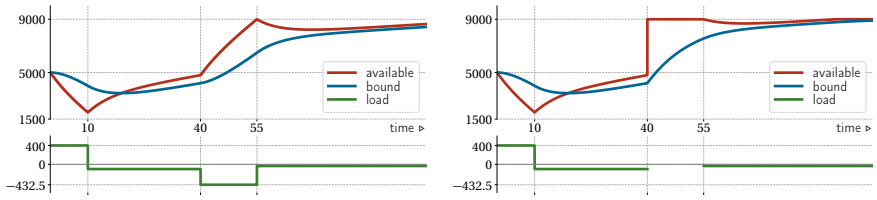
Conceptually, the two scenarios described below are a special case of Algorithm 1 where $\varepsilon \coloneqq \Delta$. In this case, the algorithm terminates immediately without looping, returning the interval $[0, \Delta]$.

Under-approximation. The idea is to postpone the time point $\overline{\Delta}$ to time Δ, i.e. to the very end of the task. We thereby implicitly underestimate the charging load ℓ by $\vec{\ell}_{\Delta}$, which by definition saturates the SoC exactly at the end of the interval. The SoC thus evolves as captured by $\mathbb{K}_{(\Delta, \vec{\ell})}$ for the entire duration of the task.

Over-approximation. Dually, we circumvent the computation of the time point $\overline{\Delta}$ by preponing it to the very beginning of the task at hand, i.e. time 0. We assume that the available charge non-continuously jumps to $\overline{a}$ immediately and attains $\overline{a}$ during the entire task duration. Intuitively, we apply an "infinite" task load. The bound charge evolves throughout the entire task as captured by $\mathbb{K}^{\mathsf{sat}}$, which ignores this task load altogether.

These two concepts, that we are henceforth refering to as *non-iterative*, are later on going to be used to derive analytic ways to express SoC distributions in Section 3.5 as well as to engineer efficient discretization algorithms in Sections 4.2 and 4.3.

Example 5. Let us illustrate the non-iterative approximations on the same situation as in Example 3. For the under-approximation (on the left), in the interval $[40, 55]$, we apply $\vec{\ell} \approx -432.5$ instead of $\ell = -600$ so that the available charge reaches 9000 exactly at $t = 55$. From here on, the SoC is in both components smaller than the SoC from the previous figure.



For the over-approximation (on the right), in the interval $[40,55]$, we intuitively apply a load $\ell \to -\infty$ so that the available charge reaches 9000 exactly at $t = 40$. Since the diffusion is finite, the available charge stays at its limit until $t = 55$ while the bound charge evolves according to $\mathbb{K}^{\mathsf{sat}}$. From this point on, the SoC is larger in both components than the SoC from the previous figure.

3.5 Adding Randomness – the Stochastic KiBaM

In order to consider the KiBaM as a stochastic object, it appears natural to consider the initial SoC $[a_0; b_0]$ as being random. In addition, we consider the load ℓ that is imposed on a battery as being a random quantity as well. The former reflects the real phenomenon of uncertain initial charge levels, rooted in wear and manufacturing variances [5] as well as self discharging rates during a battery's shelf life, while the latter reflects, for example, measurement noise.

We start by formalizing the latter concept as so-called noisy tasks. In this setting, the load on the battery is considered a random variable L, independent of the SoC, distributed as given by an associated probability density function g. We write

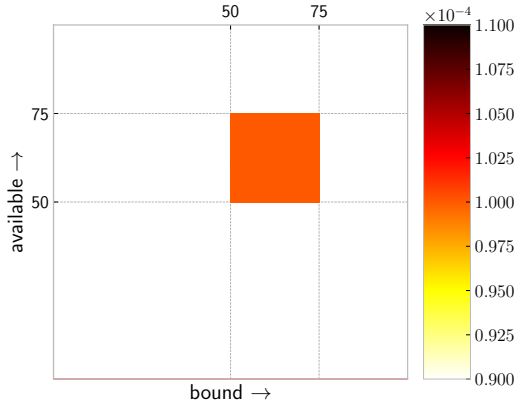
$$L \sim g \quad \text{with} \quad g : \mathbb{R} \to \mathbb{R}.$$

Definition 19 — *Noisy Tasks.* Let L be the random variable representing the random load, and let $L \sim g$. A *noisy task* is a pair (Δ, L), or interchangeably (Δ, g), where $\Delta \in \mathbb{R}_{>0}$ is a strictly positive duration. We denote the set of noisy tasks by $\widetilde{\mathbb{T}}$.

Let us now also consider the initial SoC of a battery as random. For simplicity we first ignore the capacity limits of the battery, i.e. no depletion and no saturation. We will reintegrate these concepts one by one in the upcoming sections. In such a simplified setting, we simply assume the initial SoC to be random variables $[A; B]$ jointly distributed according to a two-dimensional probability density function, i.e.

$$[A; B] \sim f \quad \text{with} \quad f : \mathbb{S} \to \mathbb{R}.$$

Example 6. Instead of a single (Dirac) SoC, we now consider that the joint density f_0 of the charge is, say, uniform over the area $[50, 75] \times [50, 75]$ as depicted below. Here the values of the two-dimensional density are color coded as a heat map.



Using similar plots, we shall illustrate how the SoC distribution evolves as the time passes on this particular example.

We are interested in propagating a random initial SoC $[A; B]$ along a noisy task (Δ, L), with $L \sim g$. This concept is formalized by another pair of random variables,

$[A_\Delta; B_\Delta]$ representing the successor of $[A; B]$, given by

$$[A_\Delta; B_\Delta] \coloneqq \mathbb{K}_{(\Delta,L)}[A_0; B_0].$$

The mathematical basis for expressing the joint density of $[A_\Delta; B_\Delta]$ is the *transformation law for random variables*, which enables the construction of unknown density functions from known ones, given the relation between the corresponding random variables.

Lemma 17 — *Transformation Law for Random Variables.* Let X be a d-dimensional random vector and $h : \mathbb{R}^d \to \mathbb{R}^d$ be an injective and continuously differentiable function. Then we can express the density function of $Y \coloneqq h(X)$ at value y in the range of h as

$$f_Y(y) = f_X\left(h^{-1}(y)\right) \cdot \left|\det\left(J_{h^{-1}}(y)\right)\right|$$

where $\det\left(J_{h^{-1}}(y)\right)$ denotes the *Jacobian determinant* of h^{-1} evaluated at y, i.e.

$$\det \begin{bmatrix} \frac{\partial h_1^{-1}}{\partial x_1}(y) & \dots & \frac{\partial h_1^{-1}}{\partial x_n}(y) \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n^{-1}}{\partial x_1}(y) & \dots & \frac{\partial h_n^{-1}}{\partial x_n}(y) \end{bmatrix}$$

We first notice, the $\mathbb{K}$-operator is not invertible, meaning that, given a successor SoC $[a_\Delta; b_\Delta]$ we are not able to uniquely retrieve the initial SoC $[a; b]$ without knowing the task (Δ, ℓ). We first need to fix the task, i.e. the load and the duration, in order retrieve the initial SoC.

Thus, given the duration (that is deterministic in the first place), we express the joint density of $[A_\Delta; B_\Delta]$, conditioned by the random load L attaining some arbitrary but fixed value ℓ. For this fixed ℓ, the $\mathbb{K}$-operator degenerates to an invertible linear mapping

$$\mathbb{K}_{(\Delta,\ell)} \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} \mathrm{d}_{aa} & \mathrm{d}_{ba} \\ \mathrm{d}_{ab} & \mathrm{d}_{bb} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} + \begin{bmatrix} \mathrm{d}_{a\ell} \\ \mathrm{d}_{b\ell} \end{bmatrix} \cdot \ell.$$

Lemma 18 — *Inverse of* $\mathbb{K}$. Given a task $(\Delta, \ell) \in \mathbb{T}$ the inverse of $\mathbb{K}$ is given by

$$\mathbb{K}^{-1}_{(\Delta,\ell)} \begin{bmatrix} a \\ b \end{bmatrix} = \mathrm{e}^{k\Delta} \begin{bmatrix} \mathrm{d}_{bb} & -\mathrm{d}_{ab} & \mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{bb}\mathrm{d}_{a\ell} \\ -\mathrm{d}_{ba} & \mathrm{d}_{aa} & \mathrm{d}_{ba}\mathrm{d}_{a\ell} - \mathrm{d}_{aa}\mathrm{d}_{b\ell} \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ \ell \end{bmatrix}.$$

Proof:

By straight-forward inversion. ■

In the saturation and depletion agnostic scenario, we are able to express the joint density of $[A_\Delta; B_\Delta]$ conditioned by $L = \ell$ for some fixed ℓ.

Lemma 19. Let $(\Delta, g) \in \widetilde{\mathbb{T}}$ be a noisy task, $[A; B]$ and L be random variables distributed as follows

$$[A; B] \sim f \quad \text{and} \quad L \sim g.$$

Then, the joint density of $[A_\Delta; B_\Delta]$ conditioned by $L = \ell$ is given by

$$f_\Delta(a, b \mid \ell) = f\left(\mathbb{K}^{-1}_{(\Delta,\ell)}[a; b]\right) \cdot \left|\mathrm{e}^{k\Delta}\right|.$$

Proof:

By application of the transformation law of random variables (Lemma 17). Here, $\mathrm{e}^{k\Delta}$ is the determinant of the Jacobian of $\mathbb{K}^{-1}_{(\Delta,\ell)}$. Interestingly, it is constant in a, b and ℓ, it only depends on Δ. It is also non-negative since $\Delta \geq 0$ and $k > 0$. ■

Finally, we get rid of the conditional $L = \ell$ by marginalizing the variable $[A_\Delta; B_\Delta]$. Intuitively, marginalization is a continuous weighted average of the conditional density values with weights being the values of g. We formalize the result with the following Lemma.

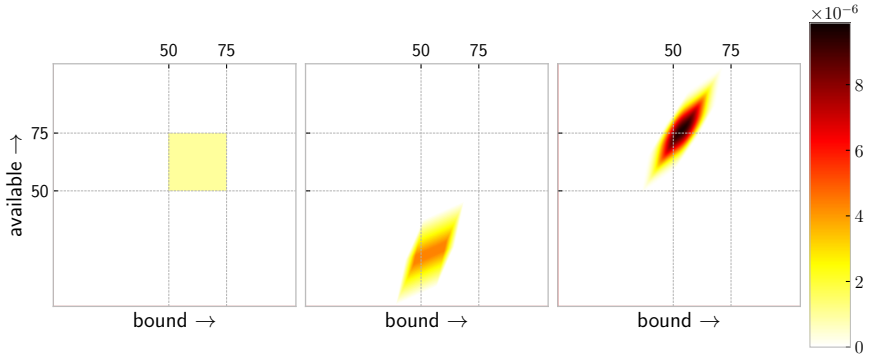
Lemma 20. Let $(\Delta, g) \in \widetilde{\mathbb{T}}$ be a noisy task and $[A_0; B_0]$ be a random SoC distributed according to f. The joint distribution of $[A_\Delta; B_\Delta]$ is absolutely continuous with density f_Δ given by

$$f_\Delta(a, b) = \mathrm{e}^{k\Delta} \int_{\mathrm{supp}(g)} f\left(\mathbb{K}^{-1}_{(\Delta,\ell)}[a; b]\right) \cdot g(\ell)\, \mathrm{d}\ell.$$

Proof:

By marginalizing L away, we obtain the integral over an expression that is derived from Lemma 19. Since the determinant of the Jacobian of $\mathbb{K}^{-1}_{(\Delta,\ell)}$, namely $\mathrm{e}^{k\Delta}$, is independent of the integration variable ℓ, we can move this factor in front of the integral. ■

Example 7. We show how to propagate a SoC distribution along tasks by picking up Example 6. The inital battery SoC (left) is uniformly distributed on $[50, 75] \times [50, 75]$. We strain the battery with the noisy tasks $T_1 \coloneqq (16, \ell_1)$ (middle) and then with $T_2 \coloneqq (13.2, \ell_2)$ (right), where $\ell_1 \sim \mathcal{N}(3, 1.4)$ and $\ell_2 \sim \mathcal{N}(-4, 1.4)$.



We now tackle the challenge to enrich the development above with depletion and saturation. We thus assume that the random variables $[A_\Delta; B_\Delta]$ evolve according to the $\overline{\underline{\mathbb{K}}}$-operator developed in Section 3.3, i.e. by overloading notation we investigate random successor SoCs given by

$$[A_\Delta; B_\Delta] \coloneqq \overline{\underline{\mathbb{K}}}_{(\Delta,\ell)}[A_0; B_0].$$

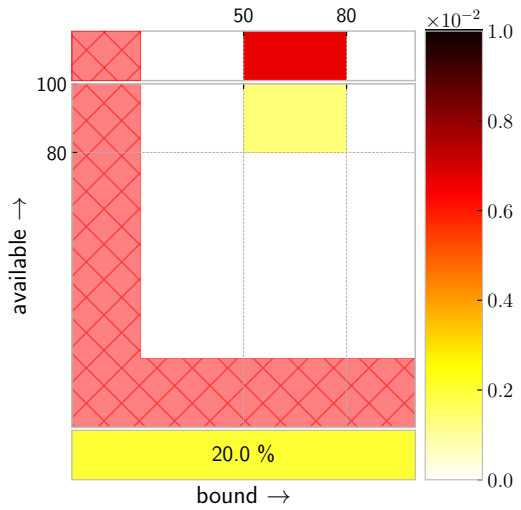
We first observe that the joint distribution of $[A_\Delta; B_\Delta]$ may not be absolutely continuous, because positive probability may accumulate in the canonical depletion SoC $\bot$ and on the set of safe saturated SoCs $\overline{\mathbb{S}}$, suggesting a need for a more complex representation of SoC distribution with three separate parts.

Definition 20 — *SoC distribution.* A *SoC distribution* is a triple $\langle \bar{f}, f, z \rangle$ where

- $\bar{f}$ is a density over $\overline{\mathbb{S}}$, (bound charge distribution of the saturated portion)
- f is a joint density over $\overline{\underline{\mathbb{S}}} \setminus \overline{\mathbb{S}}$, (the distribution of the non-saturated portion)
- $z \in [0,1]$. (the cumulative probability of depletion)

Example 8. Instead of a single SoC, we now consider a SoC distribution $\langle \bar{f}, f, z \rangle$ of the charge. We visualize these distributions as three stacked heatmaps, each assiociated with one component of the SoC distribution, in addition to a colorbar.

- The heatmap on the very top depicts $\bar{f}$, meaning the one-dimensional distribution of the bound charge with the available charge being at the saturation limit $\overline{a}$.
- The heatmap in the middle represents the two-dimensional density of the non-saturated safe portion f.
- The heatmap on the bottom is essentially just a color-coded probability value, namely the accumulated depletion risk z.



In the depiction above, we observe the following illustrating scenario:

- A probability mass of 0.2 is concentrated in $\bar{f}$, in the form of a uniform distribution over the interval $[50,80]$, thus $\bar{f}(b) = \frac{0.2}{80-50} = 0.00\overline{6}$ for each $b \in [50,80]$.
- The unsaturated safe part in f accounts for a factor of 0.6 of the total probability mass, also in the form of a uniform distribution over the area $[50,80] \times [80,100]$, with each $[a;b]$ in this area, carrying a density value of $\frac{0.6}{(80-50)\cdot(100-80)} = 0,001$.
- The remaining 0.2 of the entire probability mass is already considered depleted and thus represented in the bottom part. The color coding of this last part considers the maximum density value of the colorbar to be 1, with the lowest being 0, and is thus on a different scale.

Lastly, the red checkered area represents the area of unsafe SoCs, induced here by $\underline{a} = \underline{b} = 20$.

To argue about probabilities with respect to SoC distributions, we define what it means that a pair of random variables is distributed according to a SoC distribution.

Definition 21. We say that a pair of random variables $[A;B]$ is distributed according to a SoC distribution $\langle \bar{f}, f, z \rangle$, and write

$$[A;B] \sim \langle \bar{f}, f, z \rangle,$$

if for any measurable set $X \subseteq \mathbb{S} \times \mathbb{S}$ we have

$$\mathbf{Pr}[S \in X] = \iint_{[a;b] \in X} f(a,b)\,\mathrm{d}a\,\mathrm{d}b \;+\; \int_{[\overline{a};\overline{b}] \in X} \bar{f}(b)\,\mathrm{d}b \;+\; z\mathbb{I}_{\perp \in X}$$

where $\mathbb{I}_{\varphi}$ denotes the indicator function of a condition φ.

We fix some notation and basic concepts of SoC distributions.

Definition 22. Following convention, we interpret arithmetic or comparison operations on SoCs to be componentwise, hence for $\star \in \{+,-,\cdot,/\}$ we have

$$\langle \bar{f}_1, f_1, z_1\rangle \star x := \begin{cases} \langle \bar{f}_1 \star \bar{f}_2, f_1 \star f_2, z_1 \star z_2\rangle & \text{if } x = \langle \bar{f}_2, f_2, z_2\rangle \\ \langle \bar{f}_1 \star \mathrm{k}, f_1 \star \mathrm{k}, z_1 \star \mathrm{k}\rangle & \text{if } x = \mathrm{k} \in \mathbb{R} \end{cases}.$$

In addition, we use the shorthand notation $\langle \bar{f}, f, z\rangle_x$ for $\langle \bar{f}_x, f_x, z_x\rangle$.

We assume a random load L distributed according to a probability density function g. For a random initial SoC $[A_0; B_0]$ distributed according to a SoC distribution $\langle \bar{f}, f, z\rangle_0$ and a given noisy task $T := (\Delta, L)$ we aim at expressing the resulting SoC $[A_\Delta; B_\Delta]$ using a SoC distribution $\langle \bar{f}, f, z\rangle_\Delta$. To be able to express the distribution as integrals over simple closed-form expressions, we resort to under- and over-approximations of the SoC.

We will work with $\langle \bar{f}, f, z\rangle^\uparrow$ and $\langle \bar{f}, f, z\rangle^\downarrow$ as notations for upper, respectively lower bounding SoC distributions.

To arrive there, we define

$$[A^\uparrow_\Delta; B^\uparrow_\Delta] := \overline{\mathbb{K}}^\uparrow_{(\Delta,L)}[A_0; B_0] \quad \text{and} \quad [A^\downarrow_\Delta; B^\downarrow_\Delta] := \overline{\mathbb{K}}^\downarrow_{(\Delta,L)}[A_0; B_0]$$

respectively. We will eventually show that the above SoCs under-approximate and over-approximate the actual successor SoC $[A_\Delta; B_\Delta]$ in the following sense.

Definition 23. Let S_1 and S_2 be two random SoCs. We say that S_1 *under-approximates* S_2 *at the saturation limit* if

$$\begin{aligned} \mathbf{Pr}\big[S_1 \le S\big] &= \mathbf{Pr}\big[S_2 \le S\big] && \text{for any } S < [\overline{a}; \overline{b}], \\ \mathbf{Pr}\big[S_1 \le [\overline{a}; b]\big] &\le \mathbf{Pr}\big[S_2 \le [\overline{a}; b]\big] && \text{for any } \underline{b} \le b \le \overline{b}. \end{aligned}$$

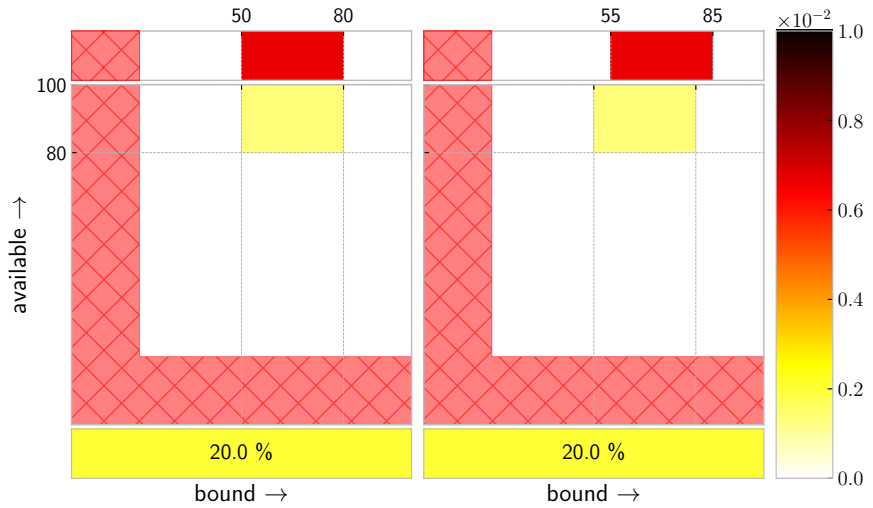
Analogously, S_1 *over-approximates* S_2 *at the saturation limit* if

$$\begin{aligned} \mathbf{Pr}\big[S_1 \le S\big] &= \mathbf{Pr}[S_2 \le S] && \text{for any } S < [\overline{a}; \overline{b}], \\ \mathbf{Pr}\big[S_1 \le [\overline{a}; b]\big] &\ge \mathbf{Pr}[S_2 \le [\overline{a}; b]] && \text{for any } \underline{b} \le b \le \overline{b}. \end{aligned}$$

Example 9. In this example, we illustrate Definition 23. Let us reconsider a random SoC $S_1 \sim \langle \bar{f}, f, z\rangle_1$ as in the previous example (Example 8), that is essentially composed of uniform distributions (left) and $S_2 \sim \langle \bar{f}, f, z\rangle_2$, with

$$\bar{f}_2(b+5) = \bar{f}_1(b), \quad f_2 = f_1 \quad \text{and } z_2 = z_1.$$

$\bar{f}_2$ is thus a shifted version of $\bar{f}_1$ (right).



It is relatively easy to see that S_1 under-approximates S_2 at the saturation limit. Firstly, we notice that their unsaturated parts are identical. Secondly, we have, due to the shifted nature of $\bar{f}_2$, that

$$\int_{50}^{b} \bar{f}_1(x)\,\mathrm{d}x \geq \int_{50}^{b} \bar{f}_2(x)\,\mathrm{d}x$$

for each $\underline{b} \leq b \leq \overline{b}$.

This approach, detailed in the next section, provides upper and lower bounds on the risk of battery depletion due to the monotonicity established in Lemma 16.

In the upcoming sections, we derive analytical expressions that under- and over-approximate the successor SoC distribution of a random SoC with respect to a noisy task.

Behavior below the saturation limit.

Assume we have a random SoC $[A;B]$ that is distributed according to $\langle \bar{f}, f, z\rangle$, and a noisy task $T \coloneqq (\Delta, g)$. We first define a joint density $\underline{f}_\Delta$ over $]{-}\infty, \overline{a}[\times]{-}\infty, \overline{b}[$ that exactly describes the $\mathbb{K}$-behavior below the saturation limit while pretending the battery is unable to deplete, i.e. we pretend for the moment that there are no unsafe SoCs induced by $[\underline{a};\underline{b}]$. From this we will derive f_Δ as well as z_Δ, essentially by cropping and accumulating the probability mass that ends up in the unsafe SoC space after powering the task T. For this part, we don't yet require approximations.

We use the transformation law of random variables (Lemma 17) to derive an expression for $\underline{f}_\Delta$.

The intricate part in expressing $\underline{f}_\Delta$ is to describe how the SoC evolves when it leaves the saturated part (a one dimensional interval) captured by $\bar{f}$ into the area below the saturation limit (a two-dimensional area) when the level of available charge is decreasing. For each unsaturated SoC $[a';b']$ we need to find out what saturated SoC of the form $[\overline{a};b]$ under what load ℓ (such that $\ell > \ell_{\mathsf{sat}}(b)$) would

evolve in time Δ exactly into $[a';b']$, i.e.

$$\mathbb{K}^{-1}_{(\Delta,\ell)}[a';b'] = [\overline{a};b].$$

By Definition 16, this is the case when using the inverse target load $\overleftarrow{\ell}_\Delta[a';b']$ which results in a bound charge $\overleftarrow{\mathrm{B}}_\Delta[a';b']$. Based on the above, we define the map

$$[a;b] \mapsto \big(\overleftarrow{\mathrm{B}}_\Delta[a;b], \overleftarrow{\ell}_\Delta[a;b]\big)$$

to apply the transformation law. This mapping is injective, since $\overleftarrow{\ell}_\Delta$ is unique, hence $\overleftarrow{\mathrm{B}}_\Delta$ is also unique. Its Jacobian determinant is easily derived to be $1/(\mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{a\ell}\mathrm{d}_{bb})$ and is constant in the SoC and the load.

Finally, putting all of the above together, we can express the joint density $\underline{f}_\Delta$ for any $a < \overline{a}$ and $b < \overline{b}$ as

$$\begin{aligned}\underline{f}_\Delta(a,b) = \bar{f}\big(\overleftarrow{\mathrm{B}}_\Delta[a;b]\big) \cdot \frac{1}{|\mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{a\ell}\mathrm{d}_{bb}|} \cdot g\big(\overleftarrow{\ell}_\Delta[a;b]\big) \\ + \mathrm{e}^{k\Delta} \int_{-\infty}^{\overleftarrow{\ell}_\Delta[a;b]} f\Big(\mathbb{K}^{-1}_{(\Delta,\ell)}[a;b]\Big) \cdot g(\ell)\,\mathrm{d}\ell.\end{aligned}$$

Let us briefly go through the individual derivations of each summand. The first summand comes from $\bar{f}$, due to the saturated SoCs leaving the saturated area, as discussed just above. It is pretty much a direct application of the transformation law, where we need only consider one load value, instead of the whole support of g.

The second summand is (almost) a direct application of the transformation law. It reflects those SoCs stemming from the density of the unsaturated area f through the standard limit-agnostic $\mathbb{K}$-operator, and that end up in, say SoC S. Ranging over all loads ℓ, it integrates the density f of such SoCs $[a_\ell;b_\ell]$ that satisfy $\mathbb{K}_{(\Delta,\ell)}[a_\ell;b_\ell] = S$, i.e. $[a_\ell;b_\ell] = \mathbb{K}^{-1}_{(\Delta,\ell)}S$. Lemma 4 again guarantees that no limits are crossed in the meantime.

Finally, we state how to derive the actual parts of the successor SoC distribution we are interested in.

Lemma 21 — *Successor SoC distribution (Part 1).* Let $(\Delta, g) \in \widetilde{\mathbb{T}}$ be a noisy task and $[A;B]$ be a random SoC distributed according to $\langle \bar{f}, f, z\rangle$. Then for the successor SoC

$$[A_\Delta;B_\Delta] \coloneqq \overline{\mathbb{K}}_{(\Delta,g)}[A;B]$$

we have

$$[A_\Delta;B_\Delta] \sim \langle \bar{f}_\Delta, f_\Delta, z_\Delta\rangle$$

for some $\bar{f}_\Delta$, where f_Δ and z_Δ are given by

$$f_\Delta(a,b) \coloneqq \underline{f}_\Delta(a,b) \qquad \text{for } [a;b] \in \overline{\mathbb{S}},$$

$$z_\Delta \coloneqq \int_{-\infty}^{\overline{a}} \int_{-\infty}^{\overline{b}} \underline{f}_\Delta(a,b)\,\mathrm{d}a\,\mathrm{d}b.$$

Proof:

Since we used the transformation law of random variables (Lemma 17), the density $\underline{f}_\Delta$ captures the $\mathbb{K}$-behavior correctly while considering the unsafe SoC region to be safe.
Accumulating the density in the unsafe area $\left]-\infty, \underline{a}\right] \times \left]-\infty, \overline{b}\right]$ via integration leads to the correct depletion risk after powering a task. We justify this using Lemma 3, which establishes that the available charge always reaches the depletion area before the bound charge does.
Cropping $\underline{f}_\Delta$ to only the safe and non-saturated SoCs quite obviously results in the correct subdistribution f.
Note that the qualification (for f) and the integration area (for z) are disjoint. ■

Behavior On The Saturation Limit

As indicated in Definition 23, we resort to approximations of the charge in order to define the subdistribution of the saturated area, more explicitly the non-iterative version.

Under-approximation. We define the under-approximation of the density for $0 \leq b \leq \overline{b}$ by

$$\begin{aligned} \bar{f}^{\uparrow}_{\Delta}(b) &= \bar{f}_0\big(b^{-1}_{\mathsf{sat}}\big) \cdot G\big(\ell_{\mathsf{sat}}(b^{-1}_{\mathsf{sat}})\big) \cdot \mathrm{e}^{c k \Delta} && (3.7) \\ &+ \bar{f}_0\big(\mathrm{B}^{\leftarrow}_{\overline{a}}\big) \cdot \big[G\big(\ell^{\leftarrow}_{\overline{a}}\big) - G\big(\ell_{\mathsf{sat}}(\mathrm{B}^{\leftarrow}_{\overline{a}})\big)\big] \cdot \left| \frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{a\ell}\mathrm{d}_{bb}} \right| && (3.8) \\ &+ \int_{-\infty}^{\infty} f_0\big(a, \mathrm{B}^{\leftarrow}_{a}\big) \cdot G\big(\ell^{\leftarrow}_{a}\big)\, \mathrm{d}a \cdot \left| \frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell} - \mathrm{d}_{a\ell}\mathrm{d}_{bb}} \right| && (3.9) \end{aligned}$$

Let us go through this expression line by line.

We first consider the portion of the density that stays saturated by the fact that the load overpowers the diffusion and thus behaves according to $\mathbb{K}^{\mathsf{sat}}$. More precisely, given a task $T \coloneqq (\Delta, \ell)$ and a bound charge level b, we define the mapping based on the inverse of $\mathbb{K}^{\mathsf{sat}}$ (or $\mathrm{B}^{\mathsf{sat}}$), i.e. find b^{-1}_{sat} such that $\mathbb{K}^{\mathsf{sat}}_{\Delta}[\overline{a}; b^{-1}_{\mathsf{sat}}] = [a; b]$. A straight forward inversion leads to

$$b^{-1}_{\mathsf{sat}} = \mathrm{e}^{c k \Delta} \cdot b - (\mathrm{e}^{c k \Delta} + 1) \cdot \overline{b}$$

Thus, in order to use the transformation law, we define the mapping

$$b \mapsto b^{-1}_{\mathsf{sat}}$$

whose determinant is given by $\mathrm{e}^{c k \Delta}$.

However, this expression is taken into account only for charging currents that cover the diffusion (i.e. $\ell \leq \ell_{\mathsf{sat}}(b^{-1}_{\mathsf{sat}})$) so that the battery evolves along the saturation limit as expressed by Lemma 6. The integration over this range of loads can be directly expressed using the *cumulative density function (CDF) G* of the load, since by definition of CDFs, we have

$$G(x) = \int_{-\infty}^{x} g(y)\, \mathrm{d}y.$$

Next, we address the case where the diffusion in the state $[\overline{a};b]$ is stronger than the charging load. The SoC thus leaves the saturated area in the beginning, but potentially returns. Let us assume that before time Δ the SoC $[\overline{a};b]$ returns to being saturated, culminating in some state $[\overline{a};b']$. We are not able to express b using a closed-form expression over b' as discussed in Section 3.3 and need to rely on approximations. We thus under-approximate the bound charge by postponing the saturation time point to exactly time Δ by charging the battery with $\vec{\ell}_\Delta$ instead, just as shown in Example 5. Let us shortly outline the derivation. The mapping for the transformation law is

$$b \mapsto \vec{\mathrm{B}}_\Delta[\overline{a};b].$$

Its inverse is simply $b \mapsto \overleftarrow{\mathrm{B}}_\Delta[\overline{a};b]$ with Jacobian determinant $-\mathrm{d}_{a\ell}/(\mathrm{d}_{ab}\mathrm{d}_{b\ell}-\mathrm{d}_{a\ell}\mathrm{d}_{bb})$.

The transformation law yields a density at time Δ of

$$\bar{f}_0\big(\mathrm{B}^{\leftarrow}_{\overline{a}}[\overline{a};b]\big)\cdot\left|\frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell}-\mathrm{d}_{a\ell}\mathrm{d}_{bb}}\right|.$$

Then, we integrate over all charging loads ℓ that are powerful enough to saturate the SoC, i.e. $\ell \le \ell^{\leftarrow}_{\overline{a}}[\overline{a};b]$, yet not too powerful to leave the saturation area in the meantime, i.e. $\ell > \ell_{\mathsf{sat}}(\mathrm{B}^{\leftarrow}_{\overline{a}}[\overline{a};b])$. The integral over the resulting range equals $G\big(\ell^{\leftarrow}_{\overline{a}}\big)-G\big(\ell_{\mathsf{sat}}(\mathrm{B}^{\leftarrow}_{\overline{a}})\big)$.

The third summand (3.9) comes from the density f_0 of the unsaturated area and under-approximates the bound charge similarly to the second summand. If the available charge of the battery reaches its saturation limit *before* time Δ, we postpone it to instead reaching it exactly at time Δ by underestimating the charging current with $\ell^{\rightarrow}_{\overline{a}}$. For the derivation, we define a map $\mathsf{K}_\Delta : [a;b;\ell] \mapsto [a;\mathrm{B}^{\rightarrow}_{\overline{a}}[a;b];\ell]$ (it is an identity in the first and the third component to make it injective) and apply the transformation law of random variables. The inverse K^{-1}_Δ and its Jacobian determinant is

$$\mathsf{K}^{-1}_\Delta : (a,b,\ell) \mapsto \big(a, \mathrm{B}^{\leftarrow}_{a}[\overline{a};b],\ell\big) \quad \text{and} \quad \det J_{\mathsf{K}^{-1}_\Delta} = \frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell}-\mathrm{d}_{a\ell}\mathrm{d}_{bb}}.$$

The density h_Δ over the co-domain of K_Δ is obtained by the transformation law as

$$\begin{aligned} h_\Delta[a;b;\ell] &= h_0\big(\mathsf{K}^{-1}_\Delta[a;b;\ell]\big)\cdot\left|\frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell}-\mathrm{d}_{a\ell}\mathrm{d}_{bb}}\right| \\ &= f_0\big(a,\mathrm{B}^{\leftarrow}_{a}[\overline{a};b]\big)\cdot g(\ell)\cdot\left|\frac{-\mathrm{d}_{a\ell}}{\mathrm{d}_{ab}\mathrm{d}_{b\ell}-\mathrm{d}_{a\ell}\mathrm{d}_{bb}}\right| \end{aligned}$$

where the density h_0 equals a product of the densities f_0 and g because of independence of SoC $[a;b]$ and load ℓ. Marginalizing away a and ℓ (using $G\big(\ell^{\leftarrow}_{a}\big)$ to integrate over all currents necessary to reach saturation) gives us the subdensity from the third summand.

Over-approximation. The over-approximation is based on similar reasoning and equals:

$$\bar{f}^{\downarrow}_\Delta(b) = \bar{f}_0\big(b^{-1}_{\mathsf{sat}}\big)\cdot G\big(\ell^{\rightarrow}_{\overline{a}}\big)\cdot \mathrm{e}^{ck\Delta} + \int_{-\infty}^{\infty} f_0\big(a,b^{-1}_{\mathsf{sat}}\big)\cdot G\big(\ell^{\rightarrow}_{\overline{a}}\big)\,\mathrm{d}a\cdot\mathrm{e}^{ck\Delta} \tag{3.10}$$

The first summand in Equation (3.10) treats the contribution of the density $\bar{f}_0$ from the point b_{sat}^{-1} as defined above. We assume the density evolves as indicated by $\mathbb{K}^{\mathsf{sat}}$ whenever $\mathbb{K}$ would result in $a_\Delta \geq \overline{a}$ (i.e. if the load is stronger than $\ell_{\overrightarrow{a}}[\overline{a}; b_{\mathsf{sat}}^{-1}]$). This is an over-approximation for exactly those charging loads ℓ satisfying $\ell_{\mathsf{sat}}(b_{\mathsf{sat}}^{-1}) < \ell < \ell_{\overrightarrow{a}}[\overline{a}; b_{\mathsf{sat}}^{-1}]$, i.e. for charging loads that are not strong enough to keep the battery saturated at $\overline{a}$ throughout the entire task but that are strong enough to eventually re-saturate the battery.

The second summand in (3.10) comes from the density f_0 of the non-saturated area. Again, whenever $\mathbb{K}$ would result in $a_\Delta > \overline{a}$ (i.e. when the charging load is stronger in magnitude than $\ell_{\overrightarrow{a}}[a; b_{\mathsf{sat}}^{-1}]$), we assume that the saturation limit is reached immediately and further evolves as given by $\mathbb{K}^{\mathsf{sat}}$, thus justifying the argument $\ell_{\overrightarrow{a}}$ to appear in the integral. This results in an over-approximation for any such SoC.

To briefly clarify the derivation, for both summands in principle the mapping $(a,b,\ell) \mapsto \left(a, \mathrm{B}_\Delta^{\mathsf{sat}}(b), \ell\right)$ is used. For the first summand, think of a as not being a variable, but the constant $\overline{a}$. For the second summand, we again use the transformation law, and afterwards marginalize away variables a and ℓ, where we hide integration over the corresponding range using the cdf G. The entire derivation is very similar to the one of the under-approximation before.

We finally obtain the following result.

Lemma 22 — *Successor SoC distribution (Part 2).* Let $(\Delta, g) \in \widetilde{\mathbb{T}}$ be a noisy task, and $[A;B]$ be a random SoC distributed according to $\langle \bar{f}, f, z\rangle$. For the approximations of the successor SoC given by

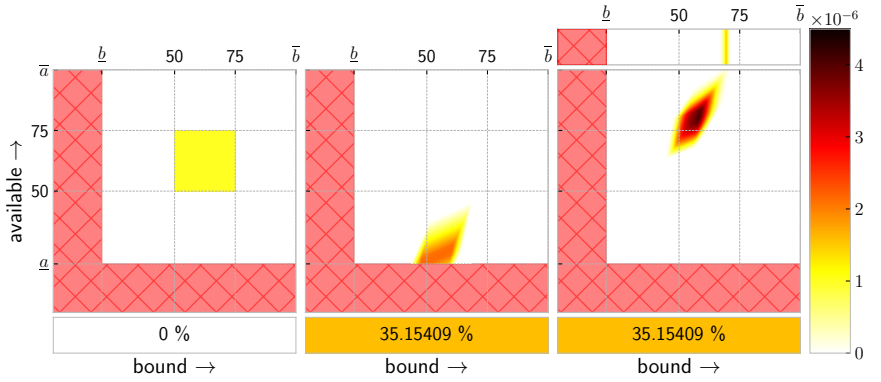
$$[A_\Delta^\uparrow; B_\Delta^\uparrow] \coloneqq \overline{\underline{\mathbb{K}}}_{(\Delta,g)}^\uparrow[A;B] \quad \text{and} \quad [A_\Delta^\downarrow; B_\Delta^\downarrow] \coloneqq \overline{\underline{\mathbb{K}}}_{(\Delta,g)}^\downarrow[A;B]$$

with

$$[A_\Delta^\uparrow; B_\Delta^\uparrow] \sim \langle \bar{f}_\Delta^\uparrow, f_\Delta, z_\Delta\rangle \quad \text{and} \quad [A_\Delta^\downarrow; B_\Delta^\downarrow] \sim \langle \bar{f}_\Delta^\downarrow, f_\Delta, z_\Delta\rangle$$

we indeed have that $[A_\Delta^\uparrow; B_\Delta^\uparrow]$ and $[A_\Delta^\downarrow; B_\Delta^\downarrow]$ under- and over-approximate $[A_\Delta; B_\Delta]$, respectively, at the saturation limit.
Here, $\bar{f}_\Delta^\uparrow$ and $\bar{f}_\Delta^\downarrow$ come from Equations 3.7 and 3.10, respectively.

Example 10. For illustrative purposes we reiterate on Example 7. Based on Lemma 22, we can under-approximate the SoC of the random battery from our second running example for battery limits $\underline{a} = \underline{b} = 25, \overline{a} = \overline{b} = 100$. The initial battery SoC (left) is uniformly distributed on $[50,75] \times [50,75]$. We strain the battery with the noisy tasks $T_1 \coloneqq (16, \ell_1)$ (middle) and then with $T_2 \coloneqq (13.2, \ell_2)$ (right), where $\ell_1 \sim \mathcal{N}(3, 1.4)$ and $\ell_2 \sim \mathcal{N}(-4, 1.4)$.



Lemma 23 — *Probability of powering a task.* A battery with SoC distribution $\langle \bar{f}, f, z \rangle_0$ $\overline{\underline{\mathbb{K}}}$-powers a task (Δ, g) with probability $p > 0$ if and only if $z_\Delta < p$.

3.6 Markov Task Processes

So far, we have only discussed execution of one task with fixed duration and random load. In this section, we give a discrete-time Markov model that generates noisy tasks that we call a *Markov Task Process* (MTP). The formalism is closely inspired by stochastic task graph models [34] or data-flow formalisms such as SDF [25] or SADF [39]. In SDF, task durations are deterministic, and thus directly supported in our framework. In SADF, durations are in general governed by discrete probability distributions, which can be translated into our framework at the price of a larger state space. We will briefly explain informally how to translate timed versions of SDF as well as SADF to MTPs, after a formal introduction of the latter.

Definition 24 — *Markov Task Process (MTP).* A *Markov Task Process (MTP)* is a tuple $\boldsymbol{M} = (\mathfrak{S}, P, \pi, \mathsf{task})$ where $\mathfrak{S}$ is a finite set of states, $P : \mathfrak{S} \times \mathfrak{S} \to [0,1]$ is a transition probability matrix, π is an initial probability distribution over $\mathfrak{S}$ and $\mathsf{task} : \mathfrak{S} \to \widetilde{\mathbb{T}}$ assigns a noisy task to each state.

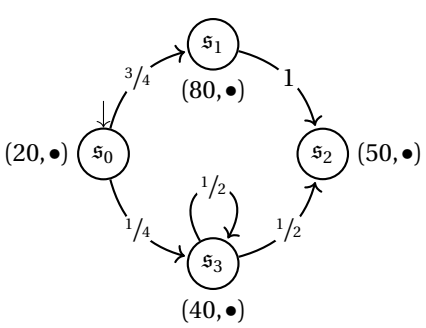
An example of an MTP is depicted in Figure 3.4.

Intuitively, a Markov Task Process $\boldsymbol{M}$ together with an initial distribution of the SoC given by $\langle \bar{f}, f, z \rangle$ behaves as follows.

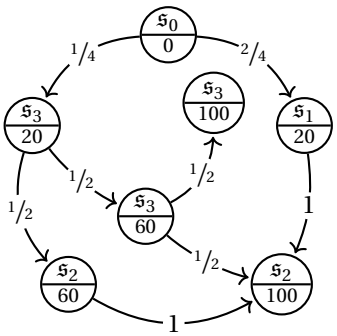
Initialization: An initial safe SoC $[a; b] \in \overline{\underline{\mathbb{S}}}$ and an initial MTP state $\mathfrak{s}_0 \in \mathfrak{S}$ are chosen independently at random according to $\langle \bar{f}, f, z \rangle$, and π, respectively.

Repeat:

- A task $T = (\Delta, \ell)$ to be powered in state $\mathfrak{s}_0$ is picked randomly according to $\ell \sim g$, where $\mathsf{task}(\mathfrak{s}_0) = (\Delta, g)$ is the assigned noisy task.
- The battery is strained with T leading to a successor SoC $\overline{\underline{\mathbb{K}}}_T[a_0; b_0]$.
- A successor MTP state $\mathfrak{s}_1 \in \mathfrak{S}$ is chosen at random with probability $P(\mathfrak{s}_0, \mathfrak{s}_1)$.



A Markov Task Process annotated with (potentially noisy) tasks.



The MTP's induced graph unfolded up to a time horizon $\bar{t} = 100$.

Figure 3.4: An MTP model annotated with (potentially noisy) tasks $\mathsf{task}(\mathfrak{s}) = (\Delta, \bullet)$ and its induced graph for a time horizon $\bar{t} = 100$ with the top label indicating the MTP state and the bottom label indicating the current time.

- Repeat with $\mathfrak{s}_0 \coloneqq \mathfrak{s}_1$.

Formally, $\boldsymbol{M}$ and $\langle \bar{f}, f, z \rangle_0$ induce a probability measure **Pr** over samples of the form $\omega = [[a_0; b_0] \,|\, (\mathfrak{s}_k)_{k=0}^{\infty}]$ where the first component is the initial SoC of the battery and the second component describes an infinite execution of $\boldsymbol{M}$.

For a given $\bar{t} \in \mathbb{R}_{\geq 0}$, the SoC of the battery at time $\bar{t}$ is expressed by random SoC $[A_{\bar{t}}; B_{\bar{t}}]$ that is for any $\omega = [[a_0; b_0] \,|\, (\mathfrak{s}_k)_{k=0}^{\infty}]$ defined as

$$\begin{bmatrix} A_{\bar{t}}(\omega) \\ B_{\bar{t}}(\omega) \end{bmatrix} \coloneqq \underline{\overline{\mathbb{K}}}_{(\Delta, g_n)} \circ \underline{\overline{\mathbb{K}}}_{\mathsf{task}(\mathfrak{s}_{n-1})} \circ \cdots \circ \underline{\overline{\mathbb{K}}}_{\mathsf{task}(\mathfrak{s}_0)} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}$$

where n is the minimal number such that the n-th task is not finished before $\bar{t}$, i.e. $n \coloneqq \min\left\{ n \;\middle|\; \bar{t} \leq \sum_{j=0}^{n} \Delta_{\mathfrak{s}_j} \right\}$, $\Delta \coloneqq \bar{t} - \sum_{j=0}^{n-1} \Delta_{\mathfrak{s}_j}$ and g_n is the load density function associated with $\mathsf{task}(\mathfrak{s}_n)$.

Definition 25. We say that a battery with a SoC $\langle \bar{f}, f, z \rangle_0$ *powers with probability* $p > 0$ a system $\boldsymbol{M}$ for time $\bar{t}$ if

$$\mathbf{Pr}\big[A_{\bar{t}} > \underline{a}\big] \geq p.$$

In order to under-approximate the probability that $\boldsymbol{M}$ is powered for a given time, we need to symbolically express the distribution of

$$\begin{bmatrix} A^{\uparrow}_{\bar{t}}(\omega) \\ B^{\uparrow}_{\bar{t}}(\omega) \end{bmatrix} \coloneqq \underline{\overline{\mathbb{K}}}^{\uparrow}_{(\Delta, g_n)} \circ \underline{\overline{\mathbb{K}}}^{\uparrow}_{\mathsf{task}(\mathfrak{s}_{n-1})} \circ \cdots \circ \underline{\overline{\mathbb{K}}}^{\uparrow}_{\mathsf{task}(\mathfrak{s}_0)} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}$$

where we just replace $\underline{\overline{\mathbb{K}}}$ with $\underline{\overline{\mathbb{K}}}^{\uparrow}$. Analogously, for an over-approximation we use $\underline{\overline{\mathbb{K}}}^{\downarrow}$ instead, i.e.

$$\begin{bmatrix} A^{\downarrow}_{\bar{t}}(\omega) \\ B^{\downarrow}_{\bar{t}}(\omega) \end{bmatrix} \coloneqq \underline{\overline{\mathbb{K}}}^{\downarrow}_{(\Delta, g_n)} \circ \underline{\overline{\mathbb{K}}}^{\downarrow}_{\mathsf{task}(\mathfrak{s}_{n-1})} \circ \cdots \circ \underline{\overline{\mathbb{K}}}^{\downarrow}_{\mathsf{task}(\mathfrak{s}_0)} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}.$$

We present an algorithm that builds upon the previous results.

Expressing the distribution of $[A_{\bar{t}}^{\uparrow}; B_{\bar{t}}^{\uparrow}]$ and $[A_{\bar{t}}^{\downarrow}; B_{\bar{t}}^{\downarrow}]$. Let us fix an input MTP $\boldsymbol{M} = (\mathfrak{S}, P, \pi, \mathsf{task})$, a SoC distribution $\langle \bar{f}, f, z \rangle_0$ that represents $[A_0; B_0]$, and a time horizon $\bar{t} > 0$. We consider the joint distribution of under- / over-approximation of the SoC and the MTP. Intuitively, we split the SoC distribution into under- and over-approximating subdistributions and move them along the paths of $\boldsymbol{M}$ according to the probabilistic branching of the MTP. We notice that we do not need to explore all exponentially many paths; when two paths visit the same MTP state at the same moment, we can again merge the two subdistributions. This process is formalized by the following graph and a procedure on how to propagate the distribution through the graph.

For a given MTP $\boldsymbol{M}$ we define a directed acyclic graph (V, E) over

$$V = S \times \{0, 1, \ldots, \lfloor \bar{t} \rfloor, \bar{t}\}$$

There is an edge from a vertex $(\mathfrak{s}_0, t_0)$ to a vertex $(\mathfrak{s}_1, t_1)$ with $\mathsf{task}(\mathfrak{s}_0) \coloneqq (\Delta, g)$ if

- $P(\mathfrak{s}_0, \mathfrak{s}_1) > 0$, and
- $t_1 = \min\{t_0 + \Delta, \bar{t}\}$.

Furthermore, we are only interested in the reachable vertices from one of the initial state vertices. Formally, we define $(V_{\mathsf{reach}}, E_{\mathsf{reach}})$ to be the graph obtained from (V, E) by removing vertices that are not reachable from any $(\mathfrak{s}, 0)$ with $\pi(\mathfrak{s}) > 0$. An example of this graph is displayed in Figure 3.4.

We propagate subdistributions through this graph as follows.

Initialization: We label each vertex of the form $(\mathfrak{s}, 0)$ where $\pi(\mathfrak{s}) > 0$ by the pair of equal initial subdistributions

$$\langle \bar{f}, f, z \rangle^{\downarrow} \coloneqq \langle \bar{f}, f, z \rangle_0 \cdot \pi(\mathfrak{s}) \qquad \text{and} \qquad \langle \bar{f}, f, z \rangle^{\uparrow} \coloneqq \langle \bar{f}, f, z \rangle_0 \cdot \pi(\mathfrak{s})$$

Repeat: We repeat the following steps as long as possible.

1. For each vertex $(\mathfrak{s}, t)$ labeled by $\langle \bar{f}, f, z \rangle^{\downarrow}$ and $\langle \bar{f}, f, z \rangle^{\uparrow}$, we obtain $\langle \bar{f}, f, z \rangle_{\Delta}^{\downarrow}$ and $\langle \bar{f}, f, z \rangle_{\Delta}^{\uparrow}$ by Lemma 22 for $\mathsf{task}(\mathfrak{s}) \coloneqq (\Delta_{\mathfrak{s}}, g)$ where we possibly crop the task duration if the time horizon is exceeded, i.e.

   $$\Delta \coloneqq \min\{\Delta_{\mathfrak{s}}, \bar{t} - t\}.$$

   Then we label the i-th outgoing *edge* of $(\mathfrak{s}, t)$ leading to some $(\mathfrak{s}', t')$ by

   $$\langle \bar{f}, f, z \rangle_i^{\downarrow} \coloneqq \langle \bar{f}, f, z \rangle_{\Delta}^{\downarrow} \cdot P(\mathfrak{s}, \mathfrak{s}') \qquad \text{and} \qquad \langle \bar{f}, f, z \rangle_i^{\uparrow} \coloneqq \langle \bar{f}, f, z \rangle_{\Delta}^{\uparrow} \cdot P(\mathfrak{s}, \mathfrak{s}').$$

2. For each vertex $(\mathfrak{s}, t)$ such that its k ingoing edges are labelled by $\langle \bar{f}, f, z \rangle_i^{\downarrow}$ and $\langle \bar{f}, f, z \rangle_i^{\uparrow}$ for $i = 1, \ldots, k$, we label $(\mathfrak{s}, t)$ by

   $$\langle \bar{f}, f, z \rangle^{\downarrow} \coloneqq \sum_{i=1}^{k} \langle \bar{f}, f, z \rangle_i^{\downarrow} \qquad \text{and} \qquad \langle \bar{f}, f, z \rangle^{\uparrow} \coloneqq \sum_{i=1}^{k} \langle \bar{f}, f, z \rangle_i^{\uparrow}.$$

Finalization: Let the i-th of all n vertices of the form $(\mathfrak{s}, \bar{t}) \in V_{\mathsf{reach}}$ be labelled by $\langle \bar{f}, f, z \rangle_i^{\downarrow}$ and $\langle \bar{f}, f, z \rangle_i^{\uparrow}$. The final distributions that represent $[A_{\bar{t}}^{\uparrow}; B_{\bar{t}}^{\uparrow}]$ and $[A_{\bar{t}}^{\downarrow}; B_{\bar{t}}^{\downarrow}]$ respectively are

$$\langle \bar{f}, f, z \rangle_{\bar{t}}^{\downarrow} \coloneqq \sum_{i=1}^{n} \langle \bar{f}, f, z \rangle_i^{\downarrow} \qquad \text{and} \qquad \langle \bar{f}, f, z \rangle_{\bar{t}}^{\uparrow} \coloneqq \sum_{i=1}^{n} \langle \bar{f}, f, z \rangle_i^{\uparrow}.$$

We naturally arrive at the following correctness statement.

Lemma 24. A battery with SoC distribution $\langle \bar{f}, f, z \rangle_0$ $\overline{\mathbb{K}}$-powers a system $\boldsymbol{M}$ for time $\bar{t}$ with probability at least $1 - z_{\bar{t}}^{\uparrow}$ and at most $1 - z_{\bar{t}}^{\downarrow}$, where $z_{\bar{t}}^{\uparrow}$ and $z_{\bar{t}}^{\downarrow}$ are the depletion probabilities of the densities representing $[A_{\bar{t}}^{\uparrow}; B_{\bar{t}}^{\uparrow}]$ and $[A_{\bar{t}}^{\downarrow}; B_{\bar{t}}^{\downarrow}]$, respectively.

This theorem relies on the simple observation that an under-approximation of the SoC is an over-approximation of the depletion probability.

> ***Remark — on complexity.*** As indicated in the beginning of this section, we do not need to track all exponentially many paths through the MTP up to time $\bar{t}$. In fact, in the algorithm above, once we have computed the subdistributions on the left hand side, we can discard the subdistribution on the right hand side of the assignments. If the he task durations are natural numbers, the amount of subdistributions we need to track simultaneously is bounded by $|S| \cdot D$ where D is the smallest common multiple of all the task durations. D always exists since all task durations are strictly positive.

Translating Timed SDF Graphs To Equivalent MTPs. As mentioned above, some well known formalisms can be translated to MTPs, among them the timed version of *Synchronous Data Flow (SDF)* [25] and some *Scenario-Aware Data Flow (SADF)* [39] flavors. We will demonstrate informally how to translate a timed SDF graph (SDFG) to an equivalent MTP.

SDF is a widely used formalism for modelling and analyzing networks of deterministic sequential processes along with their resource budget. Processes, called *actors* communicate via consuming and producing tokens (data elements) from their incoming and to their outgoing unbounded channels. Whenever an actor is activated it spawns a new active *instance*. The number of tokens an instance consumes and eventually produces is fixed a priori. The timed version additionally annotates each actor a with a constant execution time $e(a) \in \mathbb{N} \setminus \{0\}$, representing how much time passes between consumption and production of tokens.

We furthermore associate a distribution $L(a)$ over loads with each actor a to reason about energy consumption.

The semantics of an SDFG execution is a finite *Labelled Transition System (LTS)* over its configurations, which can be extended to an MTP on the same state space of configurations with Dirac transitions and additional annotations concerning load and sojourn times. An SDF configuration records

1. a vector v representing the number of tokens in each channel,

2. A set $\mathscr{A}$ collecting active actor instances a_i of any actor a and
3. the residual execution time of each running instance as a map r.

Transitions between states are of three natures:

start a**:** A new instance a_i of actor a with $r(a_i) = e(a)$ is added to $\mathscr{A}$, provided the input channels contain enough tokens, which are thereby consumed;

end a**:** An actor instance a_i is removed from $\mathscr{A}$ when its residual execution time $r(a_i)$ is 0. Thereby output tokens are produced according to a's output channels;

time t**:** Under the precondition that no **start** or **end** transitions are possible, an amount of time t passes corresponding to the minimum of the residual times, thereby decreasing the residual execution times of every active actor instance accordingly.

The precondition of **time**-type transitions implies that the LTS is free of nondeterminism between **time** and **start**/**end** transitions. The nondeterminism among **start**/**end** transitions is irrelevant thanks to the *diamond property*. This means that for each state s there is a unique *final* state s' such that each maximal sequence of **start**/**end** transitions from s ends up in s'. On each such diamond (set of states reachable from s) no time passes, thus it has no effect on the battery. As the first step in defining the MTP, we transform the LTS by collapsing each diamond into its final state. As a result, the LTS becomes deterministic with only **time** transitions remaining. If the starting state was part of a diamond collapsed to a state s', this state becomes the initial state of the transformed LTS.
The reachable part of the LTS induces an MTP $\boldsymbol{M} = (\mathfrak{S}, P, \pi, \mathsf{task})$ as follows:

- The state space $\mathfrak{S}$ is defined as the reachable states of the LTS,
- The initial probability distribution π is Dirac in the initial state of the SDFG,
- The task annotation $\mathsf{task}(\mathfrak{s}) = (\Delta_{\mathfrak{s}}, g_{\mathfrak{s}})$ is constructed as follows:
  - $\Delta_{\mathfrak{s}}$ is the residual time according to the **time**-transition leaving $\mathfrak{s}$.
  - $g_{\mathfrak{s}}$ for $\mathfrak{s} = (v, \mathscr{A}, r) \in \mathfrak{S}$ is the convolution of the $L(a)$ for each $a_i \in \mathscr{A}$.
- $P(\mathfrak{s}_0, \mathfrak{s}_1)$ is 1 if there is a **time** transition from $\mathfrak{s}_0$ to $\mathfrak{s}_1$, and 0, otherwise.

SADF extends SDF to discrete execution time distributions and scenarios (with subscenarios probabilistically chosen through discrete-time Markov Chains). An extension of the above is relatively intuitive, but technically involved. However, since the semantics of such SADF graphs under self-timed executions are *Timed Probabilistic Systems (TPS)* with the diamond property for actions [40], an analogous approach to the above can be formulated.

3.7 Consolidating The KiBaM And Measurements Via Kalman Filters

The idea of performing SoC estimation using Kalman filters (see Section 2.2) is not new [32, 18, 17], yet most approaches lack a suitable dynamical model [26]. This work is the first to propose the KiBaM to fill this gap.

The Kalman filter operates in discrete time, but just like many other systems the KiBaM does not. We thus we have to cast it into the necessary form. A simple way of doing this is to discretize the defining ODEs of the KiBaM SoC (Equation 3.1) to difference equations

$$\begin{aligned} a_{k+1} &= -\ell_k \cdot \Delta t + a_k + \mathsf{p} \cdot \Delta t \cdot \left(\frac{b_k}{1-\mathsf{c}} - \frac{a_k}{\mathsf{c}} \right) \text{ and} \\ b_{k+1} &= \qquad\qquad b_k + \mathsf{p} \cdot \Delta t \cdot \left(\frac{a_k}{\mathsf{c}} - \frac{b_k}{1-\mathsf{c}} \right), \end{aligned}$$

where Δt is the length of a discrete time step and ℓ_k is the load on the battery at time k throughout a time step. In matrix-vector notation we get

$$\begin{bmatrix} a_{k+1} \\ b_{k+1} \end{bmatrix} = \mathscr{D} \cdot \begin{bmatrix} a_k \\ b_k \end{bmatrix} + \begin{bmatrix} \Delta t & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \ell_k \\ 0 \end{bmatrix} \qquad \text{where} \qquad \mathscr{D} \coloneqq \begin{bmatrix} 1 - \frac{\mathsf{p} \cdot \Delta t}{\mathsf{c}} & \frac{\mathsf{p} \cdot \Delta t}{1-\mathsf{c}} \\ \frac{\mathsf{p} \cdot \Delta t}{\mathsf{c}} & 1 - \frac{\mathsf{p} \cdot \Delta t}{1-\mathsf{c}} \end{bmatrix}.$$

$\mathscr{D}$ is called the *diffusion matrix* that gathers terms depending on the KiBaM parameters c and p. This exactly matches the form of the first Kalman Filter equation 2.1. Thus by identification, we instantiate the Kalman filter with

$$F_k \coloneqq \mathscr{D}, \quad x_k \coloneqq \begin{bmatrix} a_k \\ b_k \end{bmatrix}, \quad B_k \coloneqq \begin{bmatrix} \Delta t & 0 \\ 0 & 0 \end{bmatrix}, \quad u_k \coloneqq \begin{bmatrix} \ell_k \\ 0 \end{bmatrix} \quad \text{and } w_k \coloneqq \mathbf{0}$$

In this way, we use the load on the battery as control input. This is realistic in a scheduling setting, as we select the tasks to perform, which in turn determines the load on the battery. We will henceforth refer to this instantiation of the Kalman filter as the *KiBaM filter*.

3.8 Proof of Concept with Synthetic Data

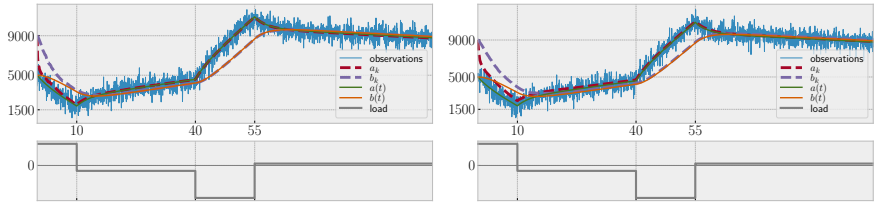
This section demonstrates the performance of the KiBaM filter on synthetic data as a proof of concept. We show that the KiBaM filter is robust against inaccurate choices of initial battery SoC levels and can indeed conjoin information from observed quantities with model predictions of the SoC.

We assume that we received a sequence of available charge measurements perturbed with white noise with known standard deviation σ. In fact, such data can easily be synthesized by adding white noise to available charge traces generated by tracing a KiBaM SoC along an arbitrary task sequence. As a running example for the purpose of visualization, we essentially adopted the data from Example 1.

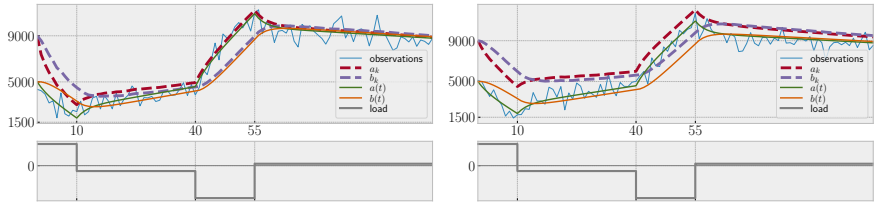
Formally, assume $v_k \coloneqq [\tilde{a}_k; 0]$ where $\tilde{a}_k \sim \mathscr{N}(0, \sigma^2)$. We map measurements into the state space using the identity mapping on the first component. Thus, we conclude

$$H_k \coloneqq \mathsf{diag}(1,0) \quad \text{and} \quad R_k \coloneqq \mathsf{diag}(\sigma^2, 0).$$

Figure 3.5 shows how the KiBaM filter can indeed very accurately estimate both available and bound charge quantities from just a noisy sequence of available charge values. We vary sample frequencies and confidence in the initial state. The filter assumes an initial SoC of [9000; 9000], which is vastly different to the actual initial SoC of [5000; 5000]. We observe that the filter quickly recognizes its severely wrong assumption if P_0 suggests large initial state variance and corrects



(a) Every 10-th sample ($\Delta t = 0.1$) and $P_0 = \mathbb{D}_{\sigma,\mathsf{c}}$ **(b)** Every 10-th sample ($\Delta t = 0.1$) and $P_0 = \frac{1}{5}\mathbb{D}_{\sigma,\mathsf{c}}$



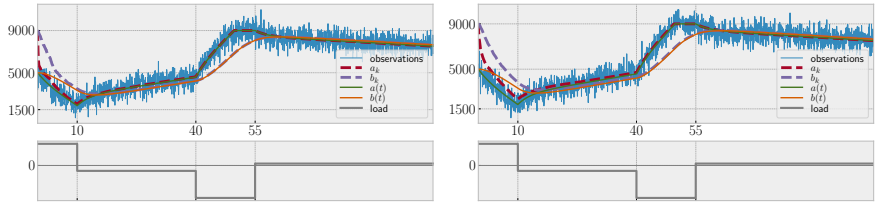
(c) Every 200-th sample ($\Delta t = 2$) and $P_0 = \mathbb{D}_{\sigma,\mathsf{c}}$ **(d)** Every 200-th sample ($\Delta t = 2$) and $P_0 = \frac{1}{5}\mathbb{D}_{\sigma,\mathsf{c}}$

Figure 3.5: The KiBaM filter ($\mathsf{c} = 0.5$, $\mathsf{p} = 0.04$) estimating the SoC without capacity limits from a series of noisy available charge measurements ($\sigma = 600$) at every time step $\Delta t = 0.01$, assuming an initial state of $x_0 = [9000; 9000]$ for different number of samples and initial state variance matrices P_0 involving the diagonal matrix $\mathbb{D}_{\sigma,\mathsf{c}} \coloneqq \mathsf{diag}\big(\sigma^2\mathsf{c}, \sigma^2(1-\mathsf{c})\big)$.

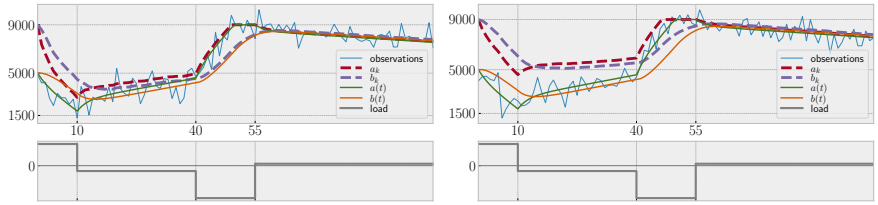
the estimation downwards quickly, since it "trusts" the measurements. Once it feels "confident" enough about its estimated state, it is unfazed by the stochastic fluctuations since the system dynamics do not allow for such quick changes. It becomes apparent that the Kalman estimates converge to the true system state quicker if running with a higher measurement frequency and higher initial state variance. Independently of the sample frequency, the estimates and the true state will not deviate again once they are close.

The KiBaM dynamical system as presented in this section thus far does not incorporate capacity limit knowledge. It is however easy to incorporate such limits into the KiBaM filter approach. Let $\overline{a}$ and $\overline{b}$ be the limits on available and bound charge, respectively. If in any time step k the KiBaM filter predicts a SoC violating either bound, we reset the corresponding component of the SoC estimate to its maximum, i.e. if for any k we have $a_k > \overline{a}$ then $a_k \coloneqq \overline{a}$ and analogously for b_k. In principle, we would have to redo the last filtering step, since the difference equations are coupled, which leads to a slight overapproximation of the bound charge in the k-th step. However, theses errors will be very small since the time step Δt is small. In addition the KiBaM filter will implicitly detect these deviations over the course of time and adjust its estimates, thereby preventing error accumulation. Notably, and in contrast to the time-continuous KiBaM, with the time-discrete version it is possible for the bound charge to reach its limit before the available charge does, if Δt is too large.

Figure 3.6 shows how the KiBaM Filter performs when we make it aware of battery capacity limits. Essentially, the estimation quality remains unchanged or actually becomes better, the reason being that the upper capacity limits do not allow for arbitrarily large SoCs while charging. The plots show that a KiBaM filter



(a) Every 10-th sample ($\Delta t = 0.1$) and $P_0 = \mathbb{D}_{\sigma,\mathsf{c}}$ **(b)** Every 10-th sample ($\Delta t = 0.1$) and $P_0 = \frac{1}{5}\mathbb{D}_{\sigma,\mathsf{c}}$



(c) Every 200-th sample ($\Delta t = 2$) and $P_0 = \mathbb{D}_{\sigma,\mathsf{c}}$ **(d)** Every 200-th sample ($\Delta t = 2$) and $P_0 = \frac{1}{5}\mathbb{D}_{\sigma,\mathsf{c}}$

Figure 3.6: The KiBaM filter ($\mathsf{c} = 0.5$, $\mathsf{p} = 0.04$) estimating the SoC with capacity limits $[\overline{a};\overline{b}] = [9000;9000]$ from a series of noisy available charge measurements ($\sigma = 600$) at every time step $\Delta t = 0.01$, assuming an initial state $x_0 = [9000;9000]$ for different number of samples and and initial state variance matrices P_0, again involving $\mathbb{D}_{\sigma,\mathsf{c}} := \mathsf{diag}\big(\sigma^2\mathsf{c}, \sigma^2(1-\mathsf{c})\big)$.

can indeed handle the capacity limits very well.

3.9 Discussion

This chapter provides a thorough and rigorous analysis of the mathematical foundations of the kinetic battery model (KiBaM). We start with the unbounded two-ODE system from the literature, and subsequently provide the solution of said ODE system in matrix-vector notation, which proves to be the main mathematical object we are working with: the operator $\mathbb{K}$.

We capture certain properties about this operator that prove valuable for an eventual efficient energy budget analysis later on, i.e. monotonicity of $\mathbb{K}$.

We augment the model with a concept of depletion in the form of unsafe states of charge (SoCs), in a relatively straight-forward and intuitive manner: an empty battery stays empty and is no further able to support any load, i.e. the successor SoC of an empty battery invariantly stays empty. This is reflected in the operator $\underline{\mathbb{K}}$.

The incorporation of capacity limits proves to be more intricate, as empty and full batteries are non-symmetric scenarios: a full battery continues to power a system, thus its further evolution must be considered.

We formalize the scenario and develop the mathematical foundations of further charging an already saturated battery, by providing a different ODE system, for which we provide subsequent mathematical analysis, resulting in the operator $\mathbb{K}^{\mathsf{sat}}$ and its properties. In conjunction with the fact that we switch from $\underline{\mathbb{K}}$ to $\mathbb{K}^{\mathsf{sat}}$ exactly at the saturation time-point results in the operator $\overline{\underline{\mathbb{K}}}$, the KiBaM operator that is aware of upper and lower capacity bounds.

Unfortunately, finding the exact time-points of battery saturation turns out to be computationally cumbersome, thus appropriate approximations are developed, essentially by pre- and postponing saturation time-points to computable under- and over-approximations of the actual (transcendental) saturation time-point.

The composition of the saturation time-point approximations and $\mathbb{K}^{\mathsf{sat}}$ eventually results in $\overline{\mathbb{K}}$'s approximation operators $\overline{\mathbb{K}}^{\uparrow}$ and $\overline{\mathbb{K}}^{\downarrow}$.

We consider the context of random SoCs and random loads, rooted in manufacturing tolerances as well as self-discharge for the former, and measurement noise for the latter, thereby rendering the KiBaM into a stochastic object.

We formalize the concept of SoC distributions and eventually formalize the distribution of its $\overline{\mathbb{K}}$-successor, by viewing $\overline{\mathbb{K}}$ as a transformer for random variables. For the same computational reasons as in the deterministic case, we continue by deriving analytic expressions for the distribution of the $\overline{\mathbb{K}}^{\uparrow}$- and $\overline{\mathbb{K}}^{\downarrow}$-successors of a random SoC.

We introduce Markov Task Processes (MTPs) as a load model. An MTP extends the prior load model of task sequences by probabilistic branching, i.e. the next task to be powered is not predetermined as in task sequences, but is instead chosen at random according to the branching structure of an MTP.

We provide an algorithm to approximate the SoC distribution at a given time horizon, by propagating the initial distribution along the paths generated by an MTP using operators $\overline{\mathbb{K}}^{\uparrow}$ and $\overline{\mathbb{K}}^{\downarrow}$, while mitigating the exponential blow-up of the number of paths that lead to the given time horizon.

Finally, we consolidate the KiBaM, a model defined on a non-measurable state of charge and the abstract load quantity, with their noisy counterparts. These play the synthetic role of noisy quantities like battery voltage and current, in order to provide precise SoC estimates during live operation of a system. To this end, we discretize the KiBaM's ODEs into difference equations that are a perfect fit for a Kalman filter. We provide a proof of concept using synthetic noisy data, and showcase that the resulting KiBaM filter is indeed able to adjust a faulty SoC (possibly due to wrong initial estimates of the actual SoC) according to synthetic noisy data.

CHAPTER

4 Algorithms

In this chapter we put our focus on efficient implementations of the content discussed in Chapter 3. We distinguish between two paradigms, the first of which is based on discretization (Section 4.1). Discretization algorithms allow us to track an entire SoC distribution along a Markov Task Process up until a given time horizon in both under– and over-approximative fashion. Here, we start by introducing discretized versions of SoC distributions and the load model of noisy tasks in Section 4.1. We develop two different schemes of discretization that work on these disretized structures, a static and an adaptive one. In static discretization (Section 4.2), we assume that the entire safe SoC space needs to be discretized and considered. In contrast to this idea, adaptive discretization, introduced in Section 4.3, only considers the relevant neighborhood of the support of a SoC distribution, which often times is significantly smaller than the entire safe SoC space. We show that the overhead of dealing with such neighborhoods is usually negligible, while the increase in precision is considerable. On the other hand, Section 4.5 suggests that in most cases only a good estimate of the depletion risk up to a certain time horizon is desirable, instead of the entire final SoC distribution. In this context, we show how only a logarithmic amount of specific SoCs, called SoC percentiles, need to be tracked along a task sequence in order to bound the depletion risk up to a given precision in an almost entirely analytical approach we call percentile propagation. The price to pay in this approach, is that of generality of the initial SoC distribution. However, we propose that the relevant subclass of SoC distributions this approach is able to handle, constitutes the arguably only subclass worth considering. We finalize the chapter by comparing percentile propagation with adaptive discretization and conclude the decisive superiority of percentile propagation in computational efficiency in both time and space, as well as precision, when it comes to simple task sequences. For sequences of noisy tasks we empirically show that percentile propagation is still a valid alternative to the discretization algorithms, because of its low space requirements.

4.1 Discretization Algorithms

In this section we will formalize what it means to discretize a SoC distribution. The aim is to eventually come up with efficient algorithms to compute the $\overline{\mathbb{K}}$-successor

of such a distribution given a task, and iteratively along a sequence of tasks generated by a Markov Task Process. We provide two algorithms that rely on discretization, both of which can be cast into the same framework. The setup of this framework is thus purposely phrased as generic as possible. It relies on bounding boxes of a SoC distribution, that essentially characterize preferably small neighborhoods of the support of a SoC distribution. Successors of a SoC distribution in general do not have the same support as the predecessor SoC distribution itself. For that reason we at first think of this neighborhood as the entire space of safe SoCs, since it is intuitively clear that the support of any $\overline{\underline{\mathbb{K}}}$-successor will be contained in this space. As the term neighborhood already suggests, this concept can be refined to come up with tighter bounds to make the discretization algorithm more accurate and computationally more efficient. The concept of such a neighborhood will thus be kept as abstract as possible so as to harbour the two above concepts.

Discretization Of SoC Distributions

A SoC distribution $\langle \bar{f}, f, z \rangle$ is a triple of two truncated probability density functions $\bar{f}$ and f, as well as the depletion risk z. Since $\bar{f}$ is a one-dimensional density function we delimit its support using two (bound charge) values, while f is a two-dimensional density function, thus we need two SoCs or two intervals, characterizing a rectangular area, to fully determine a neighborhood of support.

We call such a neighborhood, a bounding box of a SoC distribution.

Bounding Boxes

We begin by formally introducing bounding boxes.

Definition 26 — *Bounding box.* A *box* $\mathfrak{B}$ is a triple of intervals $\langle A, B, \overline{B} \rangle$ such that

$$A \subseteq \left[\underline{a}, \overline{a}\right], \quad B \subseteq \left[\underline{b}, \overline{b}\right] \quad \text{and} \quad \overline{B} \subseteq \left[\underline{b}, \overline{b}\right].$$

$\mathfrak{B}$ is called *empty* iff

$$A = \emptyset \vee B = \emptyset \quad \text{and} \quad \overline{B} = \emptyset.$$

$\mathfrak{B}$ is a *bounding box* of a SoC distribution $\langle \bar{f}, f, z \rangle$ if additionally

$$\int_B \bar{f}(b)\,\mathrm{d}b = \int_{\overline{\mathbb{S}}} \bar{f}(b)\,\mathrm{d}b \quad \text{and} \quad \iint_{A\times B} f(a,b)\,\mathrm{d}a\,\mathrm{d}b = \iint_{\overline{\mathbb{S}}\times\overline{\mathbb{S}}} f(a,b)\,\mathrm{d}a\,\mathrm{d}b.$$

The above definition essentially describes the fact that we do not lose any probability mass if only focussing on the areas contained in the box.

Discretization

We aim to provide a discretization algorithm to approximate $\overline{\underline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$, the $\overline{\underline{\mathbb{K}}}$-successor of a SoC distribution. The idea is to approximate a SoC distribution $\langle \bar{f}, f, z \rangle$ by a discrete SoC distribution defined on a grid placed within a bounding box $\mathfrak{B} = \langle A, B, \overline{B} \rangle = \langle [a^\uparrow, a^\downarrow], [b^\uparrow, b^\downarrow], [\overline{b}^\uparrow, \overline{b}^\downarrow] \rangle$ of $\langle \bar{f}, f, z \rangle$. The grid has a size N

indicating the number of chunks A, B and $\overline{B}$ is split into. We then divide the non-saturated part $A \times B$ of a box into a two-dimensional grid and the saturated part $\overline{B}$ into a one-dimensional grid of equisized chunks. Computing the $\underline{\overline{\mathbb{K}}}$-successor of each of these chunks and moving the associated probability mass accordingly, eventually leads to a discretized version of $\underline{\overline{\mathbb{K}}}_T\langle \bar{f}, f, z\rangle$.

For this purpose we first provide formal meaning of what it means to place a grid into a box.

Definition 27. Let $\mathfrak{B} = \langle [a^\uparrow, a^\downarrow], [b^\uparrow, b^\downarrow], [\overline{b}^\uparrow, \overline{b}^\downarrow]\rangle$ be a box. Given a positive *size* $N \in \mathbb{N}_{>0}$, the *grid of size* N of $\mathfrak{B}$ is defined by values

$$\begin{aligned} \overline{B}(i) &\coloneqq \overline{b}^\uparrow + i\delta_{\mathsf{sat}}, && i = 0,\dots,N \\ A(i) &\coloneqq a^\uparrow + i\delta_a, && i = 0,\dots,N \\ B(i) &\coloneqq b^\uparrow + i\delta_b, && i = 0,\dots,N \end{aligned}$$

with *discretization parameters* δ_a , δ_b and δ_{sat} given by

$$\delta_a \coloneqq \frac{(a^\downarrow - a^\uparrow)}{N}, \quad \delta_b \coloneqq \frac{(b^\downarrow - b^\uparrow)}{N} \quad \text{and} \quad \delta_{\mathsf{sat}} \coloneqq \frac{(\overline{b}^\downarrow - \overline{b}^\uparrow)}{N}.$$

By $\mathfrak{B}]i]$, $\mathfrak{B}[i[$ as well as $\mathfrak{B}]i,j]$ and $\mathfrak{B}[i,j[$ we denote half open intervals of the form

$$\begin{aligned} \mathfrak{B}[i[&\coloneqq [\overline{B}(i), \overline{B}(i+1)[, && i = 0,\dots,N-1 \\ \mathfrak{B}]i] &\coloneqq]\overline{B}(i), \overline{B}(i+1)], && i = 0,\dots,N-1 \\ \mathfrak{B}[i,j[&\coloneqq [A(i), A(i+1)[\times [B(j), B(j+1)[, && i,j = 0,\dots,N-1 \\ \mathfrak{B}]i,j] &\coloneqq]A(i), A(i+1)] \times]B(j), B(j+1)], && i,j = 0,\dots,N-1 \end{aligned}$$

and call them the *cells* of the grid.
Finally, the *representatives* of the cells $\mathfrak{B}[i,j[$ and $\mathfrak{B}[i[$ are $[A(i); B(j)]$ and $\overline{B}(i)$, respectively, while the representatives of the cells $\mathfrak{B}]i,j]$ and $\mathfrak{B}]i]$ are $[A(i+1); B(j+1)]$ and $\overline{B}(i+1)$, respectively.

The idea of the discretization is that the probability mass contained in an entire cell is condensed into the cell's representative. A depiction of the concept is displayed in Figure 4.1. Depending on whether we are interested in computing an over-approximation or an under-approximation we choose $[A(j+1); B(j+1)]$ (the top right corner point of the cell) and $\overline{B}(i+1)$ (the right-hand boundary of the cell) or $[A(j); B(j)]$ (the bottom left corner point of the cell) and $\overline{B}(i)$ (the left-hand boundary of the cell) respectively. In other words, cells are represented by the largest and the smallest SoC contained in the cell, respectively.

Formally, we define a discrete SoC distribution as follows.

Definition 28 — *Discrete SoC distribution.* Let $\mathfrak{B}$ be a box and N be the size of the grid on $\mathfrak{B}$. A *discrete SoC distribution* on $\mathfrak{B}$ is a tuple $\langle \overline{\mu}, \mu, \mathfrak{z}\rangle$ with

$$\overline{\mu}(i) \geq 0, \quad \mu[i;j] \geq 0, \text{ and } \quad \mathfrak{z} \geq 0 \qquad \text{for } i,j = 0,\dots,N$$

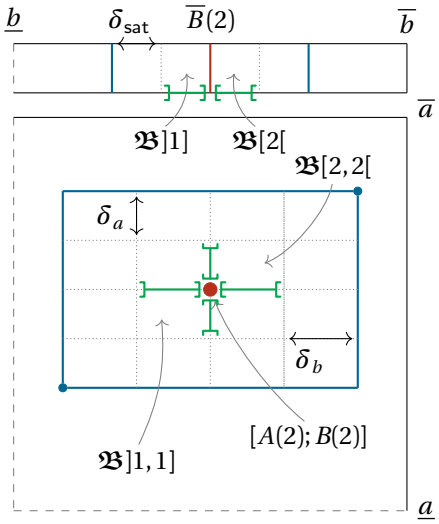


Figure 4.1: A grid (grey dotted lines) of size 4 with discretization parameters δ_a, δ_b and δ_{sat} , placed into a box $\mathfrak{B}$ (solid blue lines). The cells of the grid denoted by $\mathfrak{B}]\bullet,\bullet]$, $\mathfrak{B}[\bullet,\bullet[$ and $\mathfrak{B}]\bullet]$, $\mathfrak{B}[\bullet[$ are the areas between the grid lines (green intervals). The cell's representatives (depicted in red) referred to by $[A(\bullet);B(\bullet)]$ are the bottom left and top right corners, while $\overline{B}(\bullet)$ is represented by the left- and right-hand boundaries. Formally, the representatives are the smallest and largest SoC still contained in the cell.

and

$$\sum_{i=0}^{N} \overline{\mu}(i) \quad + \quad \sum_{i,j=0}^{N} \mu[i;j] \quad + \quad \mathfrak{z} \quad = \quad 1,$$

where we use short-hand notation

$$\overline{\mu}(i) \coloneqq \overline{\mu}(\overline{B}(i)) \quad \text{and} \quad \mu[i;j] \coloneqq \mu[A(i);B(j)].$$

The following lemma describes a recipe to discretize a SoC distribution onto a grid. The intuition behind it is, that the probability mass contained in each cell of the grid is condensed into its representative.

Lemma 25 — *Discretizing a SoC distribution.* Let $\langle \bar{f}, f, z \rangle$ be a SoC distribution, $\mathfrak{B}$ be a corresponding bounding box, with a grid of size N. Then

$\langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle$ and $\langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle$, defined as

$$
\begin{aligned}
\mathfrak{z} &\coloneqq z \\
\overline{\mu}^{\uparrow}(j) &\coloneqq \int_{\mathfrak{B}[j[} \bar{f}(b)\,\mathrm{d}b && j = 0, \dots, N-1 \\
\mu^{\uparrow}[i;j] &\coloneqq \iint_{\mathfrak{B}[i,j[} f(a,b)\,\mathrm{d}a\,\mathrm{d}b && i,j = 0, \dots, N-1 \\
\overline{\mu}^{\downarrow}(j+1) &\coloneqq \int_{\mathfrak{B}]j]} \bar{f}(b)\,\mathrm{d}b && j = 0, \dots, N-1 \\
\mu^{\downarrow}[i+1;j+1] &\coloneqq \iint_{\mathfrak{B}]i,j]} f(a,b)\,\mathrm{d}a\,\mathrm{d}b && i,j = 0, \dots, N-1
\end{aligned}
$$

are discrete SoC distributions.

Proof:

By the fact that $\mathfrak{B}$ is a bounding box of $\langle \bar{f}, f, z \rangle$, and the cells of its grid are disjoint, we integrate exactly over the area of $\mathfrak{B}$. ■

We define the following probability measure on discrete SoC distributions.

Definition 29. For any measurable set X, and discrete SoC distribution $\langle \overline{\mu}, \mu, \mathfrak{z} \rangle$ with associated grid of size N, we have

$$
\mathbf{Pr}[S \in X] \coloneqq \sum_{i,j=0}^{N} \mu[i;j] \cdot \mathbb{I}_{[A(i);B(i)] \in X} \quad + \quad \sum_{i=0}^{N} \overline{\mu}(i) \cdot \mathbb{I}_{[\overline{a};\overline{B}(i)] \in X} \quad + \quad \mathfrak{z} \cdot \mathbb{I}_{\bot \in X}
$$

where $\mathbb{I}_{\varphi}$ is again the indicator function of a condition φ.

Lemma 26. Let $\langle \bar{f}, f, z \rangle$ be a SoC distribution and $\langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle$ and $\langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle$ be its discretized versions with grid of size N according to Lemma 25. Furthermore, let $[A;B]$ as well as $[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}]$ and $[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}]$ be random variables distributed as

$$
[A;B] \sim \langle \bar{f}, f, z \rangle, \quad [A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \sim \langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle \quad \text{and} \quad [A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \sim \langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle.
$$

For any SoC $S \in \overline{\mathbb{S}}_{\bot}$, we have

$$
\mathbf{Pr}\big[[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \leq S\big] \quad \geq \quad \mathbf{Pr}[[A;B] \leq S] \quad \geq \quad \mathbf{Pr}\big[[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \leq S\big].
$$

Proof:

By the construction given in Lemma 25 we condense the entire probability mass of a cell into its representative, which is a smaller and larger SoC than any other SoC contained in the cell, respective to the kind of approximation we want to compute. ■

Thus, the discretizations of a SoC distribution soundly approximate the actual SoC distribution from above and below on each point of the associated grid of the discretization.

Later on, when we aim to compute $\overline{\mathbb{K}}$-successors of a discretized SoC distribution with respect to a certain noisy task (Δ, g), we must also discretize the load density in order to avoid integration over the support of g. We therefore restrict g to only be of finite support, meaning there is a largest and a smallest load that still has support. This is not a very severe restriction given the actual physical restrictions on the load on real batteries, which exhibit a maximal charging and discharging current.

Definition 30 — *Discrete noisy tasks.* A *discrete noisy task* is a pair (Δ, γ) where $\Delta \in \mathbb{R}_{>0}$ is a positive time duration and γ is a discrete probability distribution with finite support.

With the following lemma we provide a recipe how to discretize noisy tasks.

Lemma 27 — *Discretizing noisy tasks.* Let $T = (\Delta, g) \in \widetilde{\mathbb{T}}$ be a noisy task with g being of finite support and let $L \in \mathbb{N}_{>0}$ be the number of chunks we want to separate g into. Furthermore, let $\underline{\ell}$ and $\overline{\ell}$ be the minimal and maximal value of g with non-zero density value and δ_ℓ be the discretization parameter given by

$$\underline{\ell} := \min\{\ell \in \mathrm{supp}(g)\}, \quad \overline{\ell} := \max\{\ell \in \mathrm{supp}(g)\} \quad \text{and} \quad \delta_\ell := \frac{\overline{\ell} - \underline{\ell}}{L}$$

Then $(\Delta, \gamma^\uparrow)$ and $(\Delta, \gamma^\downarrow)$, with

$$\gamma^\uparrow(\underline{\ell} + i\delta_\ell) := \int_{[i,i+1[} g(\underline{\ell} + l\delta_\ell)\,\mathrm{d}l \qquad \text{for } i = 0, \ldots, L-1$$

$$\gamma^\downarrow(\overline{\ell} - i\delta_\ell) := \int_{[i,i+1[} g(\overline{\ell} - l\delta_\ell)\,\mathrm{d}l \qquad \text{for } i = 0, \ldots, L-1$$

are discrete noisy tasks.

Proof:

The intervals $\left[\underline{\ell} + i\delta_\ell, \underline{\ell} + (i+1)\delta_\ell\right[$ for $i = 0, \ldots, L$ are disjoint and their union equals exactly $\mathrm{supp}(g)$, making $\gamma^\uparrow$ a discrete probability distribution with finite support, given that g is a probability density function with finite support. The claim for $\gamma^\downarrow$ follows in similar fashion. ■

4.2 Static Discretization

We can now move on to formalize how to compute $\overline{\underline{\mathbb{K}}}$-successors of discrete SoC distributions. We are thus now facing the problem that the support of the successor distribution is in general not contained in the bounding box of the initial distribution. To avoid this problem entirely at first, we assume that the support of the SoC distributions spans the entire space of safe SoCs, or in other words, we choose the space of safe SoCs as the trivial bounding box. We then place an invariant grid onto the set of safe SoCs, compute the $\overline{\underline{\mathbb{K}}}$-successor of each cell representative and move the associated probability mass into the cell that $\overline{\underline{\mathbb{K}}}$ maps into. This way of computing successors of discrete SoC distributions turns out to produce sound under- and over-approximations of the actual successor SoC distribution, as we will establish later on.

We formalize the above idea. Let us for the remainder of this section assume a generic successor operator on boxes, to allow for a more general formalization. For the moment one can think of the initial box as being the entire space of safe SoCs and of the $\overline{\underline{\mathbb{K}}}$-operator on boxes as the identity, i.e.

$$\mathfrak{B} = \mathfrak{B}_{\underline{\overline{\mathbb{S}}}} \coloneqq \langle [\underline{a}, \overline{a}], [\underline{b}, \overline{b}], [\underline{b}, \overline{b}] \rangle \quad \text{and} \quad \overline{\underline{\mathbb{K}}}_T \mathfrak{B} \coloneqq \mathfrak{B}$$

and that additionally, we have an invariant grid of size $N \in \mathbb{N}_{>0}$ placed within $\mathfrak{B}$. Because the grid and the boxes stay invariant, we refer to this discretization scheme as *static discretization* (SD). We will eventually refine this concept to be adaptive in the next section.

Lemma 28 — *The set of safe SoCs is a bounding box.* For any SoC distribution, the box $\mathfrak{B}_{\underline{\overline{\mathbb{S}}}}$ given by

$$A = [\underline{a}, \overline{a}], \quad B = [\underline{a}, \overline{a}], \text{ and } \quad \overline{B} = [\underline{b}, \overline{b}]$$

is a trivial bounding box.

Proof:

By definition. ∎

Definition 31 — *Successor of a discrete SoC distribution.* Let $T = (\Delta, g)$ be a noisy task and $(\Delta, \gamma^{\downarrow})$ as well as $(\Delta, \gamma^{\uparrow})$ be its discretized versions according to Lemma 27. Furthermore, let $\langle \overline{\mu}, \mu, \mathfrak{z} \rangle$ be a discrete SoC distribution with associated box $\mathfrak{B}$ and grid of size N. Finally, let $\mathfrak{B}_T \coloneqq \overline{\underline{\mathbb{K}}}_T \mathfrak{B}$ be the successor box with its associated grid of the same size. The $\overline{\underline{\mathbb{K}}}^{\downarrow}$ and $\overline{\underline{\mathbb{K}}}^{\uparrow}$-successor of $\langle \overline{\mu}, \mu, \mathfrak{z} \rangle$ are defined by

$$\overline{\underline{\mathbb{K}}}^{\uparrow}_T \langle \overline{\mu}, \mu, \mathfrak{z} \rangle \coloneqq \langle \overline{\mu}^{\uparrow}_T, \mu^{\uparrow}_T, \mathfrak{z}^{\uparrow}_T \rangle \quad \text{and} \quad \overline{\underline{\mathbb{K}}}^{\downarrow}_T \langle \overline{\mu}, \mu, \mathfrak{z} \rangle \coloneqq \langle \overline{\mu}^{\downarrow}_T, \mu^{\downarrow}_T, \mathfrak{z}^{\downarrow}_T \rangle$$

with

$$\begin{aligned}
\overline{\mu}^{\downarrow}_T(j+1) &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\uparrow})}\gamma^{\uparrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]\in\{\overline{a}\}\times\mathfrak{B}_T]j]\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[A(k);B(l)]\in\{\overline{a}\}\times\mathfrak{B}_T]j]\right\}\right)\\
\mu^{\downarrow}_T[i+1;j+1] &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\uparrow})}\gamma^{\uparrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]\in\mathfrak{B}_T]i,j]\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[A(k);B(l)]\in\mathfrak{B}_T]i,j]\right\}\right)\\
\mathfrak{z}^{\downarrow}_T &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\uparrow})}\gamma^{\uparrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]=\bot\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\downarrow}_{(\Delta,\ell)}[A(k);B(l)]=\bot\right\}\right)+\mathfrak{z}
\end{aligned}$$

and

$$\begin{aligned}
\overline{\mu}^{\uparrow}_T(j) &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\downarrow})}\gamma^{\downarrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]\in\{\overline{a}\}\times\mathfrak{B}_T[j[\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[A(k);B(l)]\in\{\overline{a}\}\times\mathfrak{B}_T[j[\right\}\right)\\
\mu^{\uparrow}_T[i;j] &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\downarrow})}\gamma^{\downarrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]\in\mathfrak{B}_T[i,j[\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[A(k);B(l)]\in\mathfrak{B}_T[i,j[\right\}\right)\\
\mathfrak{z}^{\uparrow}_T &\coloneqq \sum_{\ell\in\mathrm{supp}(\gamma^{\downarrow})}\gamma^{\downarrow}(\ell)\cdot\left(\sum_{k=0}^{N}\left\{\overline{\mu}(k)\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[\overline{a};\overline{B}(k)]=\bot\right\}\right.\\
&\quad\left.+\sum_{k,l=0}^{N}\left\{\mu[k;l]\,\middle|\,\overline{\underline{\mathbb{K}}}^{\uparrow}_{(\Delta,\ell)}[A(k);B(l)]=\bot\right\}\right)+\mathfrak{z}
\end{aligned}$$

for each $i,j=0,\ldots,N-1$.

The $\overline{\underline{\mathbb{K}}}^{\uparrow}$ and $\overline{\underline{\mathbb{K}}}^{\downarrow}$-successor of the discrete SoC distribution are indeed under- and over-approximations of the $\overline{\underline{\mathbb{K}}}$-successor of the actual SoC distribution it was induced from.

Lemma 29. Let $T \in \widetilde{\mathbb{T}}$ be a noisy task, $\langle \bar{f}, f, z \rangle$ be a SoC distribution and let $\langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle$ and $\langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle$ and be its discretized versions according to Lemma 25. Furthermore, let $[A; B]$ as well as $[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}]$ and $[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}]$ be random variables distributed as

$$[A; B] \sim \overline{\mathbb{K}}_T \langle \bar{f}, f, z \rangle, \quad [A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \sim \overline{\mathbb{K}}^{\downarrow}_T \langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle \quad \text{and} \quad [A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \sim \overline{\mathbb{K}}^{\uparrow}_T \langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle.$$

We have, for any SoC S

$$\mathbf{Pr}\big[[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \leq S\big] \quad \geq \quad \mathbf{Pr}[[A; B] \leq S] \quad \geq \quad \mathbf{Pr}\big[[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \leq S\big].$$

Proof:

> Essentially, the claim follows from the combination of facts that we discretize in an under- and over-approximating fashion, respectively, wherever it is necessary. By Lemma 26, $\langle \overline{\mu}^{\uparrow}, \mu^{\uparrow}, \mathfrak{z} \rangle$ and $\langle \overline{\mu}^{\downarrow}, \mu^{\downarrow}, \mathfrak{z} \rangle$ approximate $\langle \bar{f}, f, z \rangle$ from below and above, respectively. Additionally, we have the fact that $\gamma^{\downarrow}$ and $\gamma^{\uparrow}$ are under- and over-approximating discretizations of the actual load distribution g. Finally, with the fact that $\overline{\mathbb{K}}^{\uparrow}$ and $\overline{\mathbb{K}}^{\downarrow}$ approximate the actual successor operator $\overline{\mathbb{K}}$ from above and below, the claim follows. ■

4.3 Adaptive Discretization

When analyzing real world battery powered systems, we usually assume that initially the battery is (nearly) fully charged. In satellites, the battery is of course inserted in a fully charged state into the device, however it may self discharge to a certain extend in its launching pod while waiting for deployment, which potentially spans a period of several months. This leaves more than enough time for the battery to equilibrate. Thus, it is not only reasonable to assume an initially almost fully charged battery, but also one that is in equilibrium. Assuming a small margin of uncertainty around the actual SoC, static discretization of the entire safe SoC space, as introduced in the previous section, would lead to just a handful of non-zero cells, while the overwhelming majority of cells would end up holding zero probability mass. This leads to an prohibitive computational overhead of essentially propagating zeros along a task sequence, becoming increasingly prominent as the length of the task sequence increases. Assume a battery of an abstract capacity of 1000, with $\mathsf{c} = 0.5$, meaning $\overline{a} = 500$ and $\overline{b} = 500$. Additionally, assume we have a grid of size $N = 1000$. For the sake of illustration, assume the battery SoC is initially estimated to be between 250 and 270 in both the available charge and the bound charge dimension. Out of the $N \cdot N = 1\,000\,000$ cells of the unsaturated SoC area in total, only a mere 1600 cells hold a non-zero probability mass. In addition, we would accumulate severe approximation errors with each task, attributed to the ambition of always using sound approximations, i.e. each cell is being represented by its bottom left or top right corner point, depending on the whether we want to compute the under- or over-approximation of the SoC distribution in question.

In the following we improve on these shortcomings. We pursue the idea of discretizing only the interesting portion of the safe SoC space, namely windows

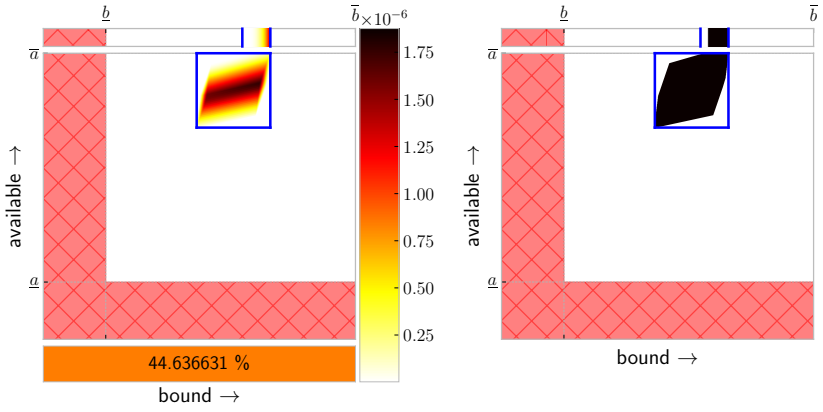


Figure 4.2: Left: A SoC distribution and a corresponding bounding box (blue). **Right:** The support of the SoC distribution (black) to visualize the tightness of the bounding box. The red hatched area is the space of unsafe SoCs.

representing only the immediate neighborhood of the support of the initial battery state distribution. Following this paradigm not only eliminates as many zero-probability cells as possible, but also decreases the size of grid cells, thereby greatly reducing approximation errors. We refer to the resulting discretization scheme as *adaptive discretization* (AD).

An example of a SoC distribution and its bounding box is given in Figure 4.2.

Note that the following development is a generalization of SD where we essentially have used the set of safe SoCs as bounding box and have left it invariant.

Successor Of A Box

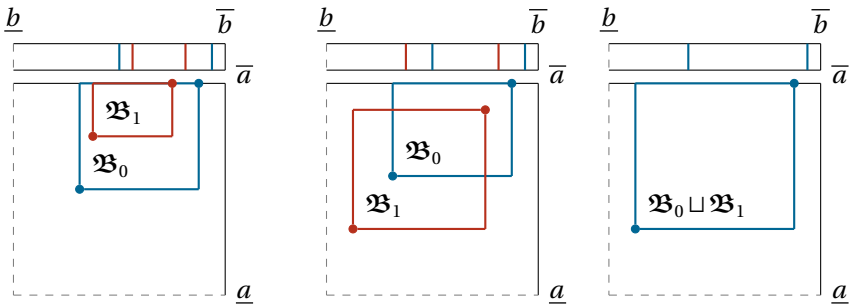
In most cases, after powering a task T, the SoC distribution will end up, at least partially, beyond its initial bounding box $\mathfrak{B}$. Thus, before we transform the SoC distribution according to $\overline{\mathbb{K}}$, we compute the successor bounding box, denoted by $\overline{\mathbb{K}}_T\,\mathfrak{B}$, from the task T and the initial bounding box $\mathfrak{B}$. With the prospect of only discretizing the area defined by the distribution's bounding box, it is desirable to keep it as tight as possible while propagating the box along T, in order to minimize the number of void cells.

We start by introducing the notion of subsumption and closure of bounding boxes, by essentially lifting the subset relation and the union operation on intervals, which we will need later on.

Definition 32 — *Subsumption.* Let $\mathfrak{B}_0 \coloneqq \langle A_0, B_0, \overline{B}_0\rangle$ and $\mathfrak{B}_1 \coloneqq \langle A_1, B_1, \overline{B}_1\rangle$ be boxes. We denote by $\mathfrak{B}_0 \sqsubseteq \mathfrak{B}_1$ that $\mathfrak{B}_0$ *is subsumed by* $\mathfrak{B}_1$, or $\mathfrak{B}_1$ *subsumes* $\mathfrak{B}_0$. Subsumption is defined as

$$\mathfrak{B}_0 \sqsubseteq \mathfrak{B}_1 \;\coloneqq\; A_0 \subseteq A_1 \;\wedge\; B_0 \subseteq B_1 \;\wedge\; \overline{B}_0 \subseteq \overline{B}_1$$

For *strict* subsumption we use the strict subset relation.



$\mathfrak{B}_0$ and $\mathfrak{B}_1$ with $\mathfrak{B}_0 \sqsubset \mathfrak{B}_1$. $\mathfrak{B}_0$ and $\mathfrak{B}_1$ as well as their closure $\mathfrak{B}_0 \sqcup \mathfrak{B}_1$.

Figure 4.3: The illustration of subsumption and closure of boxes.

Definition 33 — *Closure of boxes.* Let $\mathfrak{B}_0 \coloneqq \langle A_0, B_0, \overline{B}_0 \rangle$ and $\mathfrak{B}_1 \coloneqq \langle A_1, B_1, \overline{B}_1 \rangle$ be two boxes. The *closure* of $\mathfrak{B}_0$ and $\mathfrak{B}_1$, denoted by $\mathfrak{B}_0 \sqcup \mathfrak{B}_1$ is defined componentwise by

$$\mathfrak{B}_0 \sqcup \mathfrak{B}_1 \coloneqq \langle A_0 \sqcup A_1, B_0 \sqcup B_1, \overline{B}_0 \sqcup \overline{B}_1 \rangle,$$

where

$$M \sqcup N \coloneqq \begin{cases} M, & \text{if } M \neq \emptyset = N \\ N, & \text{if } M = \emptyset \neq N \\ [\min(M \cup N), \max(M \cup N)], & \text{if } M \neq \emptyset \neq N \end{cases}.$$

The closure over countably many boxes $\bigsqcup_{i=0}^{N} \mathfrak{B}_i$ is defined inductively by

$$\bigsqcup_{i=N}^{N} \mathfrak{B}_i = \mathfrak{B}_N \quad \text{and} \quad \bigsqcup_{i=n}^{N} \mathfrak{B}_i = \mathfrak{B}_n \sqcup \bigsqcup_{i=n+1}^{N} \mathfrak{B}_i.$$

The concepts of subsumption and closure of boxes are illustrated in Figure 4.3.

We prove a few basic and intuitive properties about the closure of boxes and subsumption.

Lemma 30 — *Closure of boxes subsumes its components.* Let $\mathfrak{B}_0$ and $\mathfrak{B}_1$ be two boxes. We have that

$$\mathfrak{B}_0 \sqsubseteq \mathfrak{B}_0 \sqcup \mathfrak{B}_1 \quad \text{and} \quad \mathfrak{B}_1 \sqsubseteq \mathfrak{B}_0 \sqcup \mathfrak{B}_1.$$

Proof:

By the fact that $m \leq \max M$ for each $m \in M$ and $m \geq \min M$ for each $m \in M$. ■

Lemma 31. Let $\langle \bar{f}, f, z \rangle$ be a SoC distribution and $\mathfrak{B}_0$ be a bounding box of $\langle \bar{f}, f, z \rangle$. Each $\mathfrak{B}_1$, with $\mathfrak{B}_0 \sqsubseteq \mathfrak{B}_1$ is also a bounding box of $\langle \bar{f}, f, z \rangle$.

Proof:

Let $\mathfrak{B}_0 = \langle A_0, B_0, \overline{B}_0 \rangle$ and $\mathfrak{B}_1 = \langle A_1, B_1, \overline{B}_1 \rangle$. Since $\int_{\overline{B}_0} \bar{f} = \int_{[\underline{b}, \overline{b}]} \bar{f}$ and $\overline{B}_0 \subseteq \overline{B}_1$ we also have $\int_{\overline{B}_1} \bar{f} = \int_{\overline{B}_0} \bar{f}$. Analogously, with $A_0 \subseteq A_1$ and $B_0 \subseteq B_1$ we also have $\iint_{A_1 \times B_1} f = \iint_{A_0 \times B_0} f$. ∎

Using the closure of boxes allows us to define basic arithmetic operations on discrete SoC distributions similarly to conventional SoC distributions.

Definition 34. Let $\star \in \{+, -, \cdot, /\}$ and $\langle \overline{\mu}_0, \mu_0, \mathfrak{z}_0 \rangle$ be a discrete SoC distribution. We define

$$\langle \overline{\mu}_0, \mu_0, \mathfrak{z}_0 \rangle \star x := \begin{cases} \langle \overline{\mu}_0 \star \overline{\mu}_1, \mu_0 \star \mu_1, \mathfrak{z}_0 \star \mathfrak{z}_1 \rangle, & \text{if } x = \langle \overline{\mu}_1, \mu_1, \mathfrak{z}_1 \rangle \\ \langle \overline{\mu}_0 \star \mathrm{k}, \mu_0 \star \mathrm{k}, \mathfrak{z}_0 \star \mathrm{k} \rangle, & \text{if } x = \mathrm{k} \in \mathbb{R}. \end{cases}$$

Countable operations $\bigstar \in \{\sum, \prod\}$ are defined inductively in the usual way:

$$\begin{aligned} \bigstar_{i=N}^{N} \langle \overline{\mu}_i, \mu_i, \mathfrak{z}_i \rangle &= \langle \overline{\mu}_N, \mu_N, \mathfrak{z}_N \rangle \quad \text{and} \\ \bigstar_{i=n}^{N} \langle \overline{\mu}_i, \mu_i, \mathfrak{z}_i \rangle &= \langle \overline{\mu}_n, \mu_n, \mathfrak{z}_n \rangle \star \bigstar_{i=n+1}^{N} \langle \overline{\mu}_i, \mu_i, \mathfrak{z}_i \rangle. \end{aligned}$$

where the underlying box of the result is given by $\bigsqcup_{i=0}^{N} \mathfrak{B}_i$.

In the next sections we derive how to propagate bounding boxes along a task.

Successor Box While Discharging

The discharging scenario is the easier scenario since every SoC will become unsaturated and thus $\overline{B}$ will become empty. In addition, some SoCs that have support may be rendered unsafe by being mapped below the depletion threshold, thus we can potentially crop the successor bounding box from below.

Therefore, let us first assume that T is a discharging task. An important concept of the construction outlined in this section is the depletion boundary (Definition 17). Remember that the depletion boundary of a task T, is a line in the SoC space, that separates the SoCs that remain safe and the SoCs that are rendered unsafe under T (Lemma 15). For a visual example about the depletion boundary, see Example 4.

From the above we get the following: If the depletion boundary intersects the bounding box, the latter will be separated into two parts. Lemma 14 indicates that the SoC located on the leftmost intersection with the bounding box will become the smallest SoC that is mapped exactly onto the depletion threshold under T, since it exhibits the minimal bound charge level of all the SoCs on the intersection

of depletion boundary and bounding box. The successor of this SoC will provide new lower interval endpoints (i.e. the new bottom left corner) of the successor bounding box. The upper interval endpoints are simply propagated via $\mathbb{K}$, meaning they constitute the new top right corner of the successor box.

Luckily, we can split up this development into multiple steps by focussing on the saturated and non-saturated parts of a bounding box individually and join the results using closure operations, i.e. to compute the successor box of $\mathfrak{B} = \langle A, B, \overline{B}\rangle$, it suffices to compute the successor boxes of the "subboxes" $\langle \emptyset, \emptyset, \overline{B}\rangle$ and $\langle A, B, \emptyset\rangle$ individually.

Successor Of The Saturated Part. Let us first assume that the non-saturated part of $\mathfrak{B}$ is empty, and the entire support of the SoC distribution is represented in $\overline{B}$. Thus, we assume $\mathfrak{B}$ is of the form $\langle \emptyset, \emptyset, \overline{B}\rangle$ with $\overline{B} = [\overline{b}^{\uparrow}, \overline{b}^{\downarrow}]$. In this case, the depletion boundary of T (i.e. with target available charge level $\tilde{a} \coloneqq \underline{a}$) is given by $b_T(\underline{a})$, and it intersects with $\mathfrak{B}$ only if $b_T(\underline{a}) \in \overline{B}$ and the successor box will be determined by $\mathbb{K}_T[\underline{a}; b_T(\underline{a})]$ as well as $\mathbb{K}_T[\underline{a}; \overline{b}^{\downarrow}]$. If the depletion boundary does not intersect with $\overline{B}$, then either $b_T(\underline{a}) < \overline{b}^{\uparrow}$, which results in the successor box being captured by $\mathbb{K}_T[\underline{a}; \overline{b}^{\uparrow}]$ and $\mathbb{K}_T[\underline{a}; \overline{b}^{\downarrow}]$; or $b_T(\underline{a}) > \overline{b}^{\downarrow}$, which will lead to an empty successor box altogether.

Definition 35. Let $T = (\Delta, \ell) \in \mathbb{T}$ be a discharging task and $\mathfrak{B} = \langle \emptyset, \emptyset, \overline{B}\rangle$ be a box with non-empty $\overline{B} = [\overline{b}^{\uparrow}, \overline{b}^{\downarrow}]$. Furthermore, let

$$[a^{\min}_{\text{mid}}; b^{\min}_{\text{mid}}] \coloneqq \mathbb{K}_T[\underline{a}; \overline{b}^{\uparrow}], \quad [\bullet; b_{\text{hit}}] \coloneqq \mathbb{K}_T[\underline{a}; b_T(\underline{a})] \text{ and } \quad [a^{\max}_{\text{mid}}; b^{\max}_{\text{mid}}] \coloneqq \mathbb{K}_T[\underline{a}; \overline{b}^{\downarrow}].$$

Then the $\underline{\overline{\mathbb{K}}}$-successor of $\overline{B}$ is defined by

$$\underline{\overline{\mathbb{K}}}_T \overline{B} \coloneqq \begin{cases} \langle \emptyset, \emptyset, \emptyset\rangle, & \text{if } b_T(\underline{a}) > \overline{b}^{\downarrow} \\ \langle [a^{\min}_{\text{mid}}, a^{\max}_{\text{mid}}], [b^{\min}_{\text{mid}}, b^{\max}_{\text{mid}}], \emptyset\rangle, & \text{if } b_T(\underline{a}) < \overline{b}^{\uparrow} \\ \langle [\underline{a}, a^{\max}_{\text{mid}}], [b_{\text{hit}}, b^{\max}_{\text{mid}}], \emptyset\rangle, & \text{if } b_T(\underline{a}) \in \overline{B} \end{cases}.$$

Figure 4.4 illustrates every case of Definition 35 separately. We prove that the successor box in this case is indeed a bounding box of the successor SoC distribution.

Lemma 32. Given a discharging task $T \in \mathbb{T}$, a SoC distribution $\langle \tilde{f}, f, z\rangle$ and its bounding box $\mathfrak{B}$ of the form $\langle \emptyset, \emptyset, \overline{B}\rangle$, $\underline{\overline{\mathbb{K}}}_T \overline{B}$ is indeed a bounding box of $\underline{\overline{\mathbb{K}}}_T \langle \tilde{f}, f, z\rangle$.

Proof:

> By properties of the depletion boundary (Lemmas 14 and 15) as well as the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1), it is straightforward to deduce that the whole support of $\underline{\overline{\mathbb{K}}}_T \langle \tilde{f}, f, z\rangle$ is captured by $\underline{\overline{\mathbb{K}}}_T \overline{B}$. ■

Successor Of The Non-Saturated Part. Let us assume next that $\mathfrak{B}$ is of the form $\langle A, B, \emptyset\rangle$ with non-empty $A = [a^{\uparrow}, a^{\downarrow}]$ and $B = [b^{\uparrow}, b^{\downarrow}]$. Given Lemma 13 we know

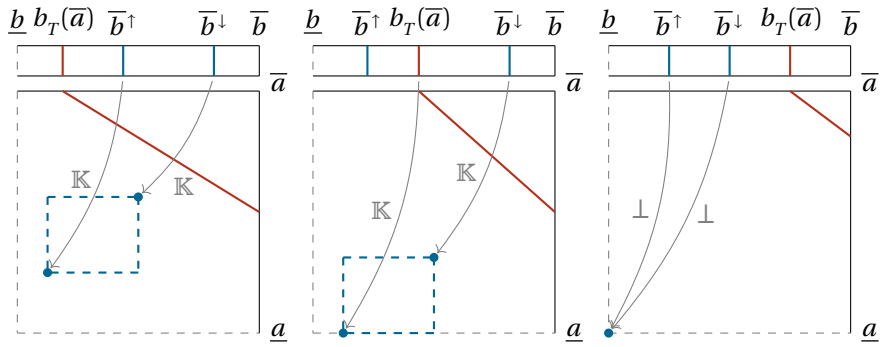


Figure 4.4: A depiction of $\overline{\mathbb{K}}_T\,\overline{B}$ in a discharging scenario, given in Definition 35. The solid blue lines define the current box, while the dashed blue lines represent the successor box. The depletion boundary is highlighted in red. The grey arrows indicate the $\mathbb{K}$ mappings necessary to compte the successor box.

that the depletion boundary is strictly monotonically decreasing, thus restricting the ways the boundary can intersect with the box. In fact, the boundary can only intersect the box at the left or top side while simultaneously attaining a minimal bound charge level.

Definition 36. Let $T = (\Delta, \ell) \in \mathbb{T}$ be a discharging task and $\mathfrak{B} = \langle A, B, \emptyset \rangle$ be a box with $A = [a^{\uparrow}, a^{\downarrow}]$ and $B = [b^{\uparrow}, b^{\downarrow}]$. Furthermore, let

$$[\bullet; b^{\text{left}}_{\text{hit}}] := \mathbb{K}_T[a_T(b^{\uparrow}); b^{\uparrow}], \quad [a^{\text{min}}_{\text{mid}}; b^{\text{min}}_{\text{mid}}] := \mathbb{K}_T[a^{\uparrow}; b^{\uparrow}]$$
$$[\bullet; b^{\text{top}}_{\text{hit}}] := \mathbb{K}_T[a^{\downarrow}; b_T(a^{\downarrow})], \quad [a^{\text{max}}_{\text{mid}}; b^{\text{max}}_{\text{mid}}] := \mathbb{K}_T[a^{\downarrow}; b^{\downarrow}]$$

Then the $\overline{\mathbb{K}}$-successor of A and B combined, is defined by

$$\overline{\mathbb{K}}_T[A; B] := \begin{cases} \langle \emptyset, \emptyset, \emptyset \rangle, & \text{if } a_T(b^{\downarrow}) > a^{\downarrow} \\ \langle [a^{\text{min}}_{\text{mid}}, a^{\text{max}}_{\text{mid}}], [b^{\text{min}}_{\text{mid}}, b^{\text{max}}_{\text{mid}}], \emptyset \rangle, & \text{if } a_T(b^{\uparrow}) < a^{\uparrow} \\ \langle [\underline{a}, a^{\text{max}}_{\text{mid}}], [b^{\text{top}}_{\text{hit}}, b^{\text{max}}_{\text{mid}}], \emptyset \rangle, & \text{if } b_T(a^{\downarrow}) \in B \\ \langle [\underline{a}, a^{\text{max}}_{\text{mid}}], [b^{\text{left}}_{\text{hit}}, b^{\text{max}}_{\text{mid}}], \emptyset \rangle, & \text{if } a_T(b^{\uparrow}) \in A \end{cases}.$$

Figure 4.5 illustrates every case of Definition 36 separately. We prove that the successor box in this case is indeed a bounding box of the successor SoC distribution.

Lemma 33. Given a discharging task $T \in \mathbb{T}$, a SoC distribution $\langle \bar{f}, f, z \rangle$ and its bounding box $\mathfrak{B} = \langle A, B, \emptyset \rangle$, $\overline{\mathbb{K}}_T[A; B]$ is a bounding box of $\overline{\mathbb{K}}_T \langle \bar{f}, f, z \rangle$.

Proof:

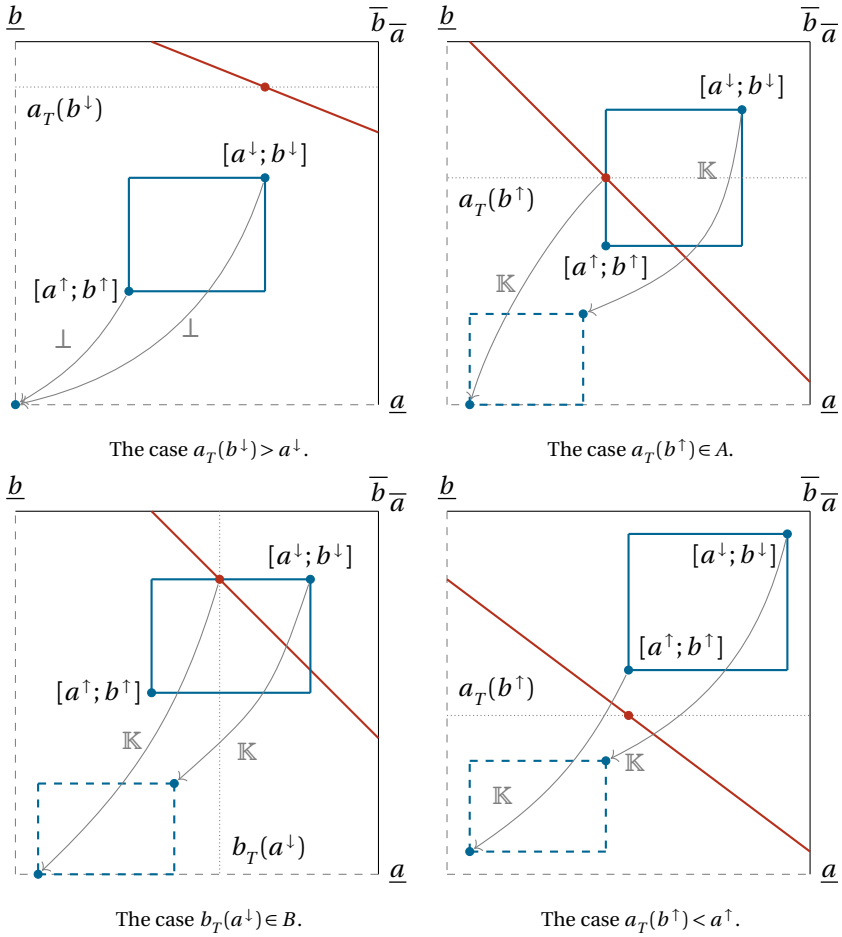


Figure 4.5: A depiction of $\underline{\overline{\mathbb{K}}}_T[A;B]$ in a discharging scenario, given in Definition 36. The solid blue lines define the current box, while the dashed blue lines represent the successor box. The depletion boundary is highlighted in red. The red dot indicates the relevant SoC on the depletion boundary we perform the case distinction on. The grey arrows indicate the $\mathbb{K}$-mappings necessary to compute the successor bounding box.

> Again, by properties of the depletion boundary (Lemmas 14 and 15) as well as the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1), it is straightforward to deduce that the whole support of $\underline{\overline{\mathbb{K}}}_T\langle \bar{f}, f, z\rangle$ is captured by $\underline{\overline{\mathbb{K}}}_T[A;B]$. ■

Successor Box While Charging

Let us now focus on the case where T is a charging task.

In this case, non-saturated SoCs may become saturated, thus after a charging task is performed, the support in the non-saturated part may contribute to the saturated portion of $\underline{\overline{\mathbb{K}}}_T\mathfrak{B}$, potentially enlarging it compared to the previous box. Dual to the discharging case, we are interested in the intersection points of the

box with the target boundary with target ($\tilde{a} \coloneqq \overline{a}$), i.e. the saturation boundary (Definition 17 and Example 4). The boundary potentially separates the box into a part that is saturated by T and a part that remains unsaturated after powering T (Lemma 15). Again, the intersection point exhibiting the least bound charge level potentially contributes to the left endpoint of the successor box's saturated part.

The treatment of the saturated part is more involved as in the discharging scenario, since some SoCs potentially leave the saturated part due to diffusion, but become saturated again, while other SoCs remain saturated throughout the entire task. In order to keep the boxes as tight as possible we distinguish these cases. Lastly, we need to be careful when parts of the non-saturated portion become saturated, which implicitly means, the saturation limit is hit. This requires the use of $\overline{\underline{\mathbb{K}}}$-mappings. Due to the problems of determining the exact saturation time points already discussed in Section 3.3, we use the approximation operators $\overline{\underline{\mathbb{K}}}^{\downarrow}$ and $\overline{\underline{\mathbb{K}}}^{\uparrow}$ accordingly. In essence, whenever we determine an upper interval endpoint of a successor box, we potentially use $\overline{\underline{\mathbb{K}}}^{\downarrow}$, in order to over-approximate the endpoint, and dually we use $\overline{\underline{\mathbb{K}}}^{\uparrow}$ for lower interval endpoints for under-approximations.

Successor Of The Saturated Part. Let us again first assume that the non-saturated part of $\mathfrak{B}$ is empty, and the entire support of the SoC distribution is represented in $\overline{B}$. Thus, we assume $\mathfrak{B}$ is of the form $\langle \emptyset, \emptyset, \overline{B} \rangle$ with $\overline{B} = [\overline{b}^{\uparrow}, \overline{b}^{\downarrow}]$. The saturation boundary of T is given by $b_T(\overline{a})$, and it intersects with $\mathfrak{B}$ only if $b_T(\overline{a}) \in \overline{B}$. The left endpoint of the successor box's saturated part is then determined by $\mathbb{K}_T[\overline{a}; b_T(\overline{a})]$ as is the top-right corner of the non-saturated part. The bottom left corner of the non-saturated part is contributed by $\mathbb{K}_T[\overline{a}; b_T(\overline{b}^{\uparrow})]$, while the right endpoint of the saturated part is given by $\overline{\underline{\mathbb{K}}}_T[\overline{a}; \overline{b}^{\downarrow}]$ or $\mathbb{K}^{\mathsf{sat}}_T[\overline{a}; \overline{b}^{\downarrow}]$ depending on whether the diffusion is overpowered by T or not. If the saturation boundary does not intersect $\overline{B}$, we either have the case that the successor ends up with an empty saturated part, or an empty non-saturated part. In the latter case, one or both endpoints may have to be propagated using $\mathbb{K}^{\mathsf{sat}}$ or $\overline{\underline{\mathbb{K}}}$, depending on whether the diffusion is overpowered by T.

Thus, one central value we are interested in, is the least bound charge level of a saturated SoC, that is able to withstand the diffusion and stay saturated, given a charging load ℓ. This value can be easily derived by rearranging the expression of $\ell_{\mathsf{sat}}(b)$ for b, which results in b^{sat}_{ℓ} according to Definition 13.

Additionally, we can somewhat reduce the number of case distinctions using the fact that the least bound charge level able to overcome the diffusion b^{sat}_{ℓ} is larger than $b_T(\overline{a})$.

Definition 37. Let $T = (\Delta, \ell) \in \mathbb{T}$ be a charging task and $\mathfrak{B} = \langle \emptyset, \emptyset, \overline{B} \rangle$ be a box with $\overline{B} = [\overline{b}^{\uparrow}, \overline{b}^{\downarrow}]$. Furthermore, let

$$[\bullet; b^{\mathsf{min}}_{\mathsf{sat}}] \coloneqq \overline{\underline{\mathbb{K}}}^{\uparrow}_T[\overline{a}; \overline{b}^{\uparrow}], \quad [a^{\mathsf{min}}_{\mathsf{mid}}; b^{\mathsf{min}}_{\mathsf{mid}}] \coloneqq \mathbb{K}_T[\overline{a}; \overline{b}^{\uparrow}], \quad [\bullet; b_{\mathsf{ret}}] \coloneqq \mathbb{K}_T[\overline{a}; b_T(\overline{a})],$$
$$[\bullet; b^{\mathsf{max}}_{\mathsf{sat}}] \coloneqq \overline{\underline{\mathbb{K}}}^{\downarrow}_T[\overline{a}; \overline{b}^{\downarrow}], \quad [a^{\mathsf{max}}_{\mathsf{mid}}; b^{\mathsf{max}}_{\mathsf{mid}}] \coloneqq \mathbb{K}_T[\overline{a}; \overline{b}^{\downarrow}].$$

Then the $\overline{\underline{\mathbb{K}}}$-successor of $\mathfrak{B}$ is defined by

$$\overline{\underline{\mathbb{K}}}_T \overline{B} \coloneqq \begin{cases} \langle \emptyset, \emptyset, \left[\mathrm{B}_\Delta^{\mathsf{sat}}(\overline{b}^\uparrow), \mathrm{B}_\Delta^{\mathsf{sat}}(\overline{b}^\downarrow) \right] \rangle, & \text{if } b_\ell^{\mathsf{sat}} < \overline{b}^\uparrow \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, a_{\mathsf{mid}}^{\mathsf{max}}], [b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{mid}}^{\mathsf{max}}], \emptyset \rangle, & \text{if } b_T(\overline{a}) > \overline{b}^\downarrow \\ \langle \emptyset, \emptyset, \left[b_{\mathsf{sat}}^{\mathsf{min}}, b_{\mathsf{sat}}^{\mathsf{max}} \right] \rangle, & \text{if } b_T(\overline{a}) < \overline{b}^\uparrow \ \wedge \ b_\ell^{\mathsf{sat}} > \overline{b}^\downarrow \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, \overline{a}], [b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{ret}}], \left[b_{\mathsf{ret}}, \mathrm{B}_\Delta^{\mathsf{sat}}(\overline{b}^\downarrow) \right] \rangle, & \text{if } b_T(\overline{a}) \in \overline{B} \ \wedge \ b_\ell^{\mathsf{sat}} \in \overline{B} \\ \langle \emptyset, \emptyset, \left[b_{\mathsf{sat}}^{\mathsf{min}}, \mathrm{B}_\Delta^{\mathsf{sat}}(\overline{b}^\downarrow) \right] \rangle, & \text{if } b_T(\overline{a}) < \overline{b}^\uparrow \ \wedge \ b_\ell^{\mathsf{sat}} \in \overline{B} \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, \overline{a}], [b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{ret}}], \left[b_{\mathsf{ret}}, b_{\mathsf{sat}}^{\mathsf{max}} \right] \rangle, & \text{if } b_T(\overline{a}) \in \overline{B} \ \wedge \ b_\ell^{\mathsf{sat}} > \overline{b}^\downarrow \end{cases}.$$

Figure 4.6 illustrates all the cases of Definition 37 separately. We prove that the successor box in this scenario is indeed a bounding box of the successor SoC distribution.

Lemma 34. Given a charging task $T \in \mathbb{T}$, a SoC distribution $\langle \bar{f}, f, z \rangle$ and its bounding box $\mathfrak{B}$ of the form $\langle \emptyset, \emptyset, \overline{B} \rangle$, $\overline{\underline{\mathbb{K}}}_T \overline{B}$ is indeed a bounding box of $\overline{\underline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$.

Proof:

> By properties of the saturation boundary (Lemmas 14 and 15) as well as the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1), it is straightforward to deduce that the whole support of $\overline{\underline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$ is captured by $\overline{\underline{\mathbb{K}}}_T \overline{B}$. ■

Successor Of The Non-Saturated Part. Similar to the discharging case, let us assume next that $\mathfrak{B}$ is of the form $\langle A, B, \emptyset \rangle$ with $A = [a^\uparrow, a^\downarrow]$ and $B = [b^\uparrow, b^\downarrow]$. Given Lemma 13 we know that the saturation boundary is strictly monotonically decreasing, which restricts its intersection points with the box, analogously to what we have seen before.

Definition 38. Let, $T = (\Delta, \ell) \in \mathbb{T}$ be a charging task and $\mathfrak{B} = \langle A, B, \emptyset \rangle$ be a box with $A = [a^\uparrow, a^\downarrow]$ and $B = [b^\uparrow, b^\downarrow]$. Furthermore, let

$$[\bullet; b_{\mathsf{sat}}^{\mathsf{min}}] \coloneqq \overline{\underline{\mathbb{K}}}_T^\uparrow [\overline{a}; \overline{b}^\uparrow] \quad [a_{\mathsf{mid}}^{\mathsf{min}}; b_{\mathsf{mid}}^{\mathsf{min}}] \coloneqq \mathbb{K}_T [a^\uparrow; b^\uparrow] \quad [\bullet; b_{\mathsf{hit}}^{\mathsf{top}}] \coloneqq \mathbb{K}_T [a^\downarrow; b_T(a^\downarrow)]$$
$$[\bullet; b_{\mathsf{sat}}^{\mathsf{max}}] \coloneqq \overline{\underline{\mathbb{K}}}_T^\downarrow [\overline{a}; \overline{b}^\downarrow] \quad [a_{\mathsf{mid}}^{\mathsf{max}}; b_{\mathsf{mid}}^{\mathsf{max}}] \coloneqq \mathbb{K}_T [a^\downarrow; b^\downarrow] \quad [\bullet; b_{\mathsf{hit}}^{\mathsf{left}}] \coloneqq \mathbb{K}_T [a_T(b^\uparrow); b^\uparrow]$$

Then the $\overline{\underline{\mathbb{K}}}$-successor of A and B, is defined by

$$\overline{\underline{\mathbb{K}}}_T [A; B] \coloneqq \begin{cases} \langle \emptyset, \emptyset, \left[b_{\mathsf{sat}}^{\mathsf{min}}, b_{\mathsf{sat}}^{\mathsf{max}} \right] \rangle, & \text{if } a_T(b^\uparrow) < a^\uparrow \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, a_{\mathsf{mid}}^{\mathsf{max}}], [b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{mid}}^{\mathsf{max}}], \emptyset \rangle, & \text{if } a_T(b^\downarrow) > a^\downarrow \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, \overline{a}], \left[b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{hit}}^{\mathsf{top}} \right], \left[b_{\mathsf{hit}}^{\mathsf{top}}, b_{\mathsf{sat}}^{\mathsf{max}} \right] \rangle, & \text{if } b_T(a^\downarrow) \in B \\ \langle [a_{\mathsf{mid}}^{\mathsf{min}}, \overline{a}], [b_{\mathsf{mid}}^{\mathsf{min}}, b_{\mathsf{hit}}^{\mathsf{left}}], \left[b_{\mathsf{hit}}^{\mathsf{left}}, b_{\mathsf{sat}}^{\mathsf{max}} \right] \rangle, & \text{if } a_T(b^\uparrow) \in A \end{cases}.$$

Figure 4.7 illustrates all the cases of Definition 38 separately. We prove that the successor box in this case is indeed a bounding box of the successor SoC distribution.

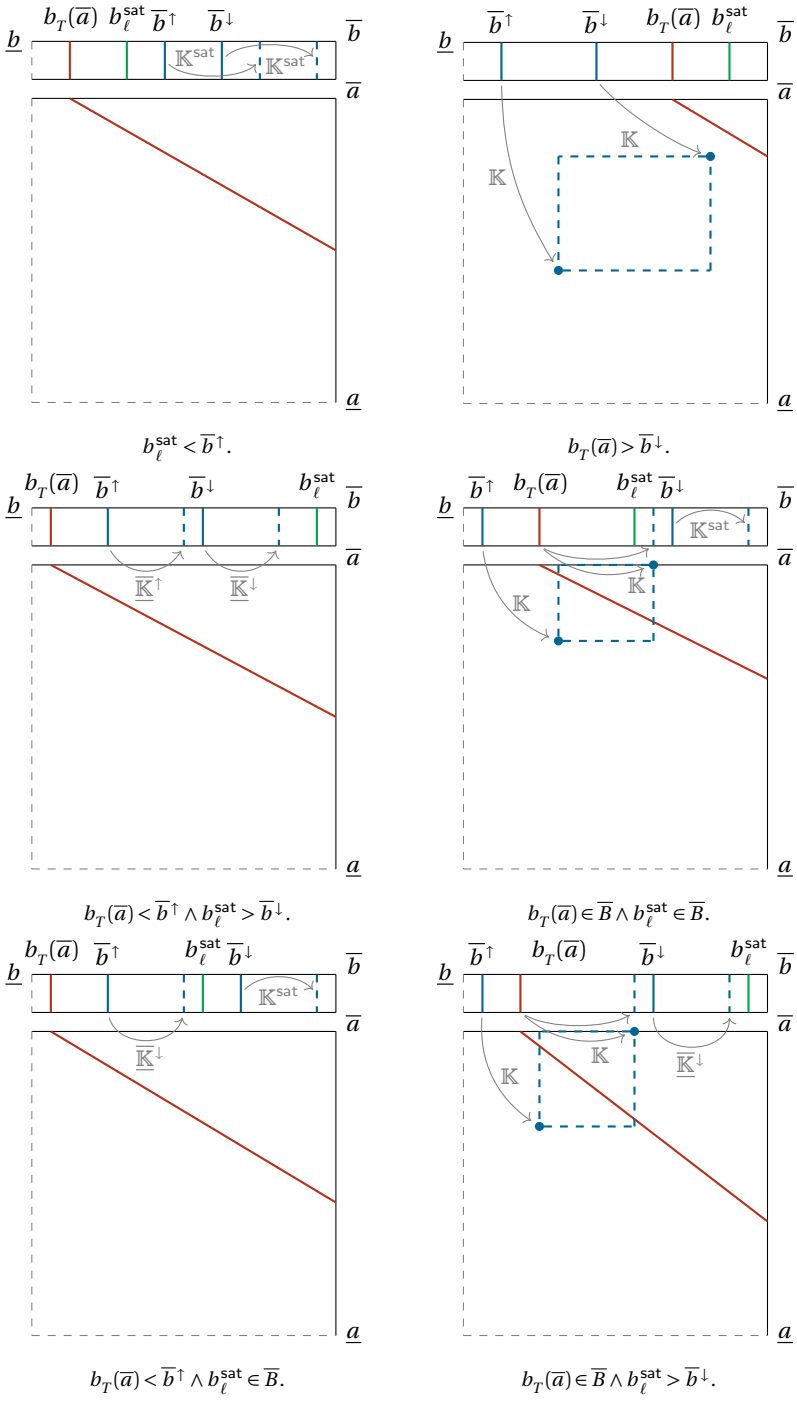


Figure 4.6: A depiction of $\underline{\overline{\mathbb{K}}}_T \overline{B}$ in a charging scenario, given in Definition 38. The solid blue lines define the current box, while the dashed blue lines represent the successor box. The saturation boundary and the least bound charge staying saturated are highlighted in red and green, respectively. The grey arrows indicate the mappings necessary to compute the successor box.

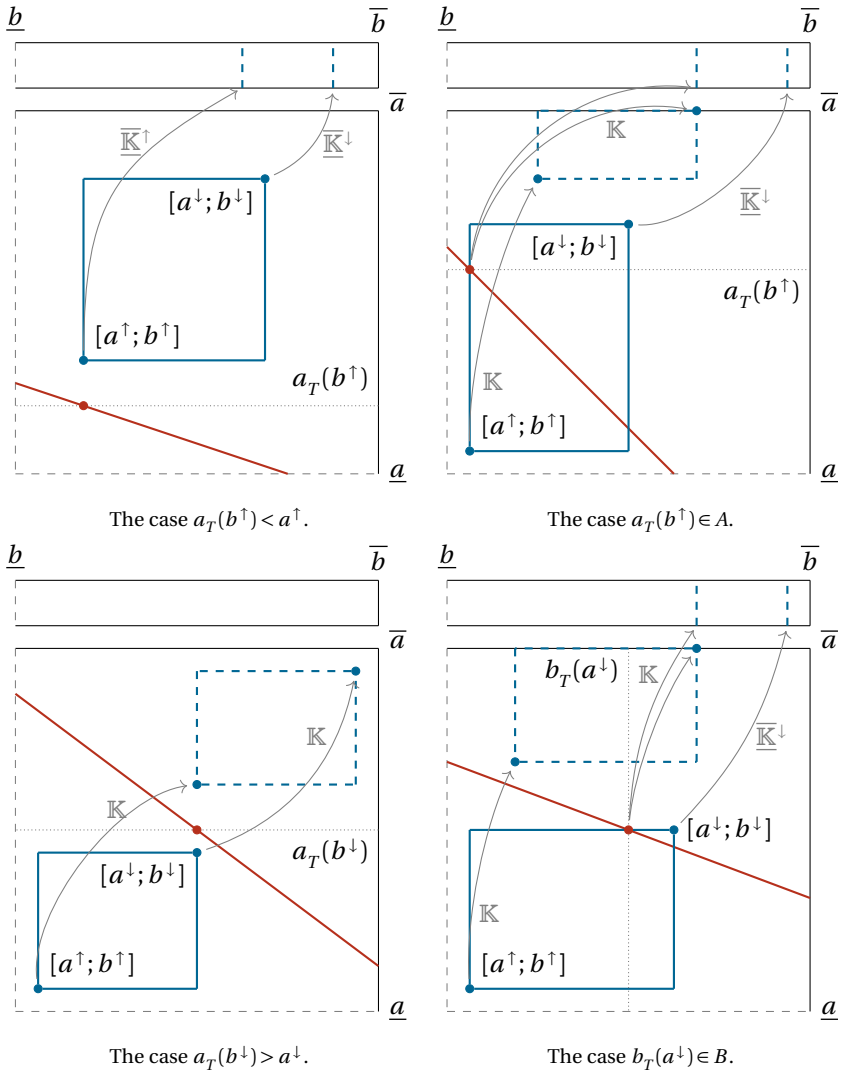


Figure 4.7: A depiction of $\underline{\overline{\mathbb{K}}}_T[A; B]$ in a charging scenario, given in Definition 38. The solid blue lines define the current box, while the dashed blue lines represent the successor box. The saturation boundary is highlighted in red. The red dot visualizes the SoC on the saturation boundary responsible for the entire case distinction. The grey arrows indicate the mappings necessary to compute the successor box.

Lemma 35. Given a charging task $T \in \mathbb{T}$, a SoC distribution $\langle \bar{f}, f, z \rangle$ and its bounding box $\mathfrak{B} = \langle A, B, \emptyset \rangle$, $\underline{\overline{\mathbb{K}}}_T[A; B]$ is a bounding box of $\underline{\overline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$.

Proof:

Again, by properties of the saturation boundary (Lemmas 14 and 15) as well as the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1), it is straightforward to deduce that the whole support of $\underline{\overline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$ is captured by $\underline{\overline{\mathbb{K}}}_T[A; B]$. ■

Successor Box While Resting

We can further simplify the computation of successor boxes in the case of resting, meaning the battery undergoes a zero load task $(\Delta, 0)$, for some positive duration $\Delta > 0$. In this special case, we do not need to worry about any target boundary as depletion as well as saturation do not occur during periods of resting.

Definition 39. Let, $T = (\Delta, 0) \in \mathbb{T}$ be a resting task and $\mathfrak{B} = \langle A, B, \overline{B} \rangle$ be a box with $A = [a^{\uparrow}, a^{\downarrow}]$, $B = [b^{\uparrow}, b^{\downarrow}]$ and $\overline{B} = [\overline{b}^{\uparrow}, \overline{b}^{\downarrow}]$. Furthermore, let

$$[a^{\max}_{\mathsf{sat}}; b^{\max}_{\mathsf{sat}}] \coloneqq \mathbb{K}_T[\overline{a}; \overline{b}^{\downarrow}], \quad [a^{\max}_{\mathsf{mid}}; b^{\max}_{\mathsf{mid}}] \coloneqq \mathbb{K}_T[a^{\downarrow}; b^{\downarrow}]$$
$$[a^{\min}_{\mathsf{sat}}; b^{\min}_{\mathsf{sat}}] \coloneqq \mathbb{K}_T[\overline{a}; \overline{b}^{\uparrow}], \quad [a^{\min}_{\mathsf{mid}}; b^{\min}_{\mathsf{mid}}] \coloneqq \mathbb{K}_T[a^{\uparrow}; b^{\uparrow}]$$

Then the $\underline{\overline{\mathbb{K}}}$-successor of A and B combined, is defined by

$$\underline{\overline{\mathbb{K}}}_T[A; B] \coloneqq \langle [a^{\min}_{\mathsf{mid}}, a^{\max}_{\mathsf{mid}}], [b^{\min}_{\mathsf{mid}}, b^{\max}_{\mathsf{mid}}], \emptyset \rangle,$$
$$\underline{\overline{\mathbb{K}}}_T \overline{B} \coloneqq \langle \big[a^{\min}_{\mathsf{sat}}, a^{\max}_{\mathsf{sat}}\big], \big[b^{\min}_{\mathsf{sat}}, b^{\max}_{\mathsf{sat}}\big], \emptyset \rangle.$$

Lemma 36. Let $T \in \mathbb{T}$ be a resting task, and $\langle \bar{f}, f, z \rangle$ be a SoC distribution. If its bounding box is of the form $\mathfrak{B} = \langle A, B, \emptyset \rangle$, then $\underline{\overline{\mathbb{K}}}_T[A; B]$ is a bounding box of $\underline{\overline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$. If the bounding box is of the form $\mathfrak{B} = \langle \emptyset, \emptyset, \overline{B} \rangle$, then $\underline{\overline{\mathbb{K}}}_T \overline{B}$ is a bounding box of $\underline{\overline{\mathbb{K}}}_T \langle \bar{f}, f, z \rangle$.

Proof:

By the fact that $\mathbb{K}$ preserves $\leq$ (Corollary 1). ■

Figure 4.8 summarizes Definition 39 visually.

For any scenario, whether it is charging, discharging or resting we receive the final successor box as follows.

Definition 40. Let $T \in \mathbb{T}$ be any task and $\mathfrak{B} = \langle A, B, \overline{B} \rangle$. The $\underline{\overline{\mathbb{K}}}$-successor of $\mathfrak{B}$ is defined by

$$\underline{\overline{\mathbb{K}}}_T \mathfrak{B} \coloneqq \underline{\overline{\mathbb{K}}}_T[A; B] \sqcup \underline{\overline{\mathbb{K}}}_T \overline{B}.$$

Finally, for charging, discharging and resting scenarios we define that the successor of an empty box remains empty, since it effectively represents a fully depleted SoC distribution.

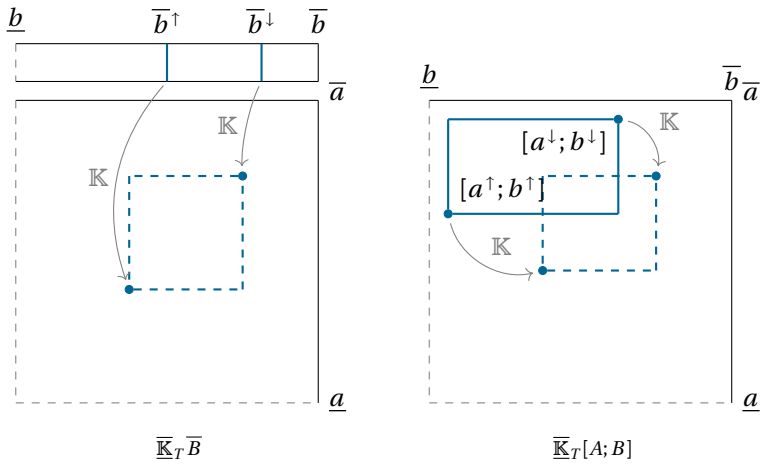


Figure 4.8: Left: A depiction of $\overline{\mathbb{K}}_T\,\overline{B}$ in case of T being a resting task, with the current box indicated with solid blue lines and the successor box being depicted in dashed blue lines. The grey arrows indicate the $\mathbb{K}$-mappings that are performed. The box will surely leave the saturated portion, and no depletion will occur. **Right:** A depiction of $\overline{\mathbb{K}}_T[A; B]$ with T being a resting task.

Definition 41. For any task $T \in \mathbb{T}$ we define $\overline{\mathbb{K}}$ invariant to empty boxes as

$$\overline{\mathbb{K}}_T\langle\emptyset,\emptyset,\emptyset\rangle := \langle\emptyset,\emptyset,\emptyset\rangle.$$

Lemma 37. The empty box $\langle\emptyset,\emptyset,\emptyset\rangle$ is a bounding box for any SoC distribution $\langle\bar{f}, f, z\rangle$ with $z = 1$.

Proof:

> The SoC distribution exhibits no probability mass through f nor $\bar{f}$, which coincides with the empty bounding box, since all intervals are empty and integrals amount to 0. ■

Finally, using all the preceding lemmas it is quite straightforward to see that $\overline{\mathbb{K}}$ actually computes valid successor boxes.

Lemma 38. Let $T \in \mathbb{T}$ be any task, $\langle\bar{f}, f, z\rangle$ be a SoC distribution and $\mathfrak{B}$ its associated bounding box. We have that $\overline{\mathbb{K}}_T\,\mathfrak{B}$ is a bounding box of $\overline{\mathbb{K}}_T\langle\bar{f}, f, z\rangle$.

Proof:

> We first assume that $\mathfrak{B}$ is not completely empty. If T is a discharging task we deduce the claim by the preceding Lemmas 33 and 32, as well as Lemma 30.

Similarly, for charging tasks we get to the claim with the Lemmas 35 and 34, as well as Lemma 30. Finally, for resting tasks, we use Lemmas 36, and again 30. If $\mathfrak{B}$ is empty then consequently $\langle \bar{f}, f, z \rangle$ is a fully depleted SoC distribution. Lemma 37 immediately establishes the claim. ■

Lastly, the only remaining thing left to do is define is how to compute the successor box with respect to a noisy task and its discretized counterparts. Essentially, we compute the box of each load instance within the support of the load density of the noisy task. Using the closure of boxes we are able to formalize this concept quite elegantly and concisely, for discretized noisy tasks, which ensures that the computed boxes stay as small as possible. For arbitrary noisy tasks, especially those with continuous load densities, this requires the computation of the convex hull over an uncountable number of boxes, hence it is undefined.

Definition 42. Let $T = (\Delta, \gamma)$ be a discrete noisy task with finite support. Furthermore, let D be a (possibly discrete) SoC distribution and $\mathfrak{B}$ be an associated bounding box. Then, the $\overline{\underline{\mathbb{K}}}$-successor box of D according to T is given by

$$\overline{\underline{\mathbb{K}}}_T \mathfrak{B} \coloneqq \bigsqcup_{\ell \in \operatorname{supp}(\gamma)} \overline{\underline{\mathbb{K}}}_{(\Delta,\ell)} \mathfrak{B}.$$

4.4 Algorithm

With all of the above in place, we are ready to formalize an algorithm that tracks an initial SoC distribution $\langle \bar{f}, f, z \rangle$ along a Markov Task Process $\boldsymbol{M}$ up to a time horizon $\bar{t}$. A pseudo-code formulation is given in Algorithm 2.

Initialization. We start the algorithm by discretizing the input SoC distribution $\langle \bar{f}, f, z \rangle$ and annotating it with the starting time 0 (Line 1). Line 2 introduces an empty priority list todo that serves as a worklist, to keep track of the time frontier, although this step is not necessary for correctness but for efficiency, since it allows for merging paths as early as possible. We scale the discretized SoC distributions to subdistributions, and additionally annotate them with an MTP state $\mathfrak{s}$ for each state with non-zero probability with respect to $\pi(\mathfrak{s})$ inside a foreach-loop (Lines 3–6).

Propagation. The propagation phase is contained in the repeat-until loop, starting in Line 7. The discrete SoC subdistribution pair having the smallest time point annotation is popped from the worklist (Line 8). We grab the task according to the MTP state the distributions are annotated with (Line 9), and adjust its duration in case we would exceed the time horizon in the following line. Next, the appropriate successors are computed in Lines 11–12, adjusting the annotated time to coincide with the end of the task. The outer foreach-loop in Line 13 starts off by moving the scaled successors (Line 14) over to the next MTP state $\tilde{\mathfrak{s}}$ according to the transition function $P(\bullet, \tilde{\mathfrak{s}})$ adjusting the annotated state in the process. The inner foreach-loop (Line 16) then selects every distribution already in the worklist, having the same time and state annotation, merging them componentwise with the distributions we are currently processing, removing them in the process (Lines 17–19). Note that also the bounding box is implicitly adjusted according to Definition 34

Input : A SoC distribution $\langle \bar{f}, f, z \rangle$ with bounding box $\mathfrak{B}$, an MTP $\boldsymbol{M} = (\mathfrak{S}, P, \pi, \mathsf{task})$, a time horizon $\bar{t}$, and a grid size N
Output : Two discrete SoC distributions $\langle \overline{\mu}^{\uparrow}_{\bar{t}}, \mu^{\uparrow}_{\bar{t}}, \mathfrak{z}^{\uparrow}_{\bar{t}} \rangle$ and $\langle \overline{\mu}^{\downarrow}_{\bar{t}}, \mu^{\downarrow}_{\bar{t}}, \mathfrak{z}^{\downarrow}_{\bar{t}} \rangle$

$D^{\downarrow}_{(0,\bullet)}, D^{\uparrow}_{(0,\bullet)} \coloneqq$ discretizations of $\langle \bar{f}, f, z \rangle$ with box $\mathfrak{B}$
todo $\coloneqq$ empty priority list with order $D^{\bullet}_{(t_0,\bullet)} \leq_t D^{\bullet}_{(t_1,\bullet)} \coloneqq t_0 \leq t_1$
foreach $\mathfrak{s} \in \mathfrak{S}$ *with* $\pi(\mathfrak{s}) \neq 0$ **do**
    $D^{\downarrow}_{(0,\mathfrak{s})} \coloneqq D^{\downarrow}_{(0,\bullet)} \cdot \pi(\mathfrak{s})$ with box $\mathfrak{B}_{(0,\mathfrak{s})}$
    $D^{\uparrow}_{(0,\mathfrak{s})} \coloneqq D^{\uparrow}_{(0,\bullet)} \cdot \pi(\mathfrak{s})$ with box $\mathfrak{B}_{(0,\mathfrak{s})}$
    todo.`insert`$(D^{\downarrow}_{(0,\mathfrak{s})}, D^{\uparrow}_{(0,\mathfrak{s})})$
repeat
    $D^{\downarrow}_{(t,\mathfrak{s})}, D^{\uparrow}_{(t,\mathfrak{s})} \coloneqq$ todo.`pop()` with box $\mathfrak{B}_{(t,\mathfrak{s})}$
    $(\Delta, g) \coloneqq \mathsf{task}(\mathfrak{s})$
    if $t + \Delta > \bar{t}$ **then** $\Delta \coloneqq \bar{t} - t$
    $D^{\downarrow}_{(t+\Delta,\mathfrak{s})} \coloneqq \underline{\overline{\mathbb{K}}}^{\downarrow}_{(\Delta,g)} D^{\downarrow}_{(t,\mathfrak{s})}$ with box $\mathfrak{B}_{(t+\Delta,\mathfrak{s})} \coloneqq \underline{\overline{\mathbb{K}}}_{(\Delta,g)} \mathfrak{B}_{(t,\mathfrak{s})}$
    $D^{\uparrow}_{(t+\Delta,\mathfrak{s})} \coloneqq \underline{\overline{\mathbb{K}}}^{\uparrow}_{(\Delta,g)} D^{\uparrow}_{(t,\mathfrak{s})}$ with box $\mathfrak{B}_{(t+\Delta,\mathfrak{s})}$
    foreach $\tilde{\mathfrak{s}} \in \mathfrak{S}$ *with* $P(\mathfrak{s}, \tilde{\mathfrak{s}}) \neq 0$ **do**
        $D^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq D^{\downarrow}_{(t+\Delta,\mathfrak{s})} \cdot P(\mathfrak{s}, \tilde{\mathfrak{s}})$ with box $\mathfrak{B}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq \mathfrak{B}_{(t+\Delta,\mathfrak{s})}$
        $D^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq D^{\uparrow}_{(t+\Delta,\mathfrak{s})} \cdot P(\mathfrak{s}, \tilde{\mathfrak{s}})$ with box $\mathfrak{B}_{(t+\Delta,\tilde{\mathfrak{s}})}$
        foreach $\widetilde{D}^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})}, \widetilde{D}^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})} \in$ todo *with box* $\widetilde{\mathfrak{B}}$ **do**
            $D^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq D^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})} + \widetilde{D}^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})}$ with box $\mathfrak{B}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq \mathfrak{B}_{(t+\Delta,\tilde{\mathfrak{s}})} \sqcup \widetilde{\mathfrak{B}}$
            $D^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})} \coloneqq D^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})} + \widetilde{D}^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})}$ with box $\mathfrak{B}_{(t+\Delta,\tilde{\mathfrak{s}})}$
            todo.`remove`$(\widetilde{D}^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})}, \widetilde{D}^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})})$
        todo.`insert`$(D^{\downarrow}_{(t+\Delta,\tilde{\mathfrak{s}})}, D^{\uparrow}_{(t+\Delta,\tilde{\mathfrak{s}})})$
until $\forall D^{\bullet}_{(t,\bullet)} \in$ todo$: t = \bar{t}$
$D^{\downarrow}_{(\bar{t},\bullet)} \coloneqq \sum \left\{ D^{\downarrow}_{(\bar{t},\mathfrak{s})} \,\middle|\, D^{\downarrow}_{(\bar{t},\mathfrak{s})} \in \text{todo} \right\}$ with box $\mathfrak{B} \coloneqq \bigsqcup \left\{ \mathfrak{B}_{(\bar{t},\mathfrak{s})} \,\middle|\, D^{\downarrow}_{(\bar{t},\mathfrak{s})} \in \text{todo} \right\}$
$D^{\uparrow}_{(\bar{t},\bullet)} \coloneqq \sum \left\{ D^{\uparrow}_{(\bar{t},\mathfrak{s})} \,\middle|\, D^{\uparrow}_{(\bar{t},\mathfrak{s})} \in \text{todo} \right\}$ with box $\mathfrak{B}$
return $D^{\downarrow}_{(\bar{t},\bullet)}$, $D^{\uparrow}_{(\bar{t},\bullet)}$

Algorithm 2: A discretization algorithm to under- and over-approximate a SoC distribution $\langle \bar{f}, f, z \rangle$ powering an MTP $\boldsymbol{M}$ for $\bar{t}$ time units.

to the pairwise closure of the subdistribution's boxes. After the merging of the distributions, we insert the resulting pair into the worklist in Line 20.

Termination. The propagation phase ends if every SoC distribution has reached and is annotated by the time horizon, as given in the until-condition of the loop (Line 21). In that case, we have propagated the approximations of the input SoC distribution through the input MTP until time $\bar{t}$, albeit in different MTP states. We collect and merge all the subdistributions left in the worklist, creating the final discrete SoC distributions as their componentwise sums (Line 24). The algorithm inevitably terminates since each MTP state is annotated with a noisy task of non-zero duration, thus moving the time frontier closer to the time horizon every time.

Efficiency. The worklist in Line 2 is initialized as a priority list. The sole reason for this is, to traverse the MTP in a breadth-first fashion so as to not propagate one path all the way to the time horizon by itself. Instead we aim to propagate the time annotation of every subdistribution along as a time frontier, in order to not waste any opportunity to merge two subdistributions, whenever they reach the same MTP state at the same time. It turns out that keeping the worklist sorted with respect to the time annotations achieves just that.

Another performance tweak, that is not reflected in the above algorithm for the sake of readability, is to not immediately scale the distributions in Lines 3 and 14, but instead annotate them with their scaling factor during initialization, and only update this factor with each scaling operation. This way a scaling operation boils down to only one scalar multiplication instead of a matrix-scalar multiplication. During the merging process (Lines 17–19) we annotate the resulting distribution with the sum of scaling factors of the affected distributions. Only at the very end, before returning the result pair of distributions we perform the scaling operation on the matrix.

Correctness. We state the following lemma in order to establish correctness of the above algorithm.

Lemma 39 — *Correctness of Algorithm 2.* Let $\boldsymbol{M}$ be an MTP, $\bar{t} \in \mathbb{R}_{>0}$ be a time horizon, N be a positive grid size, $\langle \bar{f}, f, z \rangle$ be an initial SoC distribution, $\langle \bar{f}_{\bar{t}}, f_{\bar{t}}, z_{\bar{t}} \rangle$ the resulting SoC distribution after powering $\boldsymbol{M}$ for $\bar{t}$ time units, and $\langle \overline{\mu}^{\downarrow}_{\bar{t}}, \mu^{\downarrow}_{\bar{t}}, \mathfrak{z}^{\downarrow}_{\bar{t}} \rangle$, $\langle \overline{\mu}^{\uparrow}_{\bar{t}}, \mu^{\uparrow}_{\bar{t}}, \mathfrak{z}^{\uparrow}_{\bar{t}} \rangle$ be the two discrete SoC distributions computed by Algorithm 2. Furthermore, let $[A; B]$ as well as $[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}]$ and $[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}]$ be random variables distributed as

$$[A; B] \sim \langle \bar{f}_{\bar{t}}, f_{\bar{t}}, z_{\bar{t}} \rangle, \quad [A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \sim \langle \overline{\mu}^{\downarrow}_{\bar{t}}, \mu^{\downarrow}_{\bar{t}}, \mathfrak{z}^{\downarrow}_{\bar{t}} \rangle \quad \text{and} \quad [A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \sim \langle \overline{\mu}^{\uparrow}_{\bar{t}}, \mu^{\uparrow}_{\bar{t}}, \mathfrak{z}^{\uparrow}_{\bar{t}} \rangle.$$

For any SoC $S \in \underline{\overline{\mathbb{S}}}_{\perp}$, we have

$$\mathbf{Pr}\big[[A^{\uparrow}_{\boxplus}; B^{\uparrow}_{\boxplus}] \leq S\big] \quad \geq \quad \mathbf{Pr}[[A; B] \leq S] \quad \geq \quad \mathbf{Pr}\big[[A^{\downarrow}_{\boxplus}; B^{\downarrow}_{\boxplus}] \leq S\big].$$

Proof:

> By inductively using Lemma 29. ■

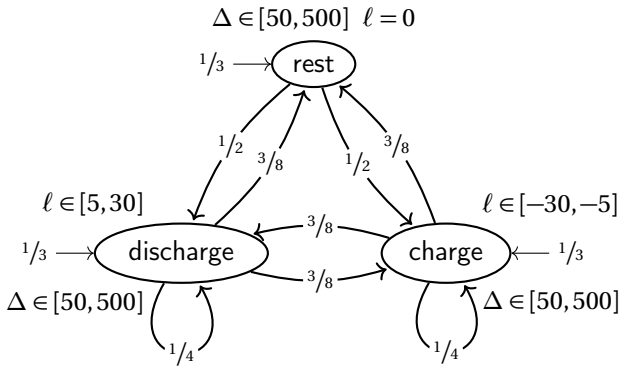


Figure 4.9: A meta-MTP to sample task sequences.

Implementation Details

The algorithm was implemented in `Python` with extensive use of the `NumPy` library which provides efficient vectorized implementations of matrix-vector operations. Grids of size N are implemented as `NumPy`-arrays of size $N \times N$. The successor operators $\mathbb{K}$ and all of its variations are implemented as multithreaded vectorized functions with the help of the `NumExpr` package, a fast numerical expression evaluator for `NumPy`. In addition, the successor with respect to a discretized noisy task is computed in concurrent fashion, by providing one thread for each load in the support of the noisy task.

While using `NumPy`'s vectorization capabilities lead to an efficient implementation, it also leads to restrictions. It is only efficient if every single operation has a vectorized implementation. Especially for some iterative schemes this is not always easy to achieve, one example being the algorithm to approximate saturation time points (Algorithm 1). Hence, for each saturating task (Δ, ℓ) we resort by default to the non-iterative approximation of the saturation time point without invoking the algorithm. Conceptually, this corresponds to running the algorithm with a precision of $\varepsilon \coloneqq \Delta$.

Comparison With Static Discretization

In order to achieve a satisfying sample size of traces we synthesize an arbitrary number of reasonable task sequences from the MTP-like structure displayed in Figure 4.9.

This structure serves as a meta-MTP, which is to be interpreted as follows. Initially we draw the initial state at random according to the initial distribution π. In a state annotated with $([\Delta_{\min}, \Delta_{\max}], [\ell_{\min}, \ell_{\max}])$, we sample a uniformly distributed time duration $\Delta \sim \mathscr{U}[\Delta_{\min}, \Delta_{\max}]$ as well as a load $\ell \sim \mathscr{U}[\ell_{\min}, \ell_{\max}]$ from the task annotation. The sampled duration is to be interpreted as is, whereas the sampled load ℓ serves as the location parameter of a normal distribution $\mathscr{N}(\ell, 1)$. The noisy task $(\Delta, \mathscr{N}(\ell, 1))$ is then added to the task sequence, and a successor state is drawn at random according to the branching behavior of the meta-MTP. This is repeated until the task sequence has the desired length.

The meta-MTP allows for a minimal degree of structure in the task sequences, while still delivering a high enough degree of randomness in task lengths and intensities of the tasks loads it generates. For example it doesn't allow multiple consecutive different resting periods, however it does account for consecutive charging or discharging tasks to represent load changes, although with a slight bias towards leaving the current MTP state.

The battery is instantiated with a capacity of 175 000 J, $\mathsf{c} = 0.5$ and various diffusion parameters p, with a depletion threshold $\mathsf{depl} = \overline{a} \cdot 0.5$. Its initial SoC is uniformly distributed on the set $\overline{a}\,[0.6, 0.7] \times [0.6, 0.7]\,\overline{b}$.

In order to compare SD against AD, we proceed as follows. We set a specific target precision level ε we want both SD and AD to reach, i.e. the resulting depletion risk intervals of both algorithms should be of width equal or less than ε. To this end, we search for the respective grid sizes that both algorithms need in order to achieve the given precision, via an interval-halving scheme. We start with grid sizes $N^{\uparrow} = 0$ and $N^{\downarrow} = 6000$, where 6000 is deliberately chosen high enough. We run the algorithm with grid size $N = (N^{\downarrow} - N^{\uparrow})/2$. If the resulting depletion risk interval $\left[\mathfrak{z}^{\uparrow}, \mathfrak{z}^{\downarrow}\right]$ is of a larger width than ε, i.e. $\mathfrak{z}^{\downarrow} - \mathfrak{z}^{\uparrow} \leq \varepsilon$, we update $N^{\uparrow} = (N^{\downarrow} - N^{\uparrow})/2$, otherwise we set $N^{\downarrow} = (N^{\downarrow} - N^{\uparrow})/2$. We continue updating until $N^{\uparrow}$ and $N^{\downarrow}$ are less than $\varepsilon_N = 5$ apart, since grid sizes that are less than 5 apart do not significantly distinguish themselves in terms of depletion risk interval width. We visualize the runtimes and grid sizes of the last iteration of each run for each algorithm as a datapoint in a set of plots given in Figure 4.10.

We evaluate three different scenarios to investigate the shortcomings and advantages of each method.

1. **Normal task sequences, high diffusion:** In this scenario, we set the diffusion parameter $\mathsf{p} = 0.0001$, and interpret the sampled task sequences as is, i.e. without noise.

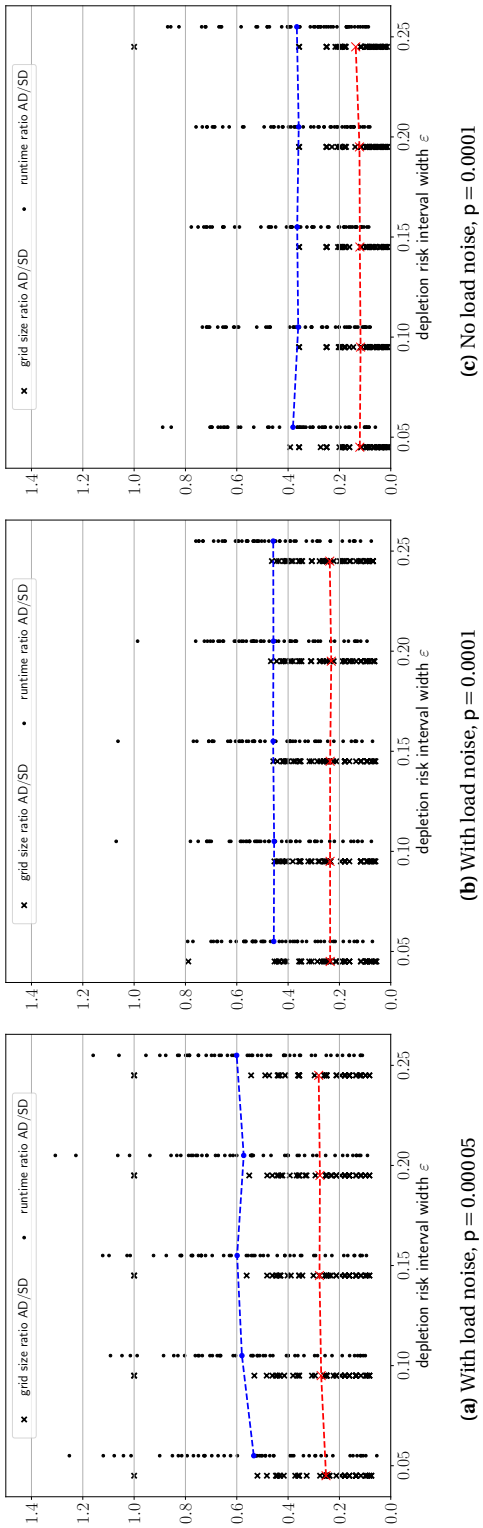
2. **Noisy task sequences, high diffusion:** We keep the diffusion parameter at $\mathsf{p} = 0.0001$, and interpret the sampled task sequences as location parameter of a Gaussian distribution, that is subsequently truncated and discretized into 5 loads.

3. **Noisy task sequences, low diffusion:** The sampled task sequences are again interpreted as location parameter of a Gaussian distribution, that is subsequently truncated and discretized into 5 loads. We reduce the diffusion rate to $\mathsf{p} = 0.000\,05$.

For each scenario we synthesize 50 task sequences of length 100.

Discussion

A comparison of plots 4.10a and 4.10b shows that by increasing the diffusion rate p, the performance gain of AD relative to SD slightly increases, both in terms of runtime and grid size, and thus in memory consumption. The reason is that the support of the SoC distribution is less spread out, and thus occupies a smaller portion of the bounding box, meaning we have less cells carrying a non-zero probability mass. In contrast, for a slower diffusion rate, most of the cells carry a non-zero probability mass in the AD scheme, while for SD this increase in non-empty cells is less pronounced. Thus, we sometimes see SD marginally outperform AD on



(a) With load noise, p = 0.00005

(b) With load noise, p = 0.0001

(c) No load noise, p = 0.0001

Figure 4.10: Plots visualizing the SD vs AD comparison in terms of runtime as well as grid sizes. In red and blue, we highlight the means of both ratios.

certain runs in terms of runtime (runtime ratio larger than 1), even though AD consistently requires smaller grids.

AD is especially efficient at identifying 0 or 1 depletion risk runs. The reason is that one can essentially check for these cases with a 1×1 grid. This induces a binary scenario in which the entire probability mass either does or does not deplete. SD on the other hand, still potentially requires large grids on order to come to the same conclusions.

Plots 4.10b and 4.10c showcase a similar difference, but in this case the spread is caused by noise in the task loads, in contrast to no load noise at all.

In conclusion, the results paint a relatively clear picture of superiority of AD over SD, in terms of space as well as runtime efficiency; AD is very rarely slower than SD (and if so, only negligibly), and it certainly needs less memory, because it requires smaller grids. In addition it is extremely quick in cases of sure depletion and non-depletion.

4.5 Depletion Risk Estimation By Percentile Propagation

In the previous two sections we developed algorithms that enabled the use of arbitrary initial SoC distribution as well as load distributions, via appropriate discretization schemes of both. The price of being so general and safe at the same time, is that of precision. Due to the permanent rounding of SoCs onto the smallest/largest SoC still in the same cell after each task, accumulates more and more errors, the longer the task sequence we strain the battery with.

In certain special, yet realistic scenarios we can however do better. First, we give up the endeavour of approximating the full SoC distribution along a task sequence. Instead, we are satisfied with only computing bounds on the depletion risk. Often this constitutes the most crucial information of an energy budget analysis of a battery powered system.

The idea is to exploit that the operator $\overline{\mathbb{K}}$ preserves certain orders on SoCs when strained by a single task, and therefore, inductively, also sequences of tasks. This means, that if a SoC S depletes when strained by a task sequence $(T_i)_{i=0}^n$, then every "worse" SoC will also deplete. Dually, if S does not deplete along $(T_i)_{i=0}^n$, then we also know that every "better" SoC will survive $(T_i)_{i=0}^n$. If we additionally are able to say that S is a "better" SoC than q percent of all the SoCs within the support of the initial distribution, we can deduce that the depletion risk is *at least* q, if S indeed depletes, or dually, that the depletion risk is *at most* q if S survives $(T_i)_{i=0}^n$. Since q is between 0 and 1, the idea is to iteratively find tighter bounds for q using an interval-halving scheme reminiscent of binary search until we have narrowed down the interval around q to a width of ε.

In the following we will formalize the above idea that we refer to as *Percentile Propagation* (PP).

First, we need to establish what "better" and "worse" on SoCs actually mean. It is not immediately clear how to best compare two KiBaM SoCs $[a_0; b_0]$ and $[a_1; b_1]$ in a meaningful way, especially when $a_0 < a_1$ and $b_0 > b_1$ at the same time.

> ***Remark.*** It is conceivable to define an order with respect to a discharging task (Δ, ℓ). Conceptually, two candidates come to mind:
>
> - under a fixed load $\ell > 0$, the SoC that depletes earlier is "worse".

- under a fixed duration $\Delta > 0$, the SoC exhibiting a lower least depleting load $\ell^{\rightarrow}_{\underline{a},\Delta}$ is "worse".

Both candidates, while making a lot of conceptual sense, are not preserved by $\mathbb{K}$ and can be dismissed using the same counter example.
Suppose, $\mathsf{cap} \coloneqq 100$, $\mathsf{depl} \coloneqq 0$ and $\mathsf{c} = 0.5$ such that $\overline{a} = \overline{b} = 50$. Consider two safe SoCs $[a_\varepsilon; \overline{b}]$ and $[\overline{a}; b_\varepsilon]$, where $a_\varepsilon \in \left]\underline{a}, \underline{a} + \varepsilon\right]$ and $b_\varepsilon \in \left]\underline{b}, \underline{b} + \varepsilon\right]$ are safe but very close to unsafe available and bound charge levels. It is clear that $[a_\varepsilon; \overline{b}]$ is "worse" than $[\overline{a}; b_\varepsilon]$ with respect to the above concepts, given any relatively short depleting task of $[a_\varepsilon; \overline{b}]$, since its available charge component is close to unsafe to begin with. However, with a short, sharp charging task $T \coloneqq (\Delta_\varepsilon, \vec{\ell}[a_\varepsilon; \overline{b}])$, with Δ_ε being a very short duration, we saturate both SoCs,

$$\overline{\underline{\mathbb{K}}}_T[a_\varepsilon; \overline{b}] = [\overline{a}; b] \quad \text{and} \quad \overline{\underline{\mathbb{K}}}_T[\overline{a}; b_\varepsilon] = [\overline{a}; b']$$

Since $[\overline{a}; b_\varepsilon]$ was saturated to begin with, and the charging overpowers the diffusion for small diffusion parameters p, b' will not be much larger than b_ε, as it is not affected by the load $\vec{\ell}$, and the diffusion is slow. On the other hand, $[a_\varepsilon; \overline{b}]$ will be saturated by definition within Δ_ε time, and the bound charge won't decrease much due to diffusion during that time, again, since diffusion is slow. Hence, we have $b' < b$ and thus $[\overline{a}; b'] \leq [\overline{a}; b]$. Thus, using sharp charging tasks, one SoC can "overtake" another SoC, with respect to the above conceptual orders.

It turns out that the most permissive order that $\overline{\underline{\mathbb{K}}}$ preserves under discharging as well as charging, a property which is crucial for the algorithm we derive later on, is the componentwise $\leq$, However, $\leq$ is but a partial order on SoCs, thus some SoCs are incomparable. For example $[1;2]$ and $[2;1]$ are incomparable.

In conclusion, we restrict the initial SoC distribution to a distribution that does not support two pairwise incomparable SoCs with respect to $\leq$. In the remainder, whenever we are arguing about SoC distributions, we implicitly assume this restriction. Luckily, this is not an unrealistic assumption, since batteries that have had enough time to equilibrate, like batteries of nanosatellites sitting inactively in their launching pod waiting for deployment, exhibit exactly such SoC distributions. Such SoC distributions basically degenerate to one-dimensional distributions, since each SoC $[a; b]$ is uniquely defined by the sum of its components $a + b$. In the following we will formalize these concepts, startign with the classic definition of percentiles.

Definition 43 — *Percentile.* Let F be a cumulative density function (CDF) and f be the corresponding PDF. We define by

$$F^{-1}(q) \coloneqq \inf_{x \in \mathbb{R}} \left\{ F(x) = \int_{-\infty}^{x} f(x)\,\mathrm{d}x \geq q \right\}$$

the generalized inverse of F. We call the value $F^{-1}(q)$ the q–percentile of F.

The reason why F^{-1} is the generalized inverse of F and requires the infimum operation is because F is not necessarily invertible in the functional sense. F at-

tains a regular inverse function on the domain $[0,1]$ only for strictly monotonic, continuous CDFs. In these cases, we may drop the infimum operation. On the other hand, some bimodal or discrete distributions require the infimum operation for percentiles to be unique, since there might otherwise be a multitude of candidates.

For a certain subclass of SoC distributions it suffices to not consider the available and bound charge levels individually, but instead their sum, i.e. the total amount of charge stored. This subclass is characterized by the property that $\leq$ is a total order on the support on f and $\bar{f}$ combined. In this case, the sum of available and bound charge corresponds to exactly one unique SoC supported by the SoC distribution. We use this fact to define the notion of SoC percentiles. We start by proving the following.

Lemma 40. Let S be a set of SoCs. If $\leq$ is a total order on S, then for two SoCs $[a;b],[a';b'] \in S$, we have that

$$a+b=a'+b' \Longrightarrow [a;b]=[a';b'].$$

Proof:

For the sake of deriving a contradiction let us assume there are two SoCs $[a;b],[a';b'] \in S$ with $a+b=a'+b'$, but $[a;b] \neq [a';b']$. Since $[a;b] \neq [a';b']$ and S is a totally ordered set, either $[a;b] > [a';b']$ or $[a;b] < [a';b']$. In either case, we also immediately have $a+b < a'+b'$ or $a+b > a'+b'$, in contradiction to $a+b=a'+b'$. ↯ ■

Definition 44 — *SoC percentiles.* Let $\langle \bar{f}, f, z \rangle$ be a SoC distribution such that $(\{\overline{a}\} \times \mathrm{supp}(\bar{f})) \cup \mathrm{supp}(f)$ is a totally ordered set with respect to $\leq$. We define h as follows:

$$h(c) \coloneqq \begin{cases} \bar{f}(b), & \text{if } c = \overline{a}+b \wedge [\overline{a};b] \in \{\overline{a}\} \times \mathrm{supp}(\bar{f}) \\ f(a,b), & \text{if } c = a+b \wedge [a;b] \in \mathrm{supp}(f) \\ 0, & \text{otherwise} \end{cases}$$

Then, the $(z+q)$–percentile of $\langle \bar{f}, f, z \rangle$ is the unique SoC $[a;b]$ such that $c=a+b$ is the (conventional) q–percentile of h, for $0 \leq q \leq 1$.

Lemma 41. For each SoC distribution $\langle \bar{f}, f, z \rangle$ with $(\{\overline{a}\} \times \mathrm{supp}(\bar{f})) \cup \mathrm{supp}(f)$ being a totally ordered set with respect to $\leq$, $z+h$ is a well-defined probability density function.

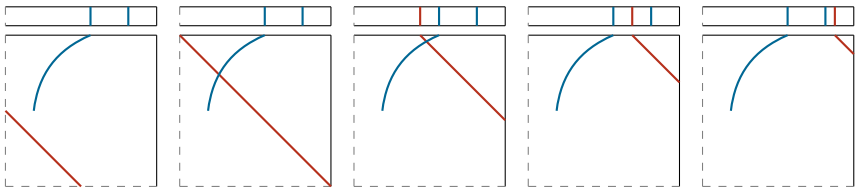
Proof:

By Lemma 40, we have that the sum of both SoC components is unique for totally ordered sets. By assumption $\{\overline{a}\} \times \mathrm{supp}(\bar{f}) \cup \mathrm{supp}(f)$ is a totally ordered set, which means that h is functional; for each value c there is at most one single

SoC supported by $\langle \bar{f}, f, z \rangle$. Furthermore, the values of the image of h are all taken from a SoC distribution, meaning they are positive. In addition, since each sum corresponds to exactly one supported safe SoC, integration over all possible values of this sum results in $1-z$, i.e.

$$\int h(c)\,\mathrm{d}c = \int_{\underline{a}+\underline{b}}^{\overline{a}+\overline{b}} h(c)\,\mathrm{d}c = 1-z \qquad \blacksquare$$

Remark — Intuition behind Definition 44. Let $\langle \bar{f}, f, z \rangle$ be a SoC distribution. The intuition behind h is, that it essentially constitutes a diagonal sweep of the safe SoC space, cumulatively "picking up" SoCs in the appropriate order. The function is well-defined if, for every position of the sweep diagonal (red line), it intersects the support of $\langle \bar{f}, f, z \rangle$ (blue lines) exactly at one single SoC, like in the following visual example:



SoC distributions are inherently non-continuous because they are by definition separated into three distinct parts. We can thus conclude, that for SoC distributions, percentiles are in general not unique without the infimum operation. With a few restrictions, however, we are able to fulfill all the necessary assumptions to drop the infimum operator.

Lemma 42. For SoC distributions of the form $\langle \bar{f}, f, 0 \rangle$, percentiles are unique without the infimum operation if

1. All of the support of $\langle \bar{f}, f, 0 \rangle$ is exclusively in f or $\bar{f}$, i.e.
$$\int_{\underline{b}}^{\overline{b}} \bar{f}(b)\,\mathrm{d}b = 1 \quad \text{or} \quad \int_{\underline{a}}^{\overline{a}} \int_{\underline{b}}^{\overline{b}} f(a,b)\,\mathrm{d}a\,\mathrm{d}b = 1$$
2. $\mathrm{supp}(f) \cup (\overline{a} \times \mathrm{supp}(\bar{f}))$ is a totally ordered set, and
3. the corresponding CDF of h
$$H(x) = \int_{\underline{a}+\underline{b}}^{x} h(c)\,\mathrm{d}c$$
is strictly monotonically increasing.

Proof:

Let us assume all of the probability mass is in $\bar{f}$. Then $h(\overline{a}+b)=\bar{f}(b)$ and the $(0+q)$–percentile of the SoC distribution is one q–percentile of h, say $\overline{a}+b_q$. Since, the CDF of $\bar{f}$ is strictly monotonically increasing, also the CDF of h is and $\overline{a}+b_q$ is the unique q–percentile of h. Hence, $[\overline{a};b_q]$ is the unique q–percentile of $\langle\bar{f},f,0\rangle$.
The case of the probability mass being in f is a bit more involved, but essentially analogous. ■

Basically, we require that initially either $\bar{f}$ or f holds all of the probability mass, and that we start with a void depletion risk. Again, these restrictions are not very prohibitive.

Example 11. Let $\langle\bar{f},f,z\rangle$ be a SoC distribution with $z=0$. The following scenarios lead to unique q–percentile of $\langle\bar{f},f,z\rangle$, since $\leq$ is a total order on $\mathrm{supp}(\bar{f})$ or $\mathrm{supp}(f)$.

- $\bar{f}$ is the PDF of a truncated Gaussian distribution on the interval $[0.8,0.9]\cdot\overline{b}$, and f holds no probability mass.
- f is the PDF of a uniform distribution on $\{x\cdot[\mathsf{c};1-\mathsf{c}]\mid x\in[0.75,0.95]\cdot\mathsf{cap}\}$, and $\bar{f}$ holds no probability mass. (A non-saturated battery in equilibrium).
- f is the PDF of a truncated Gaussian distribution on $\{0.5\overline{a}\}\times([0.5,0.85]\cdot\overline{b})$, and $\bar{f}$ holds no probability mass.

Bounding The Depletion Risk For Simple Task Sequences

We are now in the position to give a pseudocode algorithm to bound the depletion risk within an interval of arbitrary width.

The algorithm starts off with checks for sure depletion (Line 1) and sure survival (Line 4).

The largest supported initial SoC is fetched, and we check with the $\underline{\overline{\mathbb{K}}}^{\downarrow}$ over-approximation operator whether the task sequence depletes. If this is the case, we are sure that every other supported SoC also depletes under $(T_i)_{i=0}^N$, and the singleton interval $[1,1]$ is returned to indicate sure depletion.

Dually, we proceed with the smallest supported SoC and the under-approximation operator $\underline{\overline{\mathbb{K}}}^{\uparrow}$, and check for depletion. In the negative case, we can be certain that every other supported SoC will not deplete either, thus we return the singleton interval $[0,0]$ to indicate sure survival.

If both of these checks fail we start the iterative part of the algorithm with the interval $[0,1]$, since the depletion risk must be between 0 and 1. Afterwards we keep halving the interval in the following sense. We look at q, the mean probability of $z_{\downarrow}$ and $z_{\uparrow}$ (Line 11), and check whether the q–percentile depletes when strained with $(T_i)_{i=0}^N$ using the under- and over-approximation successor operators $\underline{\overline{\mathbb{K}}}^{\uparrow}$ and $\underline{\overline{\mathbb{K}}}^{\downarrow}$ (Line 13). If the over-approximation exhibits the depletion SoC $\bot$, we can be sure that the depletion risk is at least q, and thus assign $z_{\uparrow}\coloneqq q$. If the under-approximation does not exhibit the depletion SoC $\bot$, we can be sure that the depletion risk is at most q, and thus assign $z_{\downarrow}\coloneqq q$ (Line 26). If the approximations do not agree, we narrow the approximation corridor until they eventually

Input : A SoC distribution $\langle \bar{f}, f, z \rangle$, a task sequence $(T_i)_{i=0}^N$ and $\varepsilon > 0$
Output : An interval $\left[z_\uparrow, z_\downarrow\right]$ of width at most ε with $z_N \in \left[z_\uparrow, z_\downarrow\right]$

$[a_{\max}; b_{\max}] \coloneqq \max\{\operatorname{supp}(f) \cup (\overline{a} \times \operatorname{supp}(\bar{f}))\}$
if $\underline{\overline{\mathbb{K}}}^{\downarrow}_{(T_i)_{i=0}^N}[a_{\max}; b_{\max}] = \bot$ **then**
  return $[1, 1]$

$[a_{\min}; b_{\min}] \coloneqq \min\{\operatorname{supp}(f) \cup (\overline{a} \times \operatorname{supp}(\bar{f}))\}$
if $\underline{\overline{\mathbb{K}}}^{\uparrow}_{(T_i)_{i=0}^N}[a_{\min}; b_{\min}] \neq \bot$ **then**
  return $[0, 0]$

$z_\downarrow \coloneqq 1$
$z_\uparrow \coloneqq 0$
while $z_\downarrow - z_\uparrow \geq \varepsilon$ **do**
  $q \coloneqq (z_\downarrow + z_\uparrow)/2$
  $[a_q; b_q] \coloneqq q$–percentile SoC of $\langle \bar{f}, f, z \rangle$
  $S_\uparrow,\ S_\downarrow \coloneqq \underline{\overline{\mathbb{K}}}^{\uparrow}_{(T_i)_{i=0}^N}[a_q; b_q],\ \underline{\overline{\mathbb{K}}}^{\downarrow}_{(T_i)_{i=0}^N}[a_q; b_q]$ with precision $\varepsilon_{\overline{\Delta}}$
  if $S_\uparrow = \bot$ **then**
    $z_\uparrow \coloneqq q$
  else if $S_\downarrow \neq \bot$ **then**
    do
      $\varepsilon_{\overline{\Delta}} \coloneqq 0.1 \cdot \varepsilon_{\overline{\Delta}}$
      $S_\uparrow,\ S_\downarrow \coloneqq \underline{\overline{\mathbb{K}}}^{\uparrow}_{(T_i)_{i=0}^N}[a_q; b_q],\ \underline{\overline{\mathbb{K}}}^{\downarrow}_{(T_i)_{i=0}^N}[a_q; b_q]$ with precision $\varepsilon_{\overline{\Delta}}$
    while $S_\uparrow = \bot \neq S_\downarrow \vee S_\uparrow \neq \bot = S_\downarrow$
    reset $\varepsilon_{\overline{\Delta}}$
    if $S_\downarrow = \bot$ **then**
      $z_\uparrow \coloneqq q$
    else
      $z_\downarrow \coloneqq q$
  else
    $z_\downarrow \coloneqq q$
return $\left[z_\uparrow, z_\downarrow\right]$

Algorithm 3: An interval halving algorithm to bound the depletion risk of a task sequence up to ε.

agree, meaning, we gradually increase the saturation time point precision $\varepsilon_{\overline{\Delta}}$ (do-while loop starting in Line 19). We keep increasing the precision by a factor of 10^{-1} (Line 17) and recompute the approximations until a consensus is reached, upon which we reset the precision $\varepsilon_{\overline{\Delta}}$ to it's initial value. If said consensus is depletion (Line 21), we update $z_{\uparrow} \coloneqq q$, otherwise $z_{\downarrow} \coloneqq q$, for the same reason as above. Finally, after having narrowed down the interval surrounding the true depletion risk enough, the loop will terminate.

Termination. The algorithm in its current form is a semi-decision procedure. For a diverging scenario two conditions must be fulfilled, namely **(i)** the true depletion risk is q and corresponds to one of the q–percentiles probed by the algorithm in Line 10 (for example 0.5, 0.25, 0.125,...), and **(ii)** the SoC trace corresponding to the q–percentile SoC becomes saturated at least once. If those two conditions are indeed fulfilled, the inner do-while loop (Line 19) diverges, because the under-approximation (using $\underline{\overline{\mathbb{K}}}^{\uparrow}$) always indicates depletion, while the over-approximation (using $\underline{\overline{\mathbb{K}}}^{\downarrow}$) always indicates the contrary, no matter the precision.

If either of the above two conditions is not fulfilled, then **(i)** the inner do-while loop always terminates, since eventually a consensus on depletion of the trace will be reached by consecutively increasing the precision, or **(ii)** the loop will not be executed at all since the operators $\underline{\overline{\mathbb{K}}}^{\uparrow}$ and $\underline{\overline{\mathbb{K}}}^{\downarrow}$ degenerate to the $\underline{\overline{\mathbb{K}}}$ operator, and thus behave equally. The outer while-loop terminates since the difference $z_{\downarrow} - z_{\uparrow}$ is halved in every iteration inevitably decreasing to a value smaller than ε.

Runtime Efficiency. Giving a rigorous runtime analysis is not straight-forward for the following reasons. Even in the case of termination, it is hard to deduce how often the inner do-while loop (Line 19) is executed. Also, the percentile computation in Line 11 follows a numeric approximation scheme (at least for most continuous distributions) for which runtime complexity is hard to deduce. Here is a key point however: For a precision $\varepsilon \in \mathcal{O}(2^{-n})$, the outer while loop in Line 9 is executed a logarithmic number of times (in n), since the interval $\left[z_{\uparrow}, z_{\downarrow}\right]$ is halved with each iteration.

Memory Consumption. Tracking a SoC along a task sequence is constant in memory, since it only requires saving the current SoC. In each iteration a percentile computation is the most expensive operation in terms of memory, thus the algorithm has the same space complexity as the percentile computation.

Soundness. In the event of termination, soundness follows from the fact that percentiles are unique (Lemma 42). Since $\underline{\overline{\mathbb{K}}}^{\downarrow}$ and $\underline{\overline{\mathbb{K}}}^{\uparrow}$ approximate $\underline{\overline{\mathbb{K}}}$ correctly (Lemma 16), soundness follows.

Ensuring Termination In Practice. In practice we can ensure termination by imposing a maximum precision to the inner while loop, such that we abort iteration if the results of both approximations are close enough. We argue that, if both approximations are not able to reach a consensus on the q–percentile in question, we have already determined the depletion risk precisely enough and we return the interval $\left[q - \frac{1}{2}\varepsilon, q + \frac{1}{2}\varepsilon\right]$ around q. Using this fix potentially violates the soundness statement, because it is difficult to relate the two precision levels ε and $\varepsilon_{\overline{\Delta}}$. In the usual termination case we decide when to stop iteration based

on precision level ε, while using the fix, we decide the return value based on a maximum precision level $\varepsilon_{\overline{\Delta}}$ for saturation time points. This is however merely a theoretical consideration. For practical applications it is very unlikely that we run into the divergence case to begin with, and even if it happens, setting a low enough saturation time point precision delivers satisfyingly precise results.

Handling Discrete Noisy Task Sequences

We lift Algorithm 3 to sequences of discrete noisy tasks in Algorithm 4. We generate every possible task sequence of non-zero probability (Line 2), run Algorithm 3 on each sequence to find its depletion risk interval (Line 4), and weight the interval bounds with the probability of actually achieving the sequence at hand (Lines 5–6). Finally, the weighted sum of the sequence's lower and upper bound values defines the overall lower and upper bound on the depletion risk.

Input : A SoC distribution $\langle \bar{f}, f, z \rangle$, a discretized noisy task sequence $(\Delta_i, \gamma_i)_{i=0}^N$ and $\varepsilon > 0$
Output : An interval $\left[z_\uparrow, z_\downarrow\right]$ with $z_N \in \left[z_\uparrow, z_\downarrow\right]$

1 $\left[z_\uparrow, z_\downarrow\right] \coloneqq [0,0]$
2 **foreach** $(\ell_j)_{j=0}^N \in \mathrm{supp}(\gamma_0) \times \cdots \times \mathrm{supp}(\gamma_N)$ **do**
3 $\quad p \coloneqq \prod_{k=0}^N \gamma_k(\ell_k)$
4 $\quad \left[z'_\uparrow, z'_\downarrow\right] \coloneqq$ Algorithm 3 on input $\langle \bar{f}, f, z \rangle, (\Delta_j, \ell_j)_{j=0}^N, \varepsilon$
5 $\quad z_\uparrow \coloneqq z_\uparrow + p \cdot z'_\uparrow$
6 $\quad z_\downarrow \coloneqq z_\downarrow + p \cdot z'_\downarrow$
7 **return** $\left[z_\uparrow, z_\downarrow\right]$

Algorithm 4: The PP algorithm for noisy task sequences in pseudo-code.

The obvious bottleneck here is the size of the cartesian product (Line 2). For n tasks, each supporting k loads, the cartesian product has k^n members, for which we run Algorithm 3. For $k = 1$, the cartesian product degenerates to one single task sequence, and thus Algorithm 4 essentially degenerates to Algorithm 3. With a generative implementation of the cartesian product, we don't need to store every trace in memory, rather than the state of the generator, which takes $O(n)$ space. Thus, this paradigm excels in space efficiency.

Implementation And Comparison With Adaptive Discretization

The PP algorithm is implemented in `Python`, in addition with the `SciPy` library. The `SciPy` library provides predefined ways to create probability distribution objects and compute its percentiles.

To evaluate the performance of overall complexity of percentile propagation, we run both the PP (Algorithm 4) as well as the AD (Algorithm 2) on the same noisy task sequences, with different precision levels ε and grid sizes N, until they both result in a depletion risk interval of width ε around the true depletion risk. To

find suitable values, we employ an interval halving scheme. For PP we start with precision levels $\varepsilon_\perp^\uparrow = 0$ and $\varepsilon_\perp^\downarrow = 1$, and run PP on the midpoint value $(\varepsilon_\perp^\downarrow - \varepsilon_\perp^\uparrow)/2$. If PP results in a depletion risk interval $\left[z_\uparrow, z_\downarrow\right]$ of width smaller or equal than the given target precision $\varepsilon_\perp$, we repeat the previous step with $\varepsilon_\perp^\uparrow \coloneqq (\varepsilon_\perp^\downarrow - \varepsilon_\perp^\uparrow)/2$, and otherwise with $\varepsilon_\perp^\downarrow \coloneqq (\varepsilon_\perp^\downarrow - \varepsilon_\perp^\uparrow)/2$. We continue with these updates until we estimated the required precision parameter up to a given ε, meaning until $\varepsilon_\perp^\downarrow - \varepsilon_\perp^\uparrow \leq \varepsilon$.

For AD, we repeat the same procedure as in the previous comparison with SD (see Section 24).

Comparison With AD On Tasks Without Load Noise. We sample 50 task sequences of length 100 from the meta-MTP shown in Figure 4.9. For PP, this implies that Algorithm 4 reduces to only one run of Algorithm 3.

We assume a battery ($\mathsf{cap} = 175\,000\,\mathrm{J}$, $\mathsf{c} = 0.5$, $\mathsf{p} = 0.000\,05$, $\mathsf{depl} = 0.5$) that is in equilibrium and that is between 60 % and 70 % full, i.e. on the set

$$S \coloneqq \{x \cdot [\mathsf{c}; 1-\mathsf{c}] \mid x \in [0.6, 0.7] \cdot \mathsf{cap}\}.$$

First, note that $\leq$ is indeed a total order on S, thus we can apply PP. We assume that the initial SoC distribution $\langle \tilde{f}, f, z \rangle$ is such that f is the PDF of a uniform distribution over S.

We investigate precision levels $\varepsilon = 0.25, 0.2, 0.15, 0.1, 0.05$ and 0.01. In this scenario we can simply set the required precision level of PP, since Algorithm 3 then returns a depletion risk interval of width at most ε. As initial saturation time point precision for PP we use $\varepsilon_{\overline{\Delta}} = 1$.

For AD, we search for suitable grid sizes, like we did before in the comparison of SD and AD in Section 24. Saturation time points are approximated using the default non-iterative scheme.

The results are visualized in Figure 4.11c, where we use a log-scale for improved visual inspection.

Comparison With AD On Noisy Tasks. To illustrate the respective strengths and drawbacks of each method on noisy tasks, we evaluate PP against AD on noisy task sequences. We choose two different settings in order to showcase the shortcomings and advantages of each method. We again investigate precision levels $\varepsilon = 0.25, 0.2, 0.15, 0.1, 0.05$ and 0.01.

1. **Short sequences, 4 load noise samples:** We sample 50 task sequences of length 8 from the meta-MTP shown in Figure 4.9. In order to generate a representative set of sequences, we increased the maximal loads generated from the meta-MTP (Figure 4.9) from 30 to 40. Otherwise one a few runs actually exhibit non-zero depletion risks, because of the shortness of the sequences. The load of each respective task is interpreted as the mean of a truncated Gaussian distribution, which is then discretized into 4 loads. For PP, these values imply a total of $4^8 = 65\,536$ applications of Algorithm 3. The results are visualized in Figure 4.11a.

2. **Longer sequences, 2 load noise samples:** We sample 50 task sequences of length 16 from the meta-MTP shown in Figure 4.9. the maximal loads generated from the meta-MTP (Figure 4.9) are again increased from 30 to 40. The loads of each respective task is interpreted as the mean of a Gaussian

distribution, which is then discretized into 2 loads. For PP, these values also imply a total of $2^{16} = 65\,536$ applications of Algorithm 3, thus the runtime of PP is roughly equal to the previous scenario. The results are visualized in Figure 4.11b.

Discussion

Observing the plots 4.11a and 4.11b, it becomes evident that on noisy task instances that use a larger amount of load noise samples, PP does not perform well, because of the prohibitively fast growing cartesian product. It still provides an alternative with low space requirements compared to AD, and comes with an a priori configurable precision level. We can see, that the runtime ratio is less in favor of AD as the task sequences become longer, since then AD needs larger grids.

PP consistently beats AD on task sequences without load noise, as can be observed in Figure 4.11c. This is not surprising since the bottle neck of PP, the cartesian product, consists of just one single trace. Additionally, the sequence length has a negligible impact on the precision of PP, while it has a more severe impact on AD.

In general, it appears that for very high precision, as witnessed by the $\varepsilon = 0.01$ datapoints, PP catches up to AD in terms of runtime, since the runtime ratio increases in all three plots.

A Remark On Ease Of Implementation

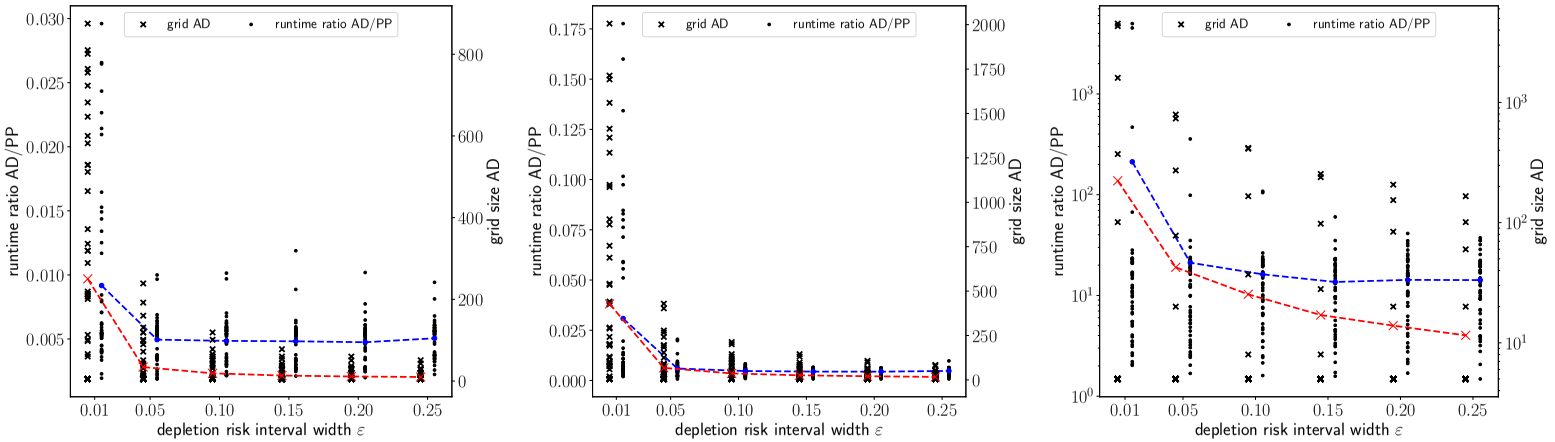
PP does not require many bells and whistles. Implementing the pseudo-code is straight-forward in `Python`. We use `SciPy` for predefined ways to create probability distribution objects and compute its percentiles, and `itertools` for a memory efficient implementation of the cartesian product as a generator, meaning it generates the members of the cartesian product on the fly without the need to store the entire product in memory. In terms of multithreading, PP is easily parallelizable. Several threads can work concurrently on the member task sequences of the cartesian product.

Overall, SD and AD are more complex to implement than PP. The former require a vectorization-friendly programming style, that often turns out to be inconvenient, requiring complex indexing and in-depth knowledge of `NumPy` to make sure that no unnecessary copies of large grids are internally produced, and that most operations are implemented inplace.

It should not come as a surprise that implementing AD (and SD) took much longer than implementing PP.

Concluding Remarks

After examining the evaluation of AD and PP a few points are worth being highlighted. First, we point out that AD is the universally applicable algorithm that can be run on any initial SoC distribution, while PP requires mild conditions on the initial charge to be fulfilled. The consequence of this is, that AD can be used to analyze systems which are already in operation, while PP is only suitable in situations of no activity (i.e. a fully equilibrated system) or fully charged systems. In addition, AD can efficiently handle not only noisy task sequences but also Markov Task Processes (see Section 3.6) as load model.



(a) 8 tasks, 4 load noise samples.

(b) 16 tasks, 2 load noise samples.

(c) 100 tasks, no load noise. Log-scale for visual inspection.

Figure 4.11: The plots visualizing the results of the AD vs PP comparison, in terms of runtime. The grid size of AD is plotted in absolute numbers on the right y-axes, to illustrate memory consumption. In red and blue, we highlight the medians of each set of datapoints.

Additionally, PP only estimates the depletion risk, which most of the time is the quantity of interest. If instead the entire final distribution is required, for example to examine where the surviving probability mass ends up at the very end of a task sequence, AD is the algorithm that should be used.

Also, we recall that AD cannot be directly tuned to achieve a certain a priori precision. In order to increase precision, one has to increase the grid size, which exposes the precision gain a posteriori only. PP on the other hand knows its precision level a priori. Algorithm 3, which is called by PP on all possible traces is designed to achieve exactly the given precision. As a result, also PP does not deviate too far from this, so one has a pretty good handle on the a priori precision.

CHAPTER

5 Applications

This chapter showcases the applicability of the algorithms introduced in Chapter 4 in several contexts. In Section 5.1 we use the Markov Task Process modelling formalism to capture the energy-relevant behavior of a 2-unit CubeSat called GOMX–1 in a probabilistically aggregated manner, in order to analyse its energy budget up to a time horizon of an entire year. In addition, we conducted robustness and sensitivity experiments based on battery capacity, degraded solar input, load noise and battery aging, in order to determine to which degree the built-in battery was appropriately dimensioned.

Section 5.2 bridges the gap from simple analysis to scheduling based on remaining battery charge. We consider GOMX–3, a 3-unit CubeSat, and model its behavior and energy-relevant particulars using Priced Timed Automata in a modular and generic fashion with the aim to eventually determine optimal schedules up to a short time horizon with UPPAAL CORA. Since UPPAAL CORA suffers from expressive incapabilities of capturing most appropriate battery models, we account for this by introducing a synthesis-validation loop, that refutes and recomputes schedules, based on the kinetic battery model and its dedicated algorithms to assess depletion risk. The section concludes with three test runs of varying scheduling horizons, that were actually executed on GOMX–3. The corresponding satellite telemetry data shows that the almost fully automatic battery-aware scheduling approach indeed computes valid plans.

Finally, in Section 5.3 we build on top of the battery-aware scheduling approach to perpetually extend schedules with each satellite pass over the ground station. We incorporate a battery SoC estimation step, that continually corrects the estimated SoC based on downlinked satellite telemetry data to eliminate any potential bias or drift in the used battery model and the utilized parameters. A rudimentary test on old GOMX–3 data provides a glimpse of the potential improvement of this scheduling approach.

5.1 Energy Budget Analysis Of GOMX–1

In this section, we apply the results established in the previous sections in a concrete scenario. The problem is inspired by experiments carried out during the mission time of the then earth orbiting nanosatellite GOMX–1 [12].

The GOMX–1 Nanosatellite

GOMX–1 [12] is a Danish two-unit CubeSat mission launched in November 2013 to perform research and experimentation in space related to Software Defined Radio (SDR) with emphasis on receiving ADS-B signals from commercial aircraft over oceanic areas. As a secondary payload the satellite flies a NanoCam C1U color camera for earth observation experimentation. Five sides are covered with NanoPower P110 solar panels, and the power system NanoPower P31u holds a 7.4 V Li-Ion battery of capacity 5000 mAh. GOMX–1 uses a radio amateur frequency for transmitting telemetry data, making it possible to receive the satellite data with low-cost infrastructure anywhere on earth. The mission is developed in collaboration between GomSpace ApS, DSE Airport Solutions and Aalborg University, financially supported by the Danish National Advanced Technology Foundation. The empirical studies carried out with GOMX–1 serve as a source for parameter values and motivate the scenario described in the remainder of the section. We use the following data collected from extensive in-flight telemetry logs.

- One orbit takes 99 minutes and is nearly polar;
- The battery capacity is $\mathsf{cap} = 5000$ mAh;
- During 4 to 7 out of on average 15 orbits per day, communication with the base station takes place. The load induced by communication is roughly 400 mA. The length of the communication depends on the distance of the pass of the satellite to the base station and varies between 5 and 15 minutes;
- In each communication, the satellite can receive instructions on what activities to perform next. This influences the subsequent background load. Three levels of background load dominate the logs, with average loads at 250 mA, 190 mA, and 90 mA. These background loads subsume the power needed for operating the respective activities, together with basic tasks such as sending beacons every 10 seconds;
- Charging happens periodically, and spans around $\frac{2}{3}$ of the orbiting time. Average charging power is 400 mA;

The above empirical observations determine the base line of our modelling efforts, which interprets the statistical data as being of stochastic nature. We make the following assumptions:

- We assume constant battery temperature. The factual temperature of the orbiting battery oscillates between −8 °C and 25 °C on its outside. There is the (unused) on-board option to heat the battery to nearly constant temperature. Using an on-off controller, this would lead to another likely nearly periodic load on the battery, well in the scope of what our modelling formalism supports.
- A constant charge from the solar panels is assumed when exposed to the sun. The factual observed charge slowly decays. This is likely caused by the fact that solar panels operate better at lower temperature (opposite to batteries), but heat up quickly when coming out of eclipse.

- We assume a strictly periodic charging behavior. The factual charging follows a more complicated pattern determined by the relative position of sun, earth and satellite. There is no fundamental obstacle to calculate and incorporate that pattern.

- We assume a uniform initial charge between 70 % and 90 % of full capacity with identical bound and available charge. Since the satellite needs to be switched off for transportation into space, assuming an equilibrated battery is valid. Being a single experiment, the GOMX–1 had a particular initial charge (though unknown). The charge of the orbiting battery can only be observed indirectly, by the voltage sustained.

- We assume that the relative distance to the base station is a random quantity, and thus interpret several of the above statistics probabilistically. In reality, the position of the base station for GOMX–1 is at a particular fixed location (Aalborg, Denmark). Our approach can either be viewed as a kind of probabilistic abstraction of the relative satellite position and uncertainty of signal transmission, or it can be seen as reflecting that base stations are scattered around the planet. This especially would be realistic in scenarios where satellite-to-satellite communication is used.

- We assume that the satellite has no protection against battery depletion. In reality, the satellite has 2 levels of software protection, activated at voltage levels 7.2 V and 6.5 V, respectively, backed up by a hardware protection activated at 6 V. In these protection modes, various non-mission-critical functionalities are switched-off. Despite omitting such power-saving modes, we still obtain conservative guarantees on the probability that the battery powers the satellite.

Satellite Model

According to the above discussion, the load on the satellite is the superposition of two workload processes.

- A probabilistic load reflecting the different operation modes, modelled by a Markov Task Process $\boldsymbol{M}$ (see Section 3.6) as depicted in Figure 5.1.

- A strictly periodic charge load alternating between 66 minutes at 400 mA, and the remaining 33 minutes at 0 mA.

One can easily express the charging load as another independent Markov Task Process (where all jump probabilities are 1) and consider the sum load generated by these two processes in superposition (methods in Section 3.6 adapt straightforwardly to this setting).

The KiBaM in our model has the following parameters:

- The ratio of the available charge $c = 0.5$ (artificially chosen value as parameters fitted by experiments on similar batteries strongly vary [42, 21]);

- the diffusion rate $p = 0.0006$ per minute (we decreased the value reported by experiments [21] by a factor of 4 because of the low average temperature in orbit, 3.5 °C, and the influence of the Arrhenius equation [27]).

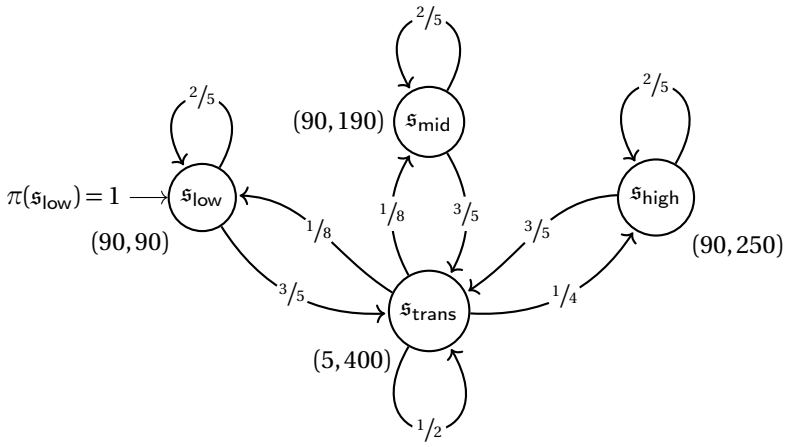


Figure 5.1: Markov Task Process of the load on the satellite. All noisy task annotations are composed of durations in minutes, as well as the mean of a normal distribution with standard deviation 5. The initial probability distribution is Dirac on $\mathfrak{s}_{\mathsf{low}}$, as indicated by the arrow. This load is superposed with a strictly periodic load modelling charge by solar power infeed.

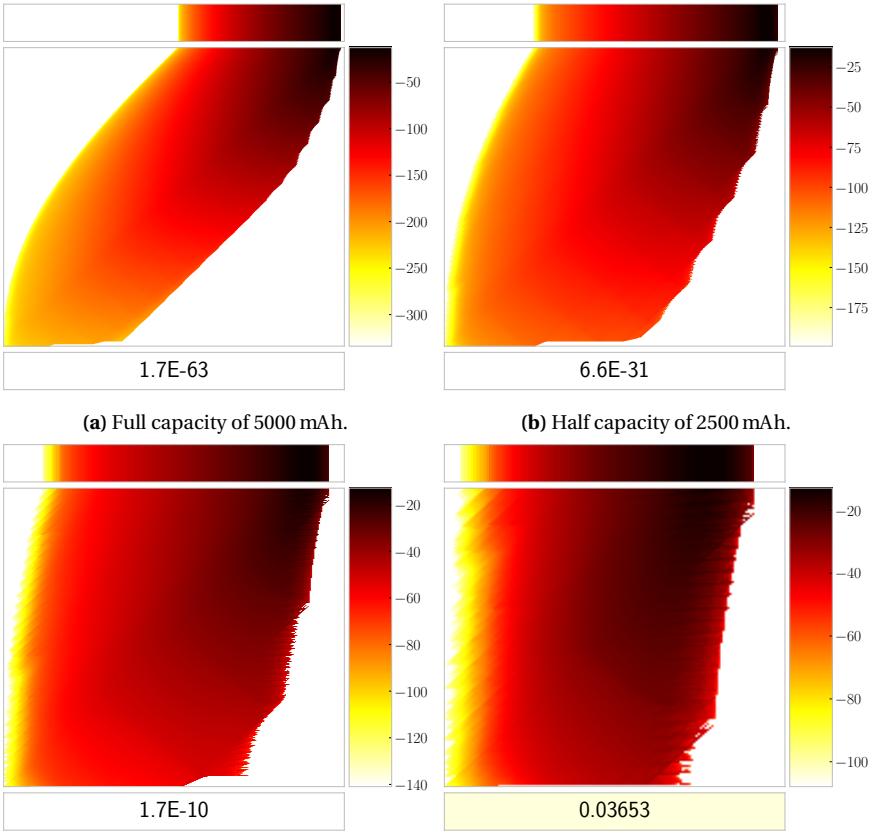
Analysis Implementation Aspects

We decided to use the static discretization algorithm (Section 4.2) to assess battery depletion risks. This decision makes sense, as the MTP in Figure 5.1 exhibits greatly varying loads, thus the support of the tracked SoC subdistributions is expected to cover large portions of the overall safe SoC space. We used grid dimensions $N = 1200, 600, 300$ and 150 for the experiments with the batteries of capacity 5000 mAh, 2500 mAh, 1250 mAh and 625 mAh, respectively to achieve equal relative precision. All the experiments have been performed on a machine equipped with an Intel Core i5-2520M CPU @ 2.50GHz and 4GB RAM. All values occurring are represented and calculated with standard IEEE 754 double-precision binary floating-point format except for the values related to the battery being depleted where we use arbitrary precision arithmetic (as this number keeps accumulating grid values that are of much lower order of magnitude). The number of subdistributions that must be kept track of simultaneously turned out to be no larger than 54.

Model Evaluation

We performed various experiments with this model, to explore the stochastic KiBaM technology introduced in Section 3.5. Here we report on five distinct evaluations, demonstrating that valuable insight into the model can be obtained.

1. The 5000 mAh battery in the real satellite is known to be over-dimensioned. Our aim was to find out how much. Hence, we performed a sequence of experiments, decreasing the size of the battery exponentially. The results (of the safe under-approximation) are displayed and explained in Figure 5.2.



(a) Full capacity of 5000 mAh.

(b) Half capacity of 2500 mAh.

(c) A quarter capacity of 1250 mAh.

(d) One eighth capacity of 625 mAh.

Figure 5.2: SoC under-approximation for different sizes of the satellite's battery after 1 year (Actually it is after 364 days, as this is in the middle of a charging phase. After 365 days the satellite is in eclipse and no density is exhibited along the capacity limit.). The leftmost SoC is with the original battery capacity, 5000 mAh. In each further plot, the battery capacity is halved, i.e. 2500 mAh, 1250 mAh, and 625 mAh. Note that all the densities are depicted on a logarithmic color scale (ticks in the colorbar stand for the order of magnitude). We observe that only the smallest battery provides insufficient guarantees. Its probability of depletion after 1 year is 0.0365; the probability decreases to $1.7 \cdot 10^{-10}$ already for the 1250 mAh battery. The smaller the battery, the more crucial is the distinction of available and bound charge as a larger area of the plots is filled with non-trivial density.

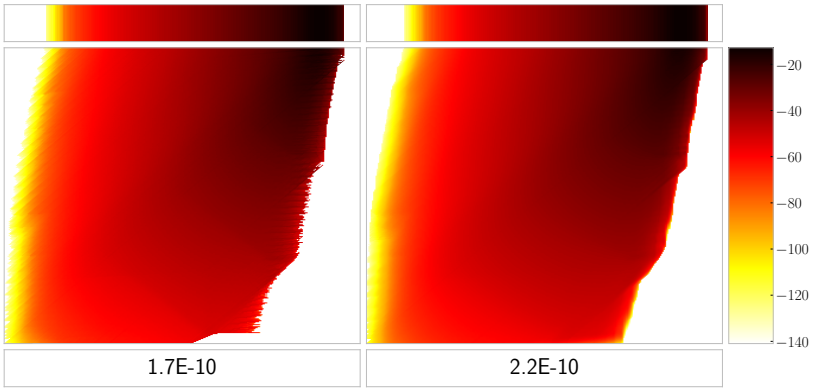


Figure 5.3: Load noise. SoC under-approximation of the 1 year run using the 1250 mAh battery with Dirac loads (left) and with noisy loads (right). We used Gaussian noise with standard deviation 5.

We found out that $\frac{1}{4}$ of the capacity still provides sufficient guarantees (since the depletion risk calculated is in the order of 10^{-10}) to power the satellite for 1 year while $\frac{1}{8}$ of the capacity, 625 mAh, does not. The following table compares the depletion risks $\mathfrak{z}^{\downarrow}$ and $\mathfrak{z}^{\downarrow}$ computed by the over- and under-approximations, respectively.

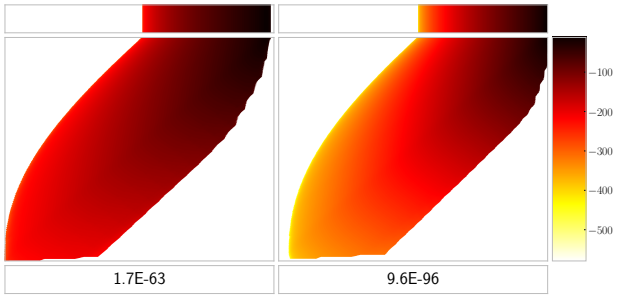
	Capacity (mAh)			
	5000	2500	1250	625
$\mathfrak{z}^{\downarrow}$	$9.61 \cdot 10^{-96}$	$4.69 \cdot 10^{-43}$	$1.01 \cdot 10^{-15}$	0.001 22
$\mathfrak{z}^{\uparrow}$	$1.66 \cdot 10^{-63}$	$6.58 \cdot 10^{-31}$	$1.73 \cdot 10^{-10}$	0.036 53

The approximations thus compute the real probability of depletion up to very small absolute errors ranging from 10^{-63} for the 5000 mAh battery to 0.036 53 for the 625 mAh battery.

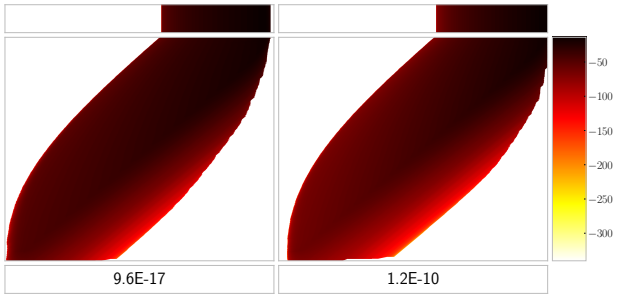
2. We compared our results with a simple linear battery model (LiBaM) of the same capacity.[1] This linear model is not uncommon in the satellite domain, it has for instance been used in the *Envisat* and *CryoSat* missions [11]. We obtain the following probabilities for battery depletion:

	Capacity (mAh)			
	LiBaM		**KiBaM**	
	5000	625	5000	625
$\mathfrak{z}^{\downarrow}$	$1.76 \cdot 10^{-144}$	$8.53 \cdot 10^{-16}$	$9.61 \cdot 10^{-96}$	0.001 22
$\mathfrak{z}^{\uparrow}$	$1.86 \cdot 10^{-84}$	$2.94 \cdot 10^{-8}$	$1.7 \cdot 10^{-63}$	0.036 53

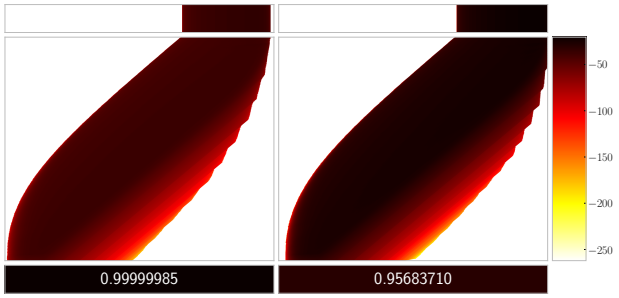
[1] The linear model can be emulated using a KiBaM with diffusion rate $p \to \infty$. This has the effect that available and bound charge wells behave equally and thus deplete synchronously. To compute the numbers we used the same algorithm and discretization constants as for the corresponding KiBaM of the same size.



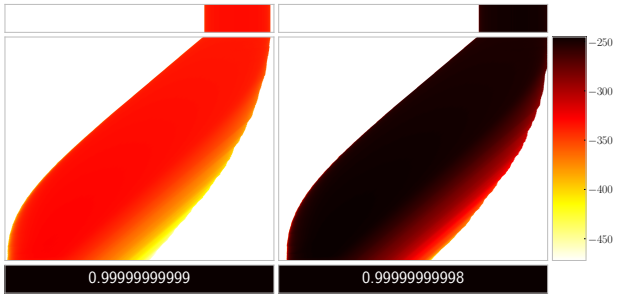
9 active solar panels (≈ 400 mA infeed rate).



8 active solar panels (≈ 355 mA infeed rate).



7 active solar panels (≈ 311 mA infeed rate).



6 active solar panels (≈ 266 mA infeed rate).

Figure 5.4: Number of active solar panels. The approximations of the full 5000 mAh battery with 9, 8, 7 and 6 solar panels, with the under/over-approximations being displayed on the left/right on a pairwise mutual colorscale. Again, the ticks of the colorscale represent the order of magnitude of the densities.

The linear model turns out to be surprisingly (and likely unjustifiably) optimistic, especially for the 625 mAh battery.

3. We (computationally) simplified the two experiments above by assuming Dirac loads. To analyze the effect of additive white noise on the loads, we compared the Dirac loads (tasks) with noisy loads (discretized noisy tasks) on the 1250 mAh battery. Each task with load ℓ is treated as a noisy task with g being a truncated gaussian with mean ℓ, that is discretized into 10 chunks, following the construction of Lemma 27. As expected, the noise **(i)** smoothens out the distribution a little, and **(ii)** pushes a bit more of the distribution to saturated as well as depleted states, see Figure 5.3.

4. Our reference satellite is a two-unit satellite, i.e. is built from two cubes, each 10 cm per side. In the current design, 9 of the 10 external sides are covered by solar panels, the remaining one is used for both radio antenna and camera. We thus conducted a robustness analysis with respect to solar infeed, by assuming that 1, 2 and 3 solar panels break down. Figure 5.4 displays that the satellite can easily deal with 1 defective solar panel. If additional panels fail, the system runs out of energy rapidly with high probability.

5. The stochastic KiBaM does not incorporate battery *aging*. In general, the degradation of a battery over time depends on many factors, most prominently how the battery was stored, which loads it was subjected to, how deeply it was discharged and at which temperatures it was used. We are not aware of a consensus method of how to model degradation of a Li-ion battery which is influenced by all of these factors. A measurable quantity related to battery age for our case study is the voltage drop when in eclipse. In-orbit measurements show that this voltage drop has worsened by 3% after one year of operation. For comparison purposes, we thus pessimistically assumed a battery with a capacity of only 4850 mAh (97 % of 5000 mAh) from the beginning. Compared to the 5000 mAh battery the depletion probabilities are only slightly higher:

	Capacity (mAh)	
	5000	4850
$\mathfrak{z}^{\downarrow}$	$9.61 \cdot 10^{-96}$	$1.73 \cdot 10^{-92}$
$\mathfrak{z}^{\uparrow}$	$1.66 \cdot 10^{-63}$	$5.58 \cdot 10^{-61}$

As of now (September 30th, 2021) GOMX–1 is still orbiting earth. To the best of the author's knowledge, the CubeSat is still able to fulfill its original purpose, i.e. demonstrating aircraft tracking via ADS-B. Some secondary payloads, like the NanoCam module to take pictures, GomSpace reported to have failed curtesy of a power surge, likely originating from a high-energy particle impact.

5.2 Battery-Aware Scheduling: The GOMX–3 Case

The GOMX–3 CubeSat was a 3 kg nanosatellite designed, delivered, and operated by Danish satellite manufacturer GomSpace. GOMX–3 was the first ever In-Orbit Demonstration (IOD) CubeSat commissioned by ESA. The GOMX–3 system used

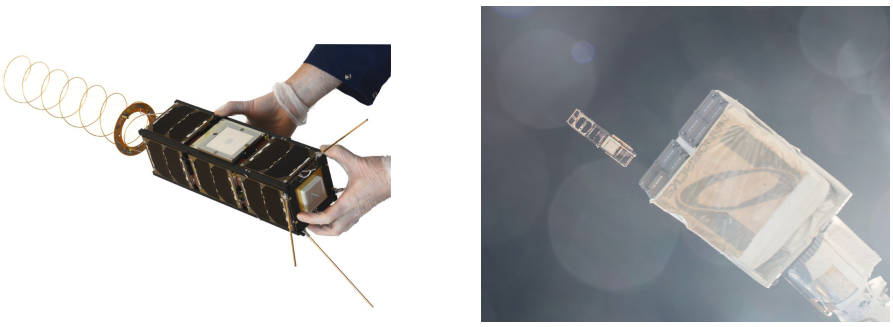


Figure 5.5: The final GOMX–3 nanosatellite (left) and its deployment from the ISS (right) together with AAUSAT5 (picture taken by Astronaut Scott Kelly).

Commercial-Off-The-Shelf (COTS) base subsystems to reduce cost, enabling to focus on payload development and testing. GOMX–3 was launched from Japan aboard the HTV–5 on August 19, 2015. It successfully berthed to the ISS a few days later. GOMX–3 was deployed from the ISS on October 5, 2015. Figure 5.5 shows the satellite and its deployment. Mission end was reached in October 2016, due to atmospheric reentry through gradual orbital decay. The rest of this section reflects the proceedings before mission end. Both GomSpace and ESA are interested in maximizing the functionality of their nanosatellite missions. As such, GOMX–3 has been equipped with a variety of technically challenging payloads and components, among them **(i)** 3-axis rotation and pointing **(ii)** in-flight tracking of commercial aircraft **(iii)** monitoring signals from geostationary INMARSAT satellites **(iv)** highspeed downlinking to stations in Toulouse (France) or Kourou (French Guiana).

For a satellite in orbit all resources are scarce and the most critical resource of all is power. Power is required to run the satellite, to maintain attitude, to communicate, to calculate, to perform experiments and all other operations. Detailed knowledge on the power budget is thus essential when operating a satellite in orbit. Furthermore, in a satellite not all power is used as it is generated. The satellite passes into eclipse (almost) every orbit and during those periods it must draw power from its batteries. This challenge is especially apparent for nanosatellites where not only the actual satellite but also the resources are very small. An operator of such a spacecraft is thus faced with a highly complex task when having to manually plan and command in-orbit operations constantly balancing power and data budgets. Manual operation was used prior to this work.

This section highlights how the (stochastic) KiBaM (Chapter 3) and its associated algorithms (Chapter 4) alongside formal modelling and verification technology can be used to provide support for commanding in-orbit operations while striving for an efficient utilization of spacecraft flight time.

Heterogeneous timing aspects, especially the aperiodic occurrences and variable durations of tasks, as well as the experimental nature of the application domain make it impossible to use traditional workload scheduling approaches, like RMS (*Rate Monotonic Scheduling*) and EDF (*Earliest Deadline First*) for periodic tasks.

The schedules we derive are tailored to maximize payload utilization while

minimizing the risk of battery depletion. The approach is flexible in the way it can express intentions of spacecraft engineers with respect to the finer optimization goals. At the end stands an automated two-step schedule synthesis procedure that provides quantifiable error bounds.

For the first step, we have developed a generic model of the GOMX–3 problem characteristics in terms of a network of Priced Timed Automata (PTA) as introduced in Section 2.3. This model is subjected to a sequence of analyses with respect to cost-optimal reachability (CORA) with dynamically changing cost and constraint assignments. We use UPPAAL CORA for this step. Note that UPPAAL CORA is limited to linear systems, therefore the simple linear battery model is used. Any schedule generated in this step has a risk of not being safe when used in-orbit, running on a real battery and with real payload.

To account for this problem, a second step validates the generated schedule, interpreted as a task sequence, using the stochastic KiBaM and its algorithms, to discriminate between schedules according to their quantified risk of depleting the battery. Low risk schedules are shipped to orbit and executed there.[2] The satellite behavior is tightly monitored and the results gained are used to improve the model as well as the overall procedure.

The entire toolchain has been developed, rolled out, experimented with, and tailored for in-the-loop use when operating the GOMX–3 satellite.

Modelling The GOMX–3 Nanosatellite

GOMX–3's mission payloads were threefold: Tracking of ADS-B beacons emitted by commercial airplanes, testing a high-rate X-Band transmitter module for in-space adequacy, and monitoring spot-beams of geo-stationary satellites belonging to the INMARSAT family, via an L-Band receiver. In addition, it featured a UHF software defined radio module for downlinking collected data to, and uplinking new instructions from, the GomSpace base station in Aalborg, Denmark. In the sequel, we refer to the operation of one of these payloads as a *job*. Each job comes with its own set of satellite attitude configurations (specifying its orientation in 3-dimensional space), making an advanced 3-axis attitude control system indispensable. This attitude determination and control system (ADCS) uses reaction wheels and magnetorquers to enable the satellite to slew into any required position. The ADCS is especially power-hungry.

As an earth-orbiting satellite, GOMX–3 naturally enters eclipse. To continue operation, it draws the necessary power to sustain its operation from an onboard battery system. These batteries are, in turn, charged by excess energy harvested during insolation periods by solar panels that cover any non-occupied surface. The solar intake in general depends on the so called β-angle, determining the degree of insolation per orbit and thus the portion in which the spacecraft is exposed to sunlight. Every attitude (as well as each transition phase between attitudes, i.e. slewing) has a constant assigned to it which represents a constant energy intake collected by the solar panels during insolation periods.

Special care needs to be taken when reasoning about battery performance, as the latter depends, among others, on ambient temperature. During eclipse, temperatures may fall below the operational limits of Li-ion batteries (according

[2] There was no actual risk of mission loss, as GOMX–3 had several FDIR mechanisms (*fault detection, isolation and recovery*) in place, for example a cascaded Watch-Dog-Timer system.

to manufacturer specification). To maintain a safe operating range, the GOMX–3 batteries are coated in heater foil enabling active temperature regulation (mostly during eclipse), at the cost of higher power consumption. A rich pool of diagnostic data from the earlier GOMX–1 mission however shows that the onboard batteries stay within the range of −7 to 17 °C. As a consequence the battery heaters in GOMX–3 are not used, and remain disabled at all times.

Another aspect of concern is that of gradual battery capacity degradation due to frequent, deep or very fast charge/discharge cycles. We work with the assumption that the scheduling horizon is so short that the battery will not degrade and thus retains a constant capacity limit along a schedule. Degradation can however be accounted for by gradually reducing the capacity along the lifetime of the mission, as the accumulated usage cycles increase. Having been ejected from the ISS, GOMX–3 roughly follows the orbit of the latter, albeit on a lower altitude. Therefore, insolation periods as well as operational windows for the different jobs are well predictable over the time horizon of about a week ahead, yet they are highly irregular.

Exploiting the pre-determined attitude configurations per mode of operation, the net power balance of every job can be predicted in conjunction with the GomSpace in-house POWERSIM tool. This tool provides orbit trajectory predictions including β-angle estimations. The aggregated information is considered trustworthy and captures the essence of the power-relevant behavior of GOMX–3 and serves as input to the scheduling approaches described in later sections.

In order to understand their joint implications for the energy budget of GOMX–3, it is important to accurately model these power-relevant aspects of the satellite components, and their interplay.

Objectives

In broad terms, the main mission goal of GOMX–3 is to maximize the amount of jobs carried out without depleting the battery. The concrete objectives spelled out by GomSpace engineers changed several times along the mission. This meant that the models have to have the necessary flexibility needed to reflect the requirements once they are made formal, not only during the design phase but especially while in orbit.

GOMX–3 switches to *Safe Mode* if the battery SoC falls below a given threshold. For GOMX–3 this threshold is at 40 % of the battery's capacity. In Safe Mode, all non-essential hardware components are switched off, preventing the satellite from being productive. Only UHF radio, ADCS and the onboard computer remain operational. The primary objective is thus to avoid Safe Mode, while maximizing secondary objectives. This Safe Mode threshold thus implictely induces the depletion threshold depl of the battery model. Several secondary objectives need to be taken into account.

- Whenever possible the UHF connection to the GomSpace base station must be scheduled and maintained throughout the entire operational window in order to enable monitoring the status of GOMX–3 and to uplink new instructions if need be. This is crucial to maintain control over the satellite and thus considered vital for the success of the mission.

- Independent of the satellite attitude, the ADS-B helix antenna is able to receive ADS-B beacons. Thus this hardware module will be active at all times, thereby constantly collecting data of airplane whereabouts.
- The X-Band windows are small, as the downlink connection can only be established if the satellite is in line of sight and close enough to the receiving ground station. The corresponding downlink rate, however, is relatively high.
- L-Band jobs have highly variable durations depending on how the satellite crosses the field of visibility of the INMARSAT, and will collect a lot of data if successful. The variations in window lengths can be observed in Section 5.2, where actual schedules are visualized.
- L-Band jobs are to become as balanced as possible across the available INMARSATs.
- Jobs filling their entire job window are most valuable. Jobs that have been aborted early or started late are not considered interesting.
- L-Band and X-Band jobs are mutually exclusive, as they require different attitudes. UHF jobs may be scheduled regardless of the current attitude, even when L-Band or X-Band jobs are currently executed.
- Only downlinked data are useful, thus the time spent on data collection payloads (L-Band, ADS-B) and downlink opportunities (X-Band) needs to be balanced in such a way that only a minimal amount of data needs to be stored temporarily in the satellite's memory. This induces the need to weigh the data collection rate and the downlink speed against each other. Based on these observations and the expertise of GomSpace engineers, it was deemed that two fully executed X-Band jobs are enough to downlink the data of one successful L-Band job together with the ADS-B data collected in the meantime.

The *ground track* of GOMX–3 visualizing its orbit and operational windows, is depicted in Figure 5.6.

GOMX–3 PTA Network

As the central modelling formalism Prized Timed Automata (PTA) are employed (see Section 2.3) when modelling the behavior of GOMX–3, with special emphasis being put on flexibility with respect to the optimization objective. In order to allow for easy extensibility, the modelling was purposely kept modular, parametric and generic. Notably, the TA formalism is not expressive enough for the nonlinearities of the kinetic battery model. Therefore we use a simple linear model instead, and account for this discrepancy at a later stage. The component models belong to the following categories.

Background load comprises the energy consumption of modules that are always active, including the ADS-B module to receive airplane location beacons, the reaction wheels and magnetorquers (even though not at full power) for keeping the attitude invariant.

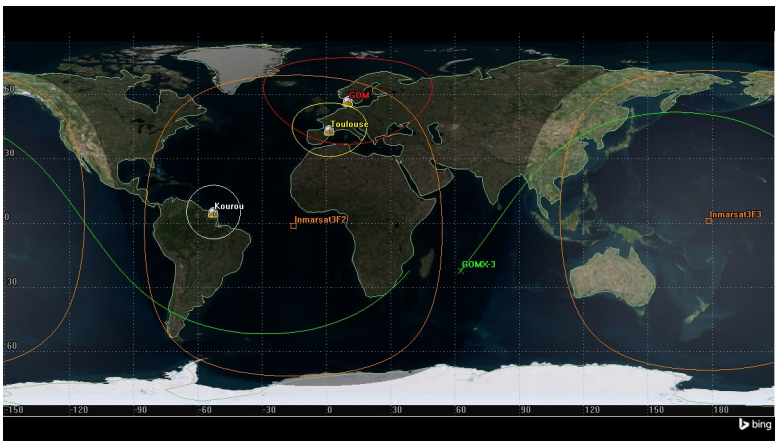


Figure 5.6: The ground track of GOMX–3. X-Band operational windows are induced when the satellite trajectory crosses the `Kourou` or `Toulouse` area, L-Band jobs can be carried out in the `Inmarsat3F2/3` areas. The `GOM` area represents the line of sight of the GomSpace station in Aalborg, Denmark.

Attitude Control represents the predetermined attitude requirements of each job and the worst case slewing time of 5 minutes. However, preparing the ADCS and actual slewing can be abstracted into one single *warm-up* period, since both stages consume the same amount of energy. Thus, the worst-case warm-up time before entering the visibility range of any job window requiring slewing was raised to 10 minutes.

Jobs are dealt with in a generic way, so that only the common characteristics are modelled. A job has a finite time window of operation, it may be skipped, it may require an a priori *warm-up* time (to ramp up the physical modules related to the job, especially the ADCS) as well as a specific attitude, it may need to activate a set of related modules inducing sequences of tasks.

Battery represents a relatively simple linear battery which can support sequences of tasks. It keeps track of its (one-dimensional) state of charge through updates based on the currently executed task. Since the battery is modelled as an automaton, the system can monitor and take decisions based on the remaining battery charge.

Sun is an intuitive two-location automaton (representing insolation and eclipse) based upon the predicted insolation times, triggering a constant energy infeed from the solar panels which in turn depends on the satellite's attitude. The automaton includes that upon changing location, i.e. entering/exiting eclipse or battery heaters being activated/deactivated. The latter are however not used.

The accumulative power consumption/infeed of GOMX–3 in mW is summarized in the following table.

Type	Name	Description
`int[0,15]`	`module_id`	A type to refer to the 15 modules present in GOMX–3
`int[0,149760000]`	`soc_t`	A type to refer to SoCs. The limits are the absolute depletion as well as the capacity limit
`int[0,4]`	`attitude_t`	A type to refer to the 4 different attitudes GOMX–3 can attain (plus one dummy attitude)
`int[0,5]`	`p_id`	A type to refer to payloads. There are 6 payloads, since both INMARSAT payloads are separated into odd and even occurrences, due to technical reasons

Table 5.1: Type definitions (`typedefs`) in the UPPAAL GOMX–3 model.

Background load	Warm-Up	X-Band	L-Band	UHF	Solar infeed
2989	1406	11945	3863	2630	[5700,6100]

Among these components, the PTAs modelling the battery and the job aspects are the most interesting. The automata modules and the global variables used by the automata are explained in more details below.

Global Declarations

In the following we list and explain the global variables of the UPPAAL system of GOMX–3.

Types: A list of type defintions is outlined in Table 5.1.

Channels: A list of declared channels is summarized in Table 5.2.

Constants: A list of (`const`) variables that remain constant throughout the schedule synthesis procedure is given in Table 5.3.

Transient Variables: Variables subject to change during the scheduling algorithm, and thus relevant to the dynamic behavior of the satellite, are detailed in Table 5.4.

Global functions: A List of globally declared functions are shortly explained in Table 5.5. The actual code of the functions is given in Listing 1.

Now that variable declarations are in place, let us take a closer look at the actual structures of the automata involved, how and where transient variables are manipulated.

`JobProvider`**:** This automaton provides the interface between multiple arrays representing the job opportunities as well as their implied preheating times, and the actual `Job` automaton. It is depicted in Figure 5.10. It waits for a job window,

Name	Description
`alignTo[attitude_t]`	An array of channels to notify that an attitude change needs to happen
`reached`	A channel to notify when the attitude change is complete
`bUpdate`	A channel to trigger a battery state update
`preHeat[p_id]`	An array of channels to notify that a warm-up period is about to start for a certain payload
`available[p_id]`	An array of channels to notify when a job window of a certain payload is about to be entered
`not_available[p_id]`	An array of channels to notify when a job window of a certain payload is about to be left

Table 5.2: Channels (declarations using the `chan` keyword) declared in the GOMX–3 model.

Name	Type	Description
`orbit_period`	`int`	The duration of an orbit. Initially set to 5570 (seconds).
`power_m[module_id]`	`load_t`	An array storing the power consumption of every module.
`bg_dc[module_id]`	`load_t`	An array storing the background duty cycle of each module.
`default_bg_load_po`	`int`	The power consumed through background load on a single orbit, initially $\prod_i$ `bg_dc[i]`$\cdot$`power_m[i]`.
`default_bg_load`	`load_t`	The default background load. Initially `default_bg_load_po/orbit_period`.
`capacity`	`soc_t`	The capacity of the battery. For GOMX–3 this is 149 760 000.
`safe_threshold`	`soc_t`	The safe threshold for the battery. For GOMX–3 this is 82 368 000, or 55% of `capacity`.
`a_N, a_Z, a_Y, a_I`	`attitude_t`	Aliases for the attitudes for easier use, conceptually standing for *nominal attitude*, *+Z to nadir*, *+Y to nadir*, *aligned to* INMARSAT.
`power_g[attitude_t]`	`load_t`	An array storing the power consumption rates of GOMX–3 for each attitude.
`pp[attitude_t][attitude_t]`	`int`	`pp[src][dst]` holds the warm-up duration from attitude `src` to attitude `dst`.
`slp[attitude_t][attitude_t]`	`int`	`slp[src][dst]` holds the time duration it takes to transition from attitude `src` to attitude `dst`.
`power_slewing`	`load_t`	The load induced by slewing.
`a_nfe[p_id]`	`int`	An array holding the attitudes required for each job.
`power_p[p_id]`	`load_t`	An array holding the power consumption per payload.
`access_preheat[p_id][n]`	`int`	A matrix holding a row of warm-up starting times for each job type. Each row is of length n, where n is not known until the input tables have been parsed.
`access_start[p_id][n]`	`int`	A matrix holding a row of job window starting times for each job type.
`access_end[p_id][n]`	`int`	A matrix holding a row of job window end times for each job type.
`eclipse_end[m]`	`int`	An array of length m, holding the time points of insolation entrances, or equivalently, eclipse exits. The length of this array is not known until the input tables have been parsed.
`insolation_end[m]`	`int`	An array of length m, holding the time points of eclipse entrances, or equivalently, insolation exits.

Table 5.3: Constants (`const` declared variables) used in the UPPAAL CORA model of GOMX–3.

```
int jobPreheatTime(const p_id pid) {
  return access_preheat[pid][c[pid]];
}

int jobStartTime(const p_id pid) {
  return access_start[pid][c[pid]];
}

int jobEndTime(const p_id pid) {
  return access_end[pid][c[pid]];
}

int eclipseEndTime() {
  return eclipse_end[sun_c];
}

int insolationEndTime() {
  return insolation_end[sun_c];
}

int jobDuration(const p_id pid) {
  return
    jobEndTime(pid) - jobStartTime(pid);
}

int costRate(const p_id pid) {
  if (pid == 0) return c⊥;
  if (pid == 1) return cUHF;
  if (pid == 2) return cX;
  if (pid == 3) return cL;
  if (pid == 4) return cL;
  if (pid == 5) return cL;
  if (pid == 6) return cL;
}

bool job_possible(const p_id pid) {
  if (costRate(pid) == 0) return false;
  if (pid == 0) return true;
  if (ac_lock) return false;
  return true;
}

bool skippable(const p_id pid) {
  return true;
}

bool isAligned(const p_id pid) {
  if (a_nfe[pid] == 0) return true;
    //independent
  return a == a_nfe[pid];
}

bool hasToSlewBack(const p_id pid) {
  if (a_nfe[pid] == 0 || a == a_N)
    return false;
  else
    return true;
}

void lockIfNeeded(const p_id pid) {
  if(a_nfe[pid] != 0) ac_lock = true;
}

void unlockIfNeeded(const p_id pid) {
  if (a_nfe[pid] != 0) ac_lock = false;
}

void startJob(const p_id pid) {
  pa[pid] = true;
}

void endJob(const p_id pid) {
  pa[pid] = false;
  nee[pid]++;
}

void update(const p_id pid) {
  new_time = jobEndTime(pid);
  c[pid]++;
  if (new_time > jobPreheatTime(pid)) {
    a=1/0; //provoke error, discard trace
  }
}

void last() {
  new_time = tlast ;
}

void termCost() {
  // function stub
}
```

Listing 1: The UPPAAL code of the system's globally declared functions.

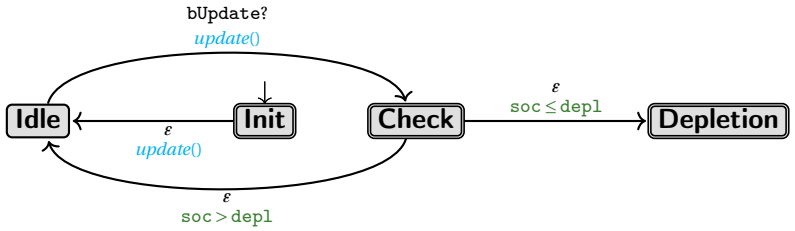


Figure 5.7: The `Battery` automaton.

Name	Type	Description
`gc`	`clock`	The global clock.
`soc`	`soc_t`	The variable storing the current SoC.
`a`	`attitude_t`	The current attitude of the satellite. Initially this is the nominal attitude $a_N = 1$.
`c[p_id]`	`int`	An array holding counters for each job. Once a job window is entered, the respective counter is incremented.
`nee[p_id]`	`int`	An array holding counters for each payload type. Once a job window is scheduled, the respective counter is incremented.
`slewing`	`bool`	A boolean variable indicating whether the satellite is currently changing attitude.
`insolation`	`bool`	A boolean variable indicating whether the satellite is currently exposed to sunlight.
`new_time`	`int`	A variable used to keep track of time and enabling computation with relevant time points. Whenever a job window is entered, this variable is assigned to the end point of that window such that battery state updates can be reliably executed.

Table 5.4: Transient variables (non-`const` declared variables) of the GOMX–3 UPPAAL model subject to change during the scheduling process.

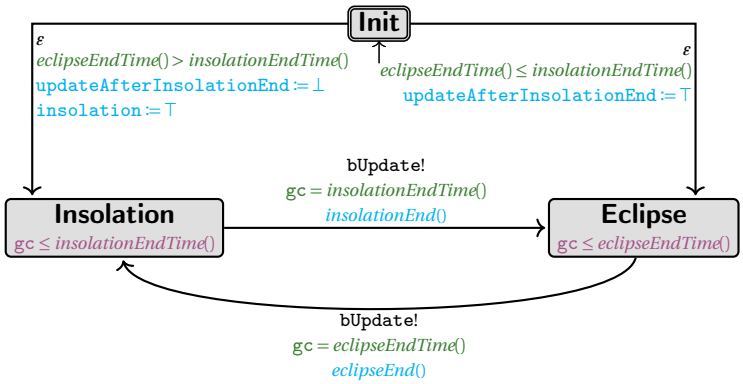


Figure 5.8: The Sun automaton.

Declaration	Description
int *jobPreheatTime*(const p_id pid)	A function to conveniently fetch the warm-up starting point of the next job window of type pid.
int *jobStartTime*(const p_id pid)	A function to conveniently fetch the entry point of the next job window of type pid.
int *jobEndTime*(const p_id pid)	A function to conveniently fetch the exit point of the current job window of type pid.
int *eclipseEndTime*()	A function to conveniently fetch the exit point of the eclipse window.
int *insolationEndTime*()	A function to conveniently fetch the exit point of the next insolation period.
int *jobDuration*(const p_id pid)	A function to conveniently fetch the exit point of the next eclipse period.
int *costRate*(const p_id pid)	Returns the cost rate of jobs of type pid.
bool *jobPossible*(const p_id pid)	Decides whether job can in principle be scheduled, with respect to system variables.
bool *skippable*(const p_id pid)	Function that decides whether a job can in principle be skipped, with respect to system variables. This can be extended to implement heuristics.
bool *isAligned*(const p_id pid)	Returns whether the satellite has the required attitude to execute a job of type pid.
bool *hasToSlewBack*(const p_id pid)	Returns whether the satellite needs to slew back to nominal attitude after finishing a job of type pid.
void *lockIfNeeded*(const p_id pid)	A function that locks the satellite, and thus prevents it from accepting jobs, depending on the attitude needed for the payload pid.
void *unlockIfNeeded*(const p_id pid)	A function that unlocks the satellite, and thus enables it to accept jobs.
void *startJob*(const p_id pid)	Bookkeeping for when a job is actually scheduled at the time the job window is entered.
void *endJob*(const p_id pid)	Bookkeeping for when the satellite leaves a job window that was scheduled.
void *update*(const p_id pid)	Updates timing related system variables.
void *last*()	A function that does the bookkeeping for the final step before the system deadlocks when the scheduling horizon is reached.
void *termCost*()	A function that realizes terminal costs to be added at the very end of the scheduling horizon. In all of our experiments, this was unused.

Table 5.5: Global functions of the GOMX–3 UPPAAL model.

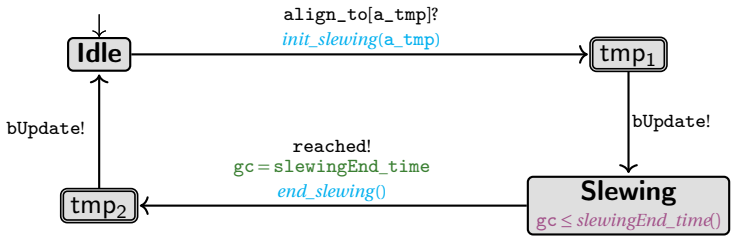


Figure 5.9: The `Attitude Control` automaton.

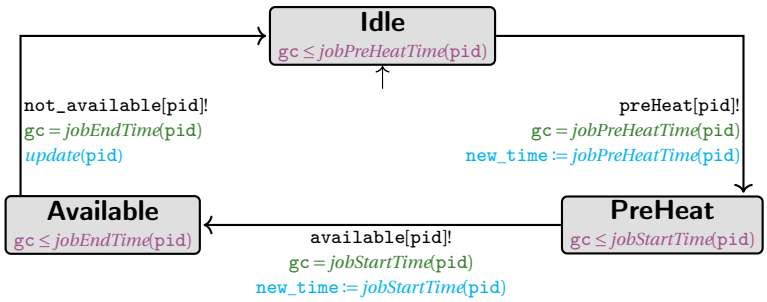


Figure 5.10: The `Job Provider` automaton.

discriminating whether the job needs a warm-up period or not, and broadcasts signals triggering the actual decision making. In the Idle location, being initial, it waits for the global clock `gc` to hit a certain job warm-up time event (stored in the array `jobPreheatTime`), sets the `time` variable to the current time, and notifies the `Job` automaton to start preheating over a dedicated `preHeat[pid]` channel, where `pid` uniquely identifies a certain job type. Upon this notification it switches into the PreHeat location and waits for the actual job to start, i.e. the global time reaching the expected start time of the job identified by `pid`, consequently transitioning into location Available, where in turn, it waits for completion of the job (`gc` reaching `jobEndTime[pid]`), switching into location Idle yet again, all the while notifying its environment on the respective dedicated channels. This module has no further local functions or variables.

`Job`**:** This automaton represents the execution or skipping of a job. Its graph representation is shown in Figure 5.11. It starts in its Idle location, waiting to be notified of impending preheating duties. At this point the take-or-skip decision is taken, as witnessed by the two outgoing transitions into locations labelled Skip and Align. A job is either skipped because it is not optimal to execute it, or because the attitude requirements don't match the current attitude of the satellite because of an already ongoing job. If the job is skipped, cost is accumulated with rate *costRate*(`pid`) over the duration of the job, effectively returning into location Idle. If it is taken, attitude requirements of the scheduled job are checked via the guard *isAligned*(`pid`), upon which the satellite starts slewing (location Slewing) to the

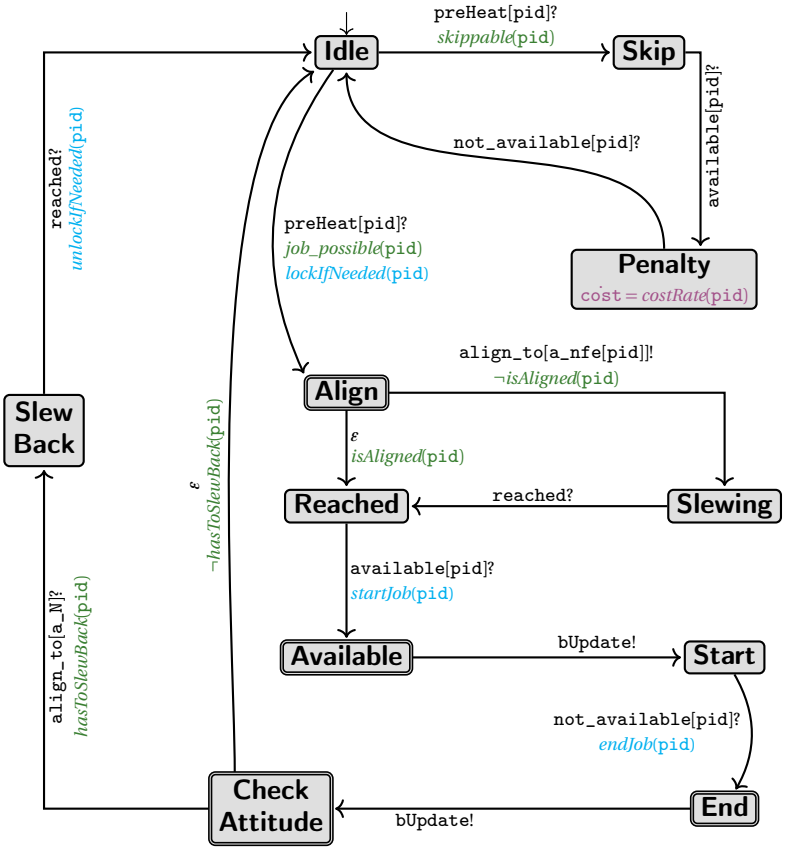


Figure 5.11: The `Job` automaton.

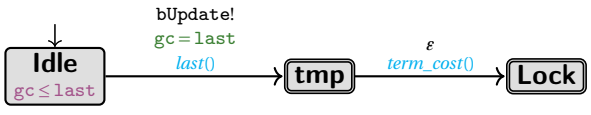


Figure 5.12: The `Termination` automaton.

```
int load = 0;
int old_time = 0;

void update() {
    soc -= load * (new_time - old_time);

    if (soc > max) { //align to limit
        soc = max;
    }

    old_time = new_time;
    load = default_bg_load;

    for (pid:p_id) {
        load += pa[pid] ? power_p[pid] : 0;
    }

    load += slewing ? power_slewing : 0;
    load -= insolation ? power_g[a] : 0;
}
```

Listing 2: The UPPAAL code of the battery module.

correct attitude (location Correct_Attitude) if need be. Upon notification, the job is executed (Start → End → Check_Attitude) triggering the battery via channel `bUpdate` to update its SoC, and finally checks whether it has to change attitude to minimize atmospheric drag using guards *hasToSlewBack*(`pid`) to finally return eventually to location Idle. This module has no further local functions or variables.

`Battery`: A visual representation of the automaton is found in Figure 5.7. This model represents a simple linear battery with capacity limits that can be charged and discharged with piecewise constant loads. It is notified of load changes via channel `bUpdate`, upon which it calls its local *update*() function and computes the length of a constant load interval via global integer variables `new_time` and `old_time`, and subtracts the result multiplied by `load` from its internal SoC `soc`, upon which it ends up in location Check. A check is performed whether the SoC fell below depletion threshold `depl`, upon which we either transition into (and stay in) the Depletion location or return to Idle to power another task. The full code of the battery module is listed in Listing 2.

`Attitude Control`: The attitude control system is a very simple cyclic automaton that essentially governs the slewing times and triggers battery state updates because of the activation of the reaction wheels and its implied load change. It is depicted in Figure 5.9. Its initial location is Idle, from where it waits to be signaled on channel `align_to[a_tmp]` (triggered by a `Job` automaton) that an attitude change is necessary. It sets the necessary variables while it transitions into the Tmp1 location, where it immediately continues to notify the battery that its load

```
Int slewingEnd_time = 0;
attitude_t a_dst = 0;

void init_slewing(const attitude_t a_tmp) {
    a_dst = a_tmp;
    slewing = true;
    slewingEnd_time = new_time + sp(a,a_tmp);
}

void end_slewing() {
    a = a_dst;
    a_dst = 0; //reduces state space
    new_time = slewingEnd_time;
    slewingEnd_time = 0; //reduces state space
    slewing = false;
}
```

Listing 3: The UPPAAL code of the attitude control module.

has changed and a SoC update is in order on channel `bUpdate`. The invariant forces the automaton to wait the necessary amount of time for the slewing to be terminated, upon which it broadcasts this fact on channel `reached`, while appropriately changing variables and transitioning into location Tmp2. Its last step before awaiting a new attitude change is to notify the battery of yet another load change, due to the completed slewing, on channel `bUpdate`. The full code of the attitude control module is listed in Listing 3.

Sun: The sun automaton is a three location automaton that in essence alternates between Insolation and Eclipse locations and notifies the battery automaton of the change in solar infeed through the `bUpdate` channel after waiting for the appropriate amount of time. It starts off initially in the Init location simply to decide whether we started in insolation of eclipse. The automaton is depicted in Figure 5.8. The code of the local functions is listed in Listing 4.

`Termination`: The termination automaton is a a purely technical automaton ensuring termination of the model checking process by deadlocking the system (location Lock) when the scheduling time horizon is reached, since the UPPAAL tool doesn't do so implicitly in certain cases. It enables the addition of terminal bookkeeping and of terminal costs through the *term_cost*() function, that, while an interesting idea, was largely unused in our scenarios. Its graph is depicted in Figure 5.12. No further local functions or variables are associated with this automaton.

Cost Model And Reachability Objectives

In the following we explain how the objectives derived by GomSpace engineers were turned into constraints and cost parameters of the PTA model.

```
bool updateAfterInsolationEnd = 0 ;

void eclipseEnd() {
    insolation = true;
    new_time = eclipseEndTime();
    if(!updateAfterInsolationEnd) sun_c++;
}

void insolationEnd() {
    insolation = false;
    new_time = insolationEndTime();
    if(updateAfterInsolationEnd) sun_c++;
}
```

Listing 4: The UPPAAL code of the sun model.

The Safe Mode threshold is kept variable and must be set before scheduling. It appears as `depl` in the automata models. Depending on the degree of aggressiveness of the intended schedule, it can either be set close to the real Safe Mode threshold of 40 % or it can be set higher, for example to 55 %, thereby adding an implicit safety margin.

UPPAAL CORA computes cost-minimal schedules. Therefore, we interpret the price annotations of PTA transitions as penalties for skipped jobs. Likewise, cost rates in states accumulate penalty per time unit a job window is left unused. An optimal schedule will then have the property that a minimal portion of important job windows was left unexploited.

An immediate consequence of this setup is that UHF jobs have a high penalty if skipped, as they are supposed to be scheduled every time they are possible. For L-Band and X-Band jobs, the number of jobs scheduled should result in an average ratio of $\frac{1}{2}$, according to the GomSpace directives. To arrive there, we proceed as follows. Let Δ_X and Δ_L denote the job windows length expectations of X-Band and L-Band jobs, respectively. Then the cost rate for skipping L-Band and X-Band window portions is set $2 \cdot \Delta_X$ and Δ_L, respectively. Likewise, the L-Band jobs on different INMARSAT are internally viewed as different jobs. Their cost rates for skipping should be set equal.

As explained in Section 2.3, in order to generate an optimal schedule from the network of PTAs up to a certain time horizon, we need to define the goal set of states to be used in a reachability objective as supported by UPPAAL CORA. In our case, the time horizon is to some extend already encoded in the `Termination` automaton. Since we want to make sure that some final bookkeeping is executed when the time horizon is reached, we explicitly query for the deadlock location `Lock` of this automaton in the query, i.e. $\exists\Diamond$`Termination.Lock`.

Model Quality Assurance

In light of the high overall significance of the GOMX–3 mission, it was from the start deemed important to assure the adequacy of the formal models used to represent

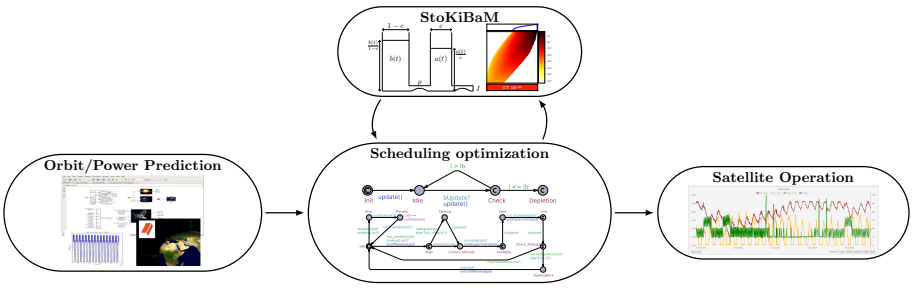


Figure 5.13: Scheduling workflow.

and to eventually manoeuvre the satellite. For this reason, a series of dedicated workshops were organized in the context of the SENSATION project , comprising, among others, the author of this document (see [33] for a snapshot of the website). On these occasions, presentations of varying technical detail were delivered by both sides, so as to expose the formal approach, the set of concrete problems, as well as possible solutions thereof. In later stages, collaborative work was organized via Google docs and Skype, which indeed provided an effective way to communicate feedback in both directions. This altogether made it possible to effectively cross-fertilize the domain expertise of the GomSpace engineers with the modelling and verification experience at Saarland University, so as to assure a high quality model. In the same vein, the design of the entire scheduling workflow (explained next) was a consensus decision.

The Scheduling Workflow

The scheduling workflow, depicted in Figure 5.13, loops through a two-step procedure of schedule generation and schedule validation. The latter is needed to account for the inaccuracies of the simple linear battery model, which is used for schedule generation, relative to real battery kinetics. Therefore any generated schedule is validated along the stochastically enhanced KiBaM known to be sensitive to such effects. If the validation does not exhibit good enough guarantees in the eyes of the GomSpace engineers, the current schedule is discarded and excluded from the generation step, and a new schedule is computed. Otherwise it will be accepted, upon which we break the loop and ship the schedule to orbit.

Operational Windows

The operational windows are the main source of variability in the scheduling workflow. They are determined by GomSpace engineers just before the start of the actual scheduling period. In this way, changing mission parameters and temporary unavailability of certain jobs is accounted for. The operational windows are delivered as `csv` (*comma separated values*) files based on the POWERSIM orbit prediction. If a job type is unavailable during a certain time span, the tables will not contain operational windows for this job for that span.

Each `csv` file contains a list of job opportunities of a certain type, for example L-Band (see table below), given by two date-time strings representing the start

time and the end time of the job window respectively, the implied duration of the time points, as well as a flag (`Scheduled`) that shows whether the opportunity should be taken. This column is filled in after a schedule has been computed. One such table could look as follows:

Access	Start Time (UTC)	Stop Time (UTC)	Duration (sec)	Scheduled
1	17 Nov 2015 00:38:38.922	17 Nov 2015 01:09:42.642	1863.720	–
2	17 Nov 2015 02:16:24.134	17 Nov 2015 02:45:23.914	1739.781	–
⋮	⋮	⋮	⋮	⋮
15	17 Nov 2015 23:41:20.490	18 Nov 2015 00:12:38.983	1878.493	–

Schedule Generation

The mission times to be considered for automatic scheduling span between 24 and 72 hours, i.e. the range of 15 to 47 orbits. Longer durations are not of interest since orbit predictions are highly accurate only for a time horizon of a handful of days, and because GOMX–3 is as a whole an experimental satellite, requiring periods of manual intervention. However, even a 24 hour schedule computation constitutes a challenge for plain CORA, since the number of states grows too large to fit in memory, due to the natural state space explosion when tackling model checking problems. [3]

The dominating contributor to the state space growth is the large number of L-Band operational windows. GOMX–3 passes several of them each orbit, invoking a take-or-leave decision for each such window. This issue is made worse as other jobs (i.e. UHF) could be taken at the same time as their attitude requirements are not mutually exclusive. Especially these circumstances induce an exponential growth of states in the length of the scheduling horizon.

Heuristics. The state-space explosion can, to certain extend, be remedied by using heuristics, i.e. exclusion of certain schedules at the risk of losing optimality. Here is a brief overview of heuristics used:

1. **Take every job if battery is almost full.** Job opportunities will be taken if the battery is close to being full, since the battery cannot store more energy anyway. This minimizes the risk of not being able to harvest energy due to a full battery.

2. **Force discard of schedules on depletion.** This simple, yet effective heuristic forces the PTA network into a dedicated deadlock location (Depletion) whenever the battery automaton reaches a non positive SoC, resulting in the schedule to be dropped.

3. **Impose upper bound on discharging loads.** This heuristic discards traces that exhibit high discharging loads at any point. It is meant to somewhat counteract the rate-capacity effect of batteries.

[3] Unfortunately UPPAAL CORA is a 32 bit executable and is thus unable to use more than 4GB of RAM

The following heuristics are specific to objectives expressed by the engineers.

4. **An L-Band job precedes two X-Band jobs.** To avoid storage of large amounts of data on the satellite, we bound the ratio of data collection and downlink jobs. A ratio $\mathtt{r}_\mathtt{X}/\mathtt{r}_\mathtt{L}$ can be approximated greedily by adding a global variable $\mathtt{r}$ (initially 0) as well as guards to the `Job` automaton such that X-Band jobs are scheduled only if $\mathtt{r} \geq \mathtt{r}_\mathtt{L}$ and L-Band jobs are scheduled only if $\mathtt{r} < (\mathtt{r}_\mathtt{X} + \mathtt{r}_\mathtt{L}) \cdot \mathtt{r}_\mathtt{X}$. Upon scheduling an X-Band and L-Band job, we set $\mathtt{r} \coloneqq \mathtt{r} - \mathtt{r}_\mathtt{L}$ and $\mathtt{r} \coloneqq \mathtt{r} + \mathtt{r}_\mathtt{X}$ respectively. With $\mathtt{r}_\mathtt{X} \coloneqq 2$ and $\mathtt{r}_\mathtt{y} \coloneqq 1$ schedules never start with an X-Band job and in the long run, the ratio of L-Band and X-Band jobs stays between 1 and $\frac{1}{2}$.

5. **Keep L-Band jobs in balance across INMARSATs.** Similarly to the realization of the above heuristic we bound the difference among L-Band jobs on the relevant INMARSATs to at most 2.

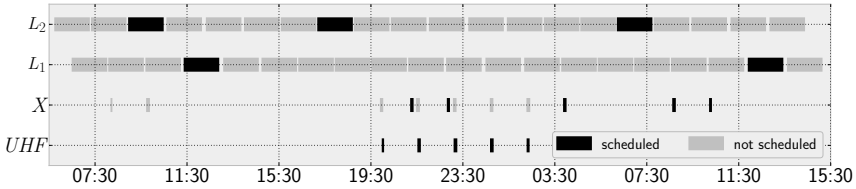
6. **Always schedule UHF jobs.** Instead of penalizing skipped UHF jobs by annotations of large costs (to enforce their scheduling), we enforce them on the automaton level, omitting any cost annotation.

Especially heuristic 3 proves useful in several ways. First, the KiBaM used in the validation step yields less energy before depletion if subjected to high discharging loads due to the rate-capacity effect (that is not captured by the linear battery model). Second, high loads are reached when UHF jobs are scheduled in addition to an L- or X-Band job. Such situations seem lucrative to UPPAAL CORA, given that they don't accumulate much cost. Yet, they often result in schedules that drain the battery very fast. Third, the bound can be chosen such that parallel jobs, and thus high loads, occur only during insolation, but not in eclipse.

Each heuristics impacts the computational efficiency (runtime, memory) as well as the schedule quality (accumulated cost), as it essentially prunes the state space. To illustrate this, we synthesized a schedule, deactivating one heuristic at a time, and report on some diagnostic quantities of the model checking procedure in the following table. The scheduling horizon was split into two parts and later conjoined in order to actually arrive at a schedule.

heuristics excluded	**CORA time** (s)	**# states explored**	**accumulated cost**
none	2.6	172 452	262 792
1	10.2	700 429	262 792
2	80.7	5 474 775	262 792
3	86.1	6 029 126	243 269
4	8.9	592 233	258 081
5	3.7	224 517	262 792
6	2.7	175 191	262 792

It becomes apparent that heuristic 2 and 3 are the most effective. Most of the combinations studied induce the schedule depicted below, where job windows of a certain type, i.e. L-Band on different INMARSATs (L_1, L_2), X-Band (X) and UHF, are displayed as black (grey) bars if they were indeed taken (skipped).



At first sight, dropping heuristics 3 or 4 lead to superior solutions. Without heuristic 4, one more X-Band job can indeed be scheduled, explaining why this schedule is cheaper in terms of accumulated penalty. It is however scheduled before the first L-Band job, rendering it useless because there is nothing to downlink. As expected, without heuristics 3, UPPAAL CORA predominately schedules UHF jobs parallel to X- or L-Band jobs, thereby straining the battery. The large number of states explored indicates that the state space exploration in this case is often misguided into eventual battery depletion.

Dynamic Scheduling. Another issue is that UPPAAL CORA's optimization criterion is static, i.e. the prices cannot be updated based on the schedule generated so far. This is contrasted by the GomSpace engineering intention of having a dynamic scheduling approach. The need for dynamic change of scheduling parameters during mission time was driven by initial uncertainty concerning the requirements to be faced and by the intention to try out operational limits, to experiment with the potential of GOMX–3 attitude control and to thereby gather experience for subsequent missions and future use case scenarios.

We take care of this by viewing the PTA network as being *parameterized*, i.e. as templates that need to be instantiated by concrete values. This enables us to divide the scheduling interval into disjoint subintervals that can be scheduled individually, with distinct scheduling objectives and prices, all the while carrying over resulting quantities as initial values to the subsequent subinterval to be scheduled. Important quantities that need to be passed on are the resulting battery state, the number of individual jobs already scheduled and the state of the PTA network at the end of the previous subinterval. This information allows us to adjust the prices and scheduling objective at the end of each subinterval, depending on the requirements previously fixed. The subschedules are then conjoined to a schedule for the actual time interval. This line of action is a trade-off between optimality and being dynamic, as it implements a greedy heuristics.

Given the back-to-back nature of this approach, it is undesired to start with an almost empty battery after a scheduling interval. We require the battery to have a certain minimum charge at the end of the schedule, greater than the Safe Mode threshold. This requirement translates directly to a reachability query on the PTA network: $\exists\Diamond(\texttt{Termination.Lock} \wedge \texttt{soc} \geq 75\,000\,000)$.

Schedule Validation

As mentioned, UPPAAL CORA's expressiveness does not allow for direct modelling of the KiBaM as a PTA. Instead the schedule computed is based on the simple linear model, that is known to not capture important effects that can be observed from measurements of real batteries. In order to validate whether the computed schedule truly doesn't violate the constraints we imposed, we need to validate the schedule along the stochastic KiBaM with capacity limits. In fact, such a schedule

can be seen as a sequence of tasks of finite length $(\Delta_j, \ell_j)_{j=0}^{N}$, which can immediately be used as input to one of the KiBaM algorithms described in Section 4. In this case the static discretization algorithm from Section 4.2 was used. The initial KiBaM SoC distribution is assumed to be a truncated 2D Gaussian around the initial battery state given to the PTA network and white noise is added to the loads of the tasks. If the validation step exhibits a low enough depletion risk, the computed schedule is accepted, otherwise the schedule is excluded and another schedule is computed. This happened only once. Across the schedules evaluated "low enough" was interpreted as "below 0.2" though much stricter guarantees were actually at hand in two of the three cases.

Implementation

The tool workflow has been implemented as follows. The operational windows for jobs are provided by GomSpace based on their inhouse tool POWERSIM as `csv` files. In addition to the interval to be scheduled, they serve as input to a `Python` script, which wraps the scheduling and validation parts. The `Python` script **(i)** preprocesses the `csv` tables provided by GomSpace **(ii)** instantiates the PTA templates with those numbers, **(iii)** calls UPPAAL CORA on the PTA instances, **(iv)** parses and postprocesses the optimal trace output by UPPAAL CORA to extract a schedule, **(v)** invokes a the static discretization algorithm (Section 4.2), to validate the schedule against a stochastic KiBaM, **(vi)** complements each operational window in the `csv` tables with the taken-or-skipped information, as dictated by UPPAAL CORA, and finally **(vii)** produces plots of the generated schedule as well as the resulting SoC distribution, including the depletion risk, for visual inspection.

The preprocessing step is mainly conversion of absolute date-time objects (i.e. `17 Nov 2015 00:38:38.922`) to UPPAAL CORA-compatible relative unix timestamps (0 being the starting point of the scheduling interval) represented as integers, as well as the selection of operational windows actually contained in the scheduling interval.

Postprocessing includes parsing of UPPAAL CORA's textual format of a trace, filtering it to remove irrelevant intermediate states and, finally, the conversion back to date-time objects to decide which operational windows were actually taken.

The `Scheduled` column of each `csv` file is filled in with 1 (the job opportunity was scheduled) or 0 (the job opportunity was skipped) accordingly. The `csv` tables, and optionally the plots, are subsequently sent to GomSpace, translated into GOMX–3-friendly instructions and uplinked via the UHF Aalborg connection.

The tool including the templates, the models, a few operational window tables and the necessary executables (except for UPPAAL CORA, which has to be acquired separately) can be accessed under

`https://www.powver.org/gomx3-supplementary-material/`.

The tool we make available is almost "push-button". It is preconfigured with meaningful parameter defaults for the GOMX–3 mission, so that passing start and end point of the scheduling interval should result in usable schedules that could be uplinked. No expert knowledge of UPPAAL CORA is required, as its model checking related arguments are fixed and passed by the `Python` wrapper. The arguments passed to the wrapper are mostly instantiations of place holders in the PTA template, and thus reflect requirements concerning the schedule, but not the model-checking routine. Using the default parameters of the tool, a schedule of 24 hours

consisting of a total of 6 UHF, 10 X-Band and 26 L-Band operational windows was synthesized on a laptop notebook. The modelchecker UPPAAL CORA needed 5.374 seconds to explore 349 191 states in order to find the optimal trace, inducing a task sequence of length 76. The StoKiBaM validation needed 4.903 seconds to compute the SoC density along this sequence. The runtime and space requirements of UPPAAL CORA grow exponentially with the scheduling horizon, while the StoKiBaM validation runtime and memory increase is linear in the task sequence length but quadratic in the grid size.

Empirical Results

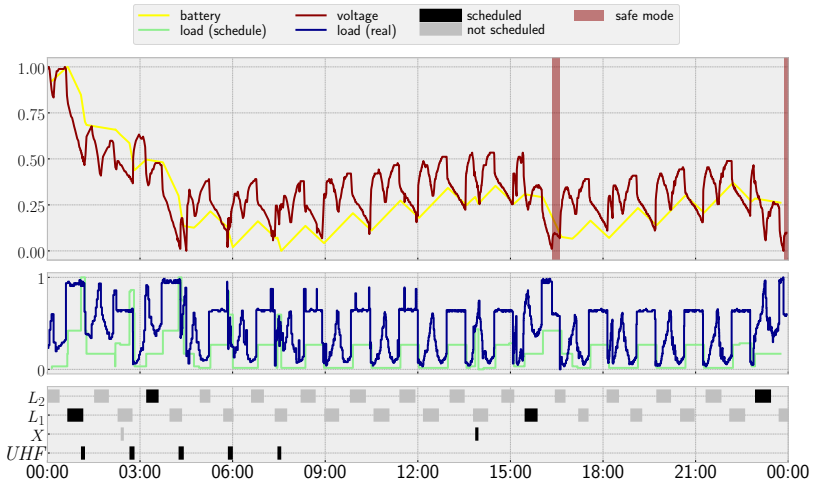
A number of successful experiments have been carried out on GOMX–3 in-orbit, so as to evaluate and refine our method, focussing on the determination of schedules to be followed for the days ahead. These in-orbit evaluations have successfully demonstrated the principal feasibility and adequacy of the approach, as we will discuss in this section.

In Figures 5.14–5.16 three representative in-orbit experiments are summarized. The schedules are visualized as three stacked plots of data against a common time line (top). The bottom ones are Gantt charts showing which jobs are scheduled (black bars) and which job windows are skipped (grey bars) respectively. The plots in the middle display the loads imposed by the jobs as predicted (light green) and as actually measured (dark blue) on GOMX–3. The top plots presents the battery SoC of the linear battery (yellow) as predicted by UPPAAL CORA as well as the actual voltage (dark red) logged by GOMX–3. Voltage and SoC are generally not comparable. However, both quantities exhibit similar tendencies during the charging/discharging process. The battery, voltage and load curves have all been normalized to the interval $[0,1]$ for comparison reasons.

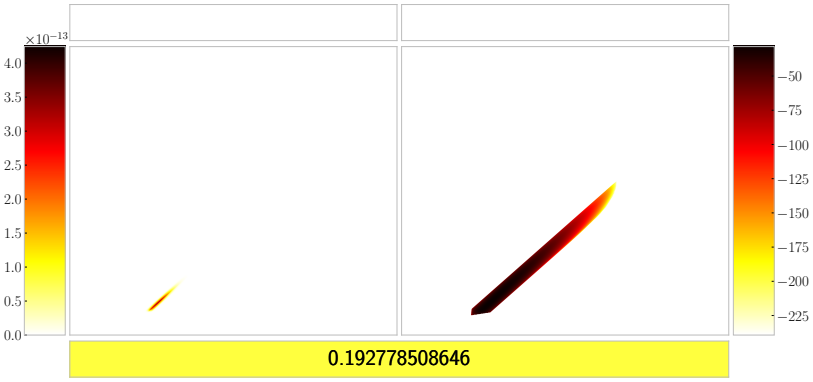
On the bottom, the three components of the SoC density resulting from the validation step are displayed, obtained by running the generated schedule along the stochastic KiBaM with capacity limits. It is to be interpreted as in Example 10. The most crucial part is at the bottom part of the plot, quantifying the risk of entering Safe Mode as specified by the GomSpace engineers (40 %). The data is summarized in Table 5.6.

November 2015. The schedule presented in Figure 5.14 spans November 17, 2015. It is a schedule that optimizes for maximum L-Band payload operations, yielding 4 L-Band operations and 1 X-Band operation together with the 5 UHF ground station passes. The battery SoC and the measured battery voltage show a close correspondence. GomSpace reported that GOMX–3 entered Safe Mode twice, if only for a short period of time. It later surfaced that an improper SoC-to-voltage conversion was used to determine the Safe Mode threshold.

February 2016. Figure 5.15 presents a schedule spanning one and a half day, starting on February 14, 2016. Before this experiment, GomSpace engineers gave notice that L-Band jobs shall receive special treatment from now on **(i)** Generally, the hardware modules related to L-Band jobs shall remain active for one whole orbit duration, once such a job is scheduled **(ii)** the ACDS needs 30 minutes of warm-up time to prepare for the L-Band job. This instance illustrates how optimized scheduling can be utilized to not only take power limitations into consideration



(a) Visualization of the synthesized/executed schedule.



(b) The final SoC distribution on a linear (left) and a logarithmic (right) color scale.

Figure 5.14: Schedule November 17, 2015 midnight to November 18, 2015 midnight.

but also handle secondary constraints like data generation and data downlinking balance via L-Band and X-Band tasks. The initial SoC and the internal depletion threshold were communicated to us as 90 % and 55 %, as the SoC-to-voltage conversion was corrected. The plot exhibits a drift between battery SoC and measured voltage around 3 PM of the first day, after initially showing a close correspondence, indicating that the battery is in a better state relative to our pessimistic predictions. The GomSpace engineers were able to track down this drift to a mismatch in the initial net power balance estimate provided by POWERSIM which are used as input to the toolchain, together with discrepancies in power draw for certain third-party modules. Deviations between power consumption values reported in data sheets and actual power draws were detected and sorted out.

March 2016. The third schedule we present (Figure 5.16) is the longest in duration, spanning from March 20 at 7 AM to March 22 at 7 PM. We were notified

Synthesis Setup (UPPAAL)				Analysis (UPPAAL)	Analysis (stoKiBaM)	Execution (GOMX–3)
Start (*dd.mm.yy hh:mm*)	Length (h)	init. SoC (%)	depl	min. SoC (%)	$\mathfrak{z}^{\uparrow}$	Safe Mode entered
17.11.15 00:00	24	85	0.4	40.3	≤ 0.193	2
14.02.16 00:00	36	90	0.55	69	$\leq 10^{-56}$	0
20.03.16 07:00	60	90	0.55	55.9	$\leq 10^{-4}$	0

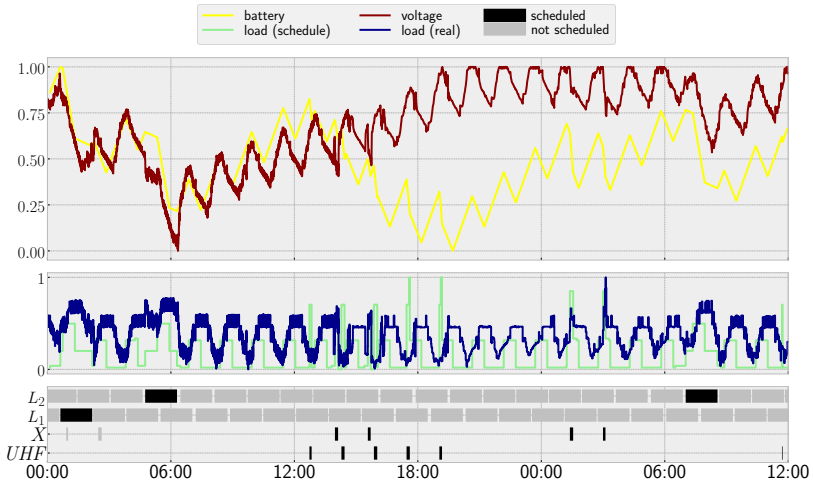
Table 5.6: A summary of the test runs performed on GOMX–3 on three different occasions. It reports on the value chosen as internal depletion threshold to the `battery` automaton, the initial SoC provided to UPPAAL CORA, the minimal SoC along the schedule generated by UPPAAL CORA, the depletion risk as calculated by the stochastic KiBaM validation step and how often GOMX–3 actually entered Safe Mode during schedule execution.

that GOMX–3 was supposed to test a simple, mission time prolonging adjustment: After each job, GOMX–3 should return to the so-called *nominal attitude* to minimize drag that would slowly decelerate the satellite, thereby shortening its time until atmospheric reentry. Thus, each scheduled job opportunity that requires a non-nominal attitude automatically had a 10 minutes ADCS cool-down phase appended. The energy consumption of cool-down and warm-up phases agree. After initial close correspondence of SoC and voltage, around 18 hours into the test run we observe a slight but continuous drift between predicted battery SoC and measured voltage, yet not as steep as in the February test run.

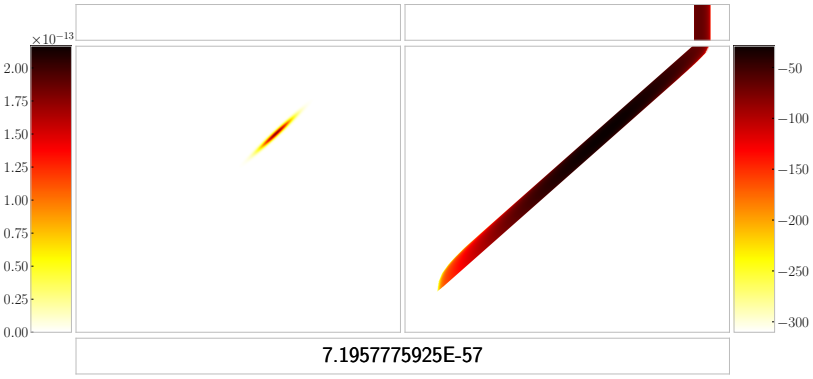
5.3 Receding-Horizon Scheduling using Model Predictive Control

Considering the results from Section 5.2, it becomes apparent that the predicted SoC and the actual voltage behavior drift apart further and further as time progresses. The SoC curve indicates a downwards tendency over time, in contrast to the voltage curve, that appears to recover to nominal voltage again and again. Figure 5.17 tracks the effects of the sequence of actually measured (not of predicted) satellite loads on the KiBaM SoC, a technique also known as *Coulomb Counting*, *Ampere Hour Counting* or *Current Integration Method* [31]. It shows a similar pattern, the SoC is driven closer and closer to the Safe Mode threshold, while the voltage indicates that the battery does not. The choice of the initial SoC is crucial, as Coulomb counting is not able to rectify an optimistic nor a pessimistic initialization over the course of time. The latter is indeed observable in Figure 5.17.

Voltage and SoC are quantities that are clearly related, yet it is unclear what the exact relation is. Despite this fact, widespread consensus is that voltage and SoC are somehow proportional in tendency in the following sense: If the measured



(a) Visualization of the synthesized/executed schedule.



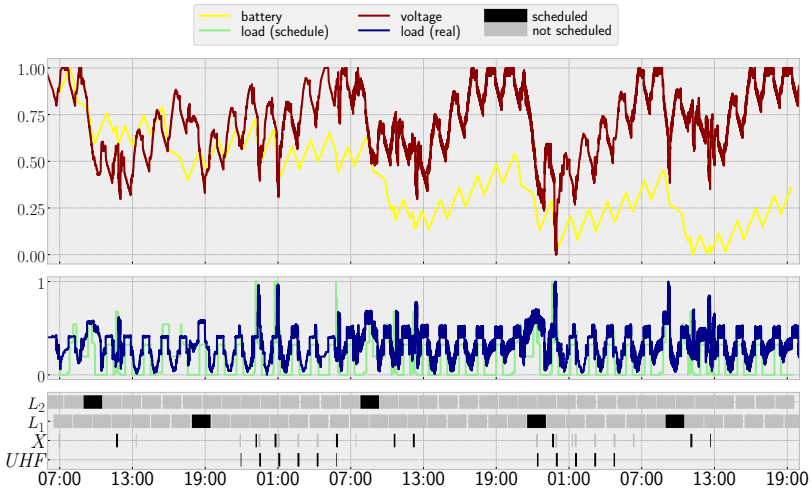
(b) The final SoC distribution on a linear (left) and a logarithmic (right) color scale.

Figure 5.15: Schedule February 14, 2016 midnight to February 15, 2016 noon

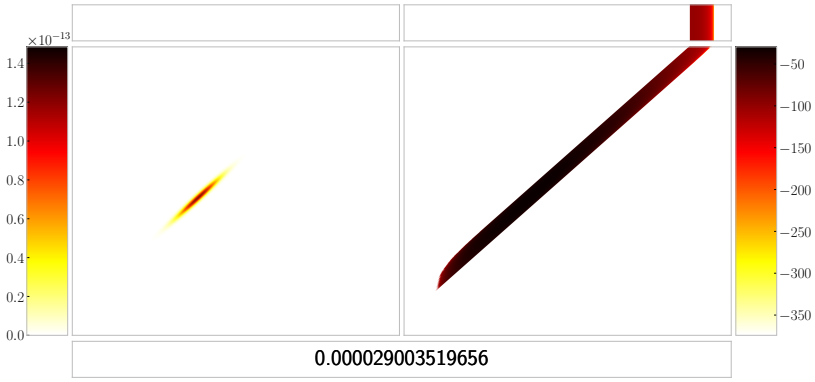
voltage is close to nominal voltage of a battery cell, it seems very unlikely that the SoC is close to depletion, or put differently, time series of measured voltage values carry information about the state of charge of a battery. Naturally the idea arises to incorporate measured voltage data from the satellite into the scheduling process as corrective measure to the SoC estimation. This paradigm is well known in the field of *Model Predictive Control (MPC)*.

Periodic Receding-Horizon Scheduling

Our goal is thus to incorporate logged (current and) voltage data from GOMX–3 into the scheduling mechanism as soon as they become available. At the same time, we want to perpetuate the thus far time-bounded scheduling approach, on the basis of the data we see. To explain our approach, let us for the moment assume for simplicity that GOMX–3 passes over the base station each orbit and is able to downlink its logged data in passing. We furthermore assume that orbits begin



(a) Visualization of the synthesized/executed schedule.



(b) The final SoC distribution on a linear (left) and a logarithmic (right) color scale.

Figure 5.16: Schedule March 20, 2016 7 AM to March 22, 2016 7 PM

when passing the base station, that GOMX–3 is currently at the beginning of its i-th overall orbit, and that it already has schedules $\mathscr{S}_i, \ldots, \mathscr{S}_{i+x}$ to follow, altogether spanning $x+1$ orbits. We refer to the overall schedule as $\mathscr{S}_i^{i+x}$. Hence it has just finished downlinking the voltage $\mathscr{U}_{i-1}$ and current data $\mathscr{I}_{i-1}$ from the $(i-1)$-st orbit to the base station. The following will be repeated in parallel.

Satellite: The satellite executes the first orbit as scheduled by $\mathscr{S}_i$ while measuring voltage $\mathscr{U}_i$ and current $\mathscr{I}_i$. Once it ends the orbit by contacting the base station, it downlinks its measured data and it receives a schedule $\mathscr{S}_{i+1}^{i+1+x}$ for the next $x+1$ orbits.

Base Station: Using $\mathscr{U}_{i-1}$ and $\mathscr{I}_{i-1}$, an estimate of the initial state of charge SoC_i of the i-th orbit is computed. As the satellite will behave according to $\mathscr{S}_i$, we can predict the initial state of charge SoC_{i+1} of the $(i+1)$-st orbit as well (by propagating the SoC_i along the predicted schedule loads). Based on this

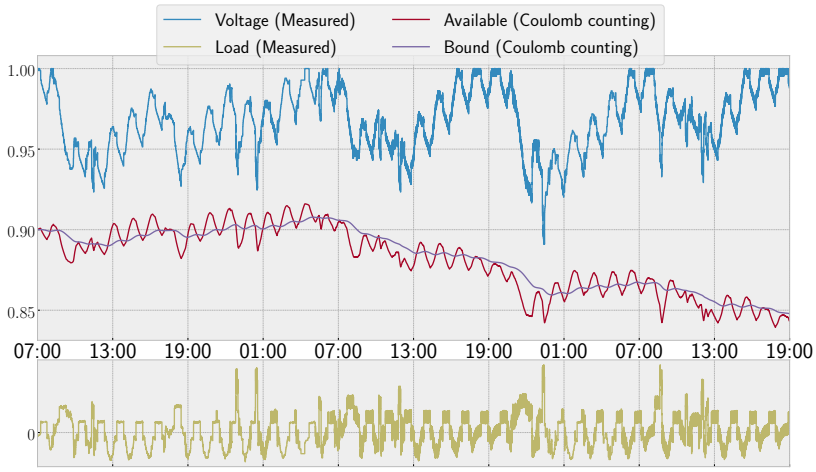


Figure 5.17: Given the current and voltage measurements of the March 20 experiment, we perform simple Coulomb counting using the KiBaM ($c = 0.2$, $p = 0.0001$) with initial SoC of $[a_0; b_0] = [90\,\%; 90\,\%]$ and compare to the measured voltage.

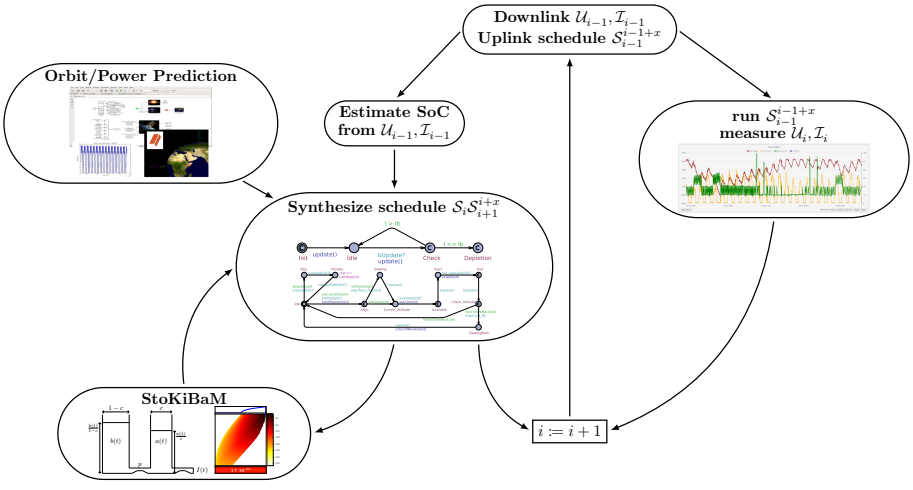


Figure 5.18: Receding-Horizon Scheduling Workflow.

estimate a new schedule $\mathscr{S}_{i+1}^{i+1+x}$ for orbits $i+1$ to $i+1+x$ is computed. If there is a job in $\mathscr{S}_i$ scheduled to run across the boundary of orbit i and $i+1$, schedule $\mathscr{S}_{i+1}$ needs to take this into account, by basically finishing what was started. Once the satellite ends the i-th orbit, the base station uplinks the computed schedule $\mathscr{S}_{i+1}^{i+1+x}$ and receives $\mathscr{U}_i$, $\mathscr{I}_i$.

This method thus creates overlapping schedules for the next $x+1$ orbits each. Each time the satellite passes the ground station, measured data is used to more accurately estimate the battery SoCs. Hence, it can be expected that SoC estimation and actually measured voltage data do not diverge in the long run. As before, each schedule is (apart from the logged data) based on the orbit predictions delivered by the GomSpace engineers and needs to pass the additional validation step using the stochastic KiBaM, just as before. This workflow is depicted in Figure 5.18.

Aperiodic Receding-Horizon Scheduling

In reality, the satellite is unable to contact the base station every orbit, thus x must be chosen large enough to guarantee operation, even if the link to the base station does not materialize for some time. So, let us assume that GomX–3 is unable to connect to the base station for n orbits, where $n < x$, but establishes connection at the end of its $(i+n)$-th orbit. In this case, it will behave as indicated by $\mathscr{S}_i^{i+n+1}$, and downlink $\mathscr{U}_i^{i+n}$ as well as $\mathscr{I}_i^{i+n}$ upon contact with the base station. The base station, in the meantime, propagates SoC_i through the next n orbits using the predicted schedule loads, and uses this as initial SoC to compute the next schedule from. This schedule then covers the next x orbits starting at orbit $i+n+1$.

Projecting KiBaM Onto The Linear Battery Model

In order to make use of the estimated KiBaM SoC for schedule synthesis, we have to convert it to a SoC of a linear battery model. It is not clear how to design such a mapping in a safe way. Nevertheless, several ideas come to mind.

Let $[a;b]$ be a KiBaM SoC with capacity limits $[\overline{a};\overline{b}]$ with given $\mathsf{c} \in]0,1[$ and $\mathsf{p} \in \mathbb{R}_{>0}$. We convert the SoC to a one-dimensional SoC of a linear model with capacity limit $\mathsf{cap} \coloneqq \overline{a}+\overline{b}$ by **(i)** $\frac{a}{\mathsf{c}}+\frac{b}{1-\mathsf{c}}$ or **(ii)** $a/\overline{a}\cdot\mathsf{cap}$, meaning by either aggregating both dimensions of the KiBaM SoC in an appropriate way, or by considering only the scaled up level of the available charge. In both suggestions, the linear model is neither an over- nor under-approximation of the KiBaM.

Using GomX–3 Data

In Section 3.7 we have already discussed the KiBaM filter and conducted a short proof of concept using synthesized, noisy available charge data. We now consider real measurements, and show that the accuracy and robustness against wrong initialization transfer well from the synthesized ideal setting. The main differences in this setting are twofold; **(i)** The abstract quantity of battery load is instantiated by actually measured current values, simply by identification, and **(ii)** the available charge of the battery can not be measured, and hence we need to rely on other measurable quantities of batteries. One such quantity is the internal voltage. In this regard, there is however no known relation between battery voltage and its state of charge. However, in related work SoC prediction schemes use learning

techniques to fit a relation between a one-dimensional battery SoC and measured voltage data [15].

In this section, we argue that measured voltage data and the available charge state of KiBaM are roughly proportional if considered over time. If the voltage samples exhibit a near nominal level over an extended period of time, it is highly unlikely that the SoC of a battery is near depletion. To the contrary, it is likely that the SoC (or at least its available charge part) is near its capacity limit as well. The voltage is however a more volatile quantity than any of the two KiBaM quantities we are interested in estimating. This becomes apparent when the sign of the load we put on a battery switches from positive to negative or vice-versa, i.e. from charging to discharging or the other way around. In these cases heavy voltage drops or peaks arise, that are presumably rooted in internal battery resistance phenomena [19]. By and large, however, the voltage recovers fast after experiencing such a drop/jump and follows a more regular trace. Apart from such short transient effects and apart from jitter, a time series of voltage measurements appears to contain noisy information about the KiBaMs available charge over time, a setting that suits a Kalman filter well, as we have seen already in Section 3.7.

Thus, in line with the receding-horizon scheduling approach, we intend to study voltage measurements obtained in-orbit as a basis for estimating the KiBaM available charge, and in transition the entire KiBaM SoC. GOMX–3 flies a battery pack with a nominal capacity of $\mathsf{cap} = 149\,760\,000\,\mathrm{mJ}$ and nominal voltage of 16 400 mV. As underlying dynamical model we chose a KiBaM with $\mathsf{c} = 0.2$, $\mathsf{p} = 0.0001$ and a time unit $\Delta t = 1$, hence $\overline{a} = 0.2\,\mathsf{cap}$. We assume that the voltage measurements are mixed with white noise of standard deviation $\sigma = 200$ (since the maximal measured voltage was just below 16 600 mV), so $R_k \coloneqq \mathsf{diag}\big(\sigma^2, 0\big)$. We map state into the measurement space using $H_k \coloneqq \mathsf{diag}(16\,600/\overline{a}, 0)$. The mapping of control variables (the measured current) onto the model state is given by the control matrix $B_k \coloneqq \mathsf{diag}(\Delta t, 0) = \mathsf{diag}(1, 0)$. The voltage and current data measured by GOMX–3 are not spaced equidistantly in the time domain, yet are placed at multiples of unit time $\Delta t = 1$. If there is no measurement available for a time point t we simply adopt the measured value from the previous time step.

In Figure 5.19 we depict the performance of a KiBaM filter estimating the SoC from measured voltage and current data compared to simple Coulomb Counting. We observe that a KiBaM Filter can indeed correct a pessimistic SoC initialization over the course of time, if the measured data suggest so. At the end of the experiment, the final SoCs produced via KiBaM filtering are 11 % higher than the one obtained by Coulomb Counting.

Schedule Improvements

Of course, it is most interesting to see to which extent the better estimates lead to schedules that exploit the satellites resources in a more cost-effective manner.

Unfortunately, by the time this scheduling paradigm was developed no satellite was in orbit on which we could perform such a study. However, we can rudimentarily demonstrate the potential of the improved scheduling approach on recorded GOMX–3 data, i.e. the data of the March 20 experiment. We can only use the data early in this experiment, because once we decide to deviate from the present schedule (to harvest the better estimates for improved cost-effectivity), we no longer have matching measurement data in the logs. This is unfortunate,

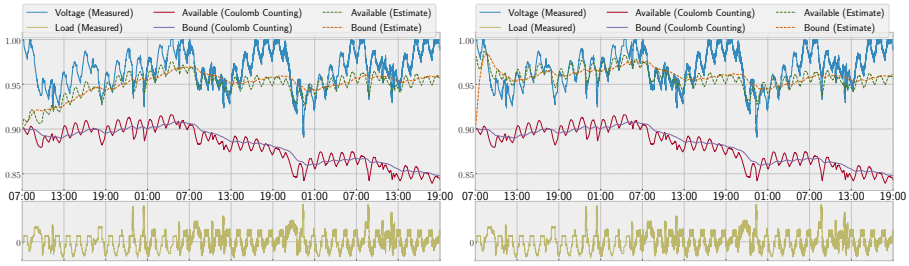


Figure 5.19: Given the current and voltage measurements of the March 20 experiment, we compare the SoC estimates produced by Coulomb Counting and a KiBaM filter. For both methods, we assume an initial SoC of $0.9[\overline{a};\overline{b}]$, yet the voltage suggests a nearly fully charged battery. **Left:** We assume an initial state variance of $P_0 \coloneqq \mathsf{diag}(\overline{a},\overline{b})$. **Right:** We assume an initial state variance of $P_0 \coloneqq 100 \cdot \mathsf{diag}(\overline{a},\overline{b})$.

because the early data will be affected overproportionally by the transient phase in which the filter calibrates itself to the circumstances.

Nevertheless, assume that GOMX–3 behaves just as the schedule (in Figure 5.16) indicates until it executes the 5-th UHF job (the second-to-last of the first batch, where SoC-voltage drift has already set in) at around 21.03.2016 05:48 UTC. We assume that the satellite succeeded in downlinking all the voltage and current logs to the base station. The KiBaM filter ($P_0 \coloneqq 10 \cdot \mathsf{diag}(\overline{a},\overline{b})$) returns a KiBaM SoC of $[0.9762; 0.9698] \cdot [\overline{a};\overline{b}]$. The SoC variable `soc` of the PTA network exhibits a SoC of 119 783 040 mJ at the same time, which is a SoC level of just below 75 %. After projecting the KiBaM SoC onto one dimension via one of the two methods sketched in Section 5.3, we end up with a SoC of 97.11 % and 97.62 % respectively, either of which constitutes a significant improvement. Propagating the KiBaM SoC using the predicted loads of the schedule until after the X-Band job that is scheduled simultaneously to the Aalborg connection, yields a SoC of $[0.9588; 0.9724] \cdot [\overline{a};\overline{b}]$ with projections to a linear model SoC of 96.97 % and 95.88 %, either of which lead to the same subsequent schedule.

Figure 5.20 visualizes how the schedule after this critical UHF job improves thanks to an updated subschedule that is computed on the basis of this new SoC estimate. We assume that in the 6-th UHF job GOMX–3 was able to receive the updated schedule and in the meantime it continued operation as dictated by the original schedule. Indeed, this single update makes it possible to increase the overall number of scheduled X-Bands jobs from 5 to 7 and of L-Band jobs from 3 to 4.

Related Work

Variants of scheduling problems for earth-observing satellites have been considered in the scientific literature, with a focus on maximizing the operational efficiency of an orbiting satellite. The problem was tackled by partial enumeration methods [16], dynamic programming (with heuristics and bounding procedures) [14], conversions into constraint satisfaction problems (enabling greedy search to find good solutions or Branch-and-Bound techniques to find optimal solutions) [1], and knapsack formalizations of the problem set (with tabu search

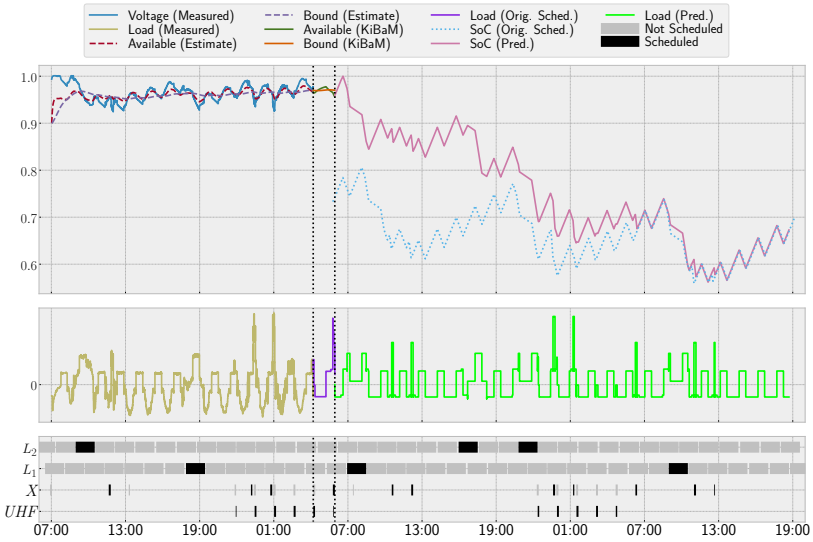


Figure 5.20: A schedule showing the potential of the KiBaM filter SoC estimation method. Left of the dotted vertical lines, the KiBaM filter ($c = 0.02$, $p = 0.0001$, $[a_0; b_0] = 0.9 \cdot [\overline{a}; \overline{b}]$) is applied on the measured data to estimate the SoC up until the 5-th Aalborg connection, with initial state variance of $P_0 := 10 \cdot \mathsf{diag}(\overline{a}, \overline{b})$. Between the dotted vertical lines, the KiBaM and the predicted loads of the original schedule are used to propagate the estimated SoC. All the computations needed take place in this period. The effect of the updated schedule is shown right of the dotted vertical lines. We use the propagated SoC as initial SoC (95.88 %) of our predictions using a linear battery model. The corresponding part of the original SoC trace from Figure 5.16 is presented as a blue dotted plot to enable visual comparison. The updated schedule lets GOMX–3 carry out two more X-Band jobs and one more L-Band job.

algorithms as means of finding feasible schedules) [41]. These works solely focus on maximizing payload throughput and do not consider dynamic power-relevant constraints.

A constraint-based technique using heuristic search with constraint propagation also focussing on renewable resources like battery power or memory has been studied as well [30], however with a linear energy storage model.

Recently, a statistical model checking approach to battery-aware scheduling has been proposed [42] that uses UPPAAL SMC and a hybrid automaton model to capture kinetic battery dynamics, together with a synthetic example inspired by the satellite domain.

The scheduling workflow exercised in this work is very close in spirit to the approach developed in [28], where a simulation-based analysis of computed schedules is used to validate or refute cost-optimal schedules, under a model with stochastic breakdowns and repairs of production machinery. Apart from the different application domain, the main conceptual distinction is that the validation step in our work is not based on simulation, but on the calculation of proper bounds of the quantities of interest.

Discussion And Conclusion

This section has presented a battery-aware scheduling approach for low-earth orbiting nanosatellites. The heterogeneous timing aspects and the experimental nature of this application domain pose great challenges, making it impossible to use traditional scheduling approaches for periodic tasks. Our approach harvests work on schedulability analysis with (priced) timed automata. It is distinguished by the following features: **(i)** The TA modelling approach is very flexible, adaptive to changing requirements, and particularly well-suited for discussion with space engineers, since it is easy to grasp. **(ii)** A dynamic approach to the use of cost decorations and constraints allows for a split scheduling approach optimizing over intervals, at the (acceptable) price of potential sub-optimality of the resulting overall schedules. **(iii)** A linear battery model is employed while scheduling, but prior to shipping, all computed schedules are subjected to a quantitative validation on the vastly more accurate Stochastic KiBaM, and possibly rejected.

The GOMX–3 in-orbit experiments have demonstrated a great fit between the technology developed and the needs of the LEO satellite sector. It became apparent that relative to a manual scheduling approach as otherwise employed by GomSpace, the presented method synthesizes better quality schedules with respect to **(i)** number of jobs performed, **(ii)** avoidance of planning mistakes, **(iii)** scheduling workload, and **(iv)** battery depletion risk provided. At the same time, the availability of scheduling tool support flexibilizes the satellite design process considerably, since it allows the GomSpace engineers to obtain answers to what-if questions, in combination with their in-house POWERSIM tool. This helps shortening development times and thus time-to-orbit. We have furthermore embedded this contribution into a perpetuated scheduling workflow that harvests in-orbit measurement to adjust and correct the KiBaM model in such a way that schedules are continuously based on the most recent and most precise information at hand.

Future work in this area is partly driven by the application domain. State of the art technology and very rapid development cycles will continue to be a crucial part of the nanosatellite market. They are the roots of a steady stream of novel scientific challenges. In fact, GomSpace has launched a 2 spacecraft constellation (GOMX–4 A and B) on February 1st 2018 and is actively pursuing several projects with much larger constellations. Deploying constellations of a large number of satellites (2 to 1000) brings a new level of complexity to the game, which in turn asks for a higher level of automation to be used than has previously been the case in the space industry. The technology investigated here is beneficial in terms of optimization and planning of satellite operations, so as to allow for more efficient utilization of spacecraft flight time. A spacecraft operator is faced with a highly complex task when having to plan and command in-orbit operations constantly balancing power and data budgets. For larger constellations tools for optimization, automation and validation are not only a benefit, but likely a necessity for proper operations.

The proposed battery-aware scheduling approaches can be improved in several dimensions, the first being to fully implement and integrate the receding-horizon approach into the present tool. Another venture consists of extending the single satellite scheduling approach to conjoined scheduling of satellite constellations. Alternative schedule synthesis formalisms come to mind for potential exploration, preferably formalisms that handle several cost variables at once, so as to optimize for more complex objective functions. Among the tools of interest

are **(i)** the MODEST toolset which supports the computation of reward-optimal time-bounded properties of Markov Decision Processes [13], **(ii)** newer versions of the UPPAAL toolset performing Pareto optimal reachability analyses on simple priced timed automata [43], or **(iii)** OPTIMATHSAT [36], an efficient and highly flexible OMT (Optimization Modulo Theories) solver from an emerging field currently getting a lot of attention. A major challenge is the direct integration of non-linear continuous dynamics into the scheduling step itself, in order to synthesize plans with respect to the kinetic battery model.

Most of these problems have been solved and have been integrated in an improved version of the scheduling workflow, that is now being used in the GOMX–4 mission [38]. Here, the scheduling synthesis step is based on Dynamic Programming, which allows for direct incorporation of KiBaM dynamics into the synthesis step.

Lifting state of the art hybrid systems tools, such as HYPRO [35] or SPACEEX [10] from verification to synthesis could be another way to achieve these goals. However, even efficient verification of linear hybrid systems is known to be a difficult problem set.

In addition, investigating and modelling temperature dependent performance properties of batteries as well as battery wear grows more and more important with longer mission times and should therefore not be neglected.

CHAPTER

6 Conclusion

6.1 Achievements

In this work we focussed on the kinetic battery model (KiBaM), an intuitive energy storage model that is particularly well suited to model the state of charge evolution of lithium-ion batteries along a sequence of tasks. It essentially splits the charge of a battery in an immediately available part, and a chemically bound part, that can be converted into each other over time, via a diffusion process.

The model was extended with the concept of depletion using a safe threshold as follows: If the available drops below the safe threshold, the device which is powered by the battery shuts down.

Another extension is that of capacity limits, a concept that is asymmetric to that of depletion, simply because a saturated battery continues to power a device. If the battery is saturated, its dynamics change instantly to a simpler ODE system, namely one where the available charge stays constant, which again can be solved in a straight-forward way. Several scenarios need to be considered if the load changes and the battery is saturated, i.e. it may leave its saturated state, it may stay saturated, or it may transiently become unsaturated, if the diffusion overpowers the load.

It is crucial to find the exact time point of saturation given a task, so as to know when to switch battery dynamics. Unfortunately, finding this time point involves solving the product logarithm problem, a problem whose solution can only be brought into closed form using the non-elementary Lambert-W function. Thus, the exact saturation time point is transcendental, and can therefore only be approximated. We provided such an approximation algorithm in the form of a bisection method. This algorithm enables to under- as well as over-approximate the actual battery dynamics in a safe way.

A second major and orthogonal extension to the KiBaM is that of uncertainties, firstly in the initial battery state, as well as in the load that is put on the battery. To this end, we introduced state of charge distributions, that are inherently discontinuous exactly at the depletion and saturation thresholds if capacity limits are considered as well. We derived analytical expressions to approximate the successor distribution of an initial SoC distribution after powering a possibly noisy task, mostly using the transformation law of random variables. These expressions are

of limited practical use, since integrals are stacked with every further task, making the expressions computationally inefficient.

In a further effort, we extended the load model from simple sequences of (noisy) tasks to Markovian load processes, called Markov Task Processes. These deterministic time processes add probabilistic jumps, thus enabling several noisy task successors with a certain probability, instead of only one as in sequences. We formally and rigorously described how to capture the entire behavior of load process up to a certain time horizon, and its effect on an initial Soc distribution.

In a short intermezzo, we discussed how to consolidate the abstract quantities of the KiBaM, i.e. SoC and load, with observable, measurable data like voltage and current. To this end, we discretized the continuous KiBaM ODEs and instantiated a Kalman filter with the resulting discrete difference equations, simply by identification. We conducted a short proof of concept study using synthetically generated data, and concluded that the resulting KiBaM filter is indeed able to satisfactorily estimate the internal state of charge of a KiBaM from noisy measurements, even if the initial SoC estimate is severely off.

Next, we tackled the computational inefficiency of the analytical way of tracking SoC distributions along sequences of tasks. In a first approach we followed a static discretization (SD) scheme, in which the entire safe SoC space as well as the tasks continuously distributed loads were gridded in a safe way, resulting in discretized SoC distributions and discretized (and possibly truncated) noisy tasks. We derived how to compute the successors of such a discretized SoC distribution given a discrete noisy task.

In an attempt to improve the efficiency of this scheme, we made it adaptive (AD), i.e. we focus the area of discretization only on the immediate rectangular neighborhood of the support of the SoC distribution, so as to eliminate a large number of empty cells in the discretization grid. These neighborhoods, called bounding boxes, were key to the efficiency of the successor computation of a SoC distribution: First the successor bounding box needed to be determined such that it stays as small as possible. To this end, the so called saturation and depletion boundaries were crucial, since they provided the exact points where the bounding box must potentially be split, because of the discontinuous nature of SoC distributions. Once the successor box was determined, a grid was put inside it, to finally proceed with the usual successor computation.

To gauge the efficiency gain, we ran both SD and AD on randomly generated task sequences with varying grid sizes until they both eventually reached the designated precision level, then compared runtimes and grid sizes. Unsurprisingly, the experiments almost unanimously saw AD as the clear winner.

In an attempt to move away from discretizing the SoC space, we developed the percentile propagation (PP) scheme. Its essence is rooted in the fact that, if a certain SoC depletes (survives) under a given task sequence, then every smaller (greater) SoC also depletes (survives), due to the monotonicity properties of the KiBaM. If we additionally know that the SoC under investigation is larger than q percent of the entire support of the initial SoC distribution, the depletion risk is at least q, should the SoC indeed deplete. Dually, if it survives we deduce that the depletion risk is at most q. In order to these so-called SoC percentiles to exist and be unique, we place a few mild restrictions on the family of viable initial SoC distributions: the *less than* relation must be a total order on the support of the initial distribution. Integrating the above step in a bisection-style scheme, we can eventually deduce an arbitrarily small interval around the true depletion risk.

We lifted this scheme to discretized noisy task sequences, by generating every possible task sequence induced by the noisy task sequence, and running the above procedure, then combining the results as a weighted sum.

In order to compare the algorithms efficiency we ran it against AD, again on randomly generated task sequences, until they reached a similar precision. We found that the efficiency of PP massively depends on the number of possible task sequences induced by the given noisy task sequence, and much less on the length of the task sequence, as AD does.

In the second part of this thesis, we focussed on the practical application aspects. We used the SD algorithm to analyse the energy budget of GOMX–1, a 2-unit CubeSat built by GomSpace, a Danish satellite operator, up to a time horizon of a year of operation. The findings confirmed what GomSpace already suspected, namely that the onboard batteries were highly over-dimensioned. Our analysis quantified this over-dimensioning with a factor of 8, meaning that the nanosatellite could have done its work with only one quarter of its actual battery capacity, thereby establishing that our analysis can be a valuable tool during the design phase of a satellite mission.

In a second collaboration with GomSpace, we bridged the gap from analysis to that of battery-aware schedule synthesis of GOMX–3, a 3-unit CubeSat. To this end, we modelled the power-relevant characteristics and modes of operation of GOMX–3 into a network of priced timed automata (PTA), so as to optimally choose jobs to be executed by the satellite, without depleting its batteries. Unfortunately, PTA are not expressive enough to capture the non-linear KiBaM, thus the simple linear battery model was embedded into an automaton instead. This implied that every schedule computed by UPPAAL CORA, a widely used modelchecker for PTA, is potentially unsafe. In order to filter out unsafe schedules, a synthesis-validation loop was introduced: Every synthesized schedule must by validated or refuted using the extended KiBaM. In the latter case, the schedule is excluded from the synthesis step, and a new schedule is computed. Once a schedule induces a low enough depletion risk, it can safely be uplinked to GOMX–3, where it is executed. In total, three test runs of up to 72 hours were conducted in conjunction with GomSpace. GOMX–3's telemetry data recorded during the test runs show that the scheduling indeed works well, yet a slight drift between reality and the scheduling pipeline was exposed.

In order to counter this drift, we extended the scheduling workflow to a receding-horizon approach. The outline is as follows: With each pass over the ground station a satellite receives a new schedule and downlinks its collected telemetry data, which in turn can be used in conjunction with the KiBaM filter, to estimate the actual SoC of the battery. A new schedule, starting from that moment onwards is then computed using the estimated SoC, thereby eliminating, or at least limiting any drift. The satellite will receive the new schedule with its next passing over the ground station, where it again downlinks its collected telemetry data, and so on. Unfortunately GOMX–3 had already deorbited by the time the receding-horizon scheduling pipeline was ready to be used. However, we at least provided a rudimentary proof of concept using old GOMX–3 telemetry data, which showcased the potential of this new approach.

6.2 Outlook

This work has established that formal methods in conjunction with formal enegry storage models can be of great value during the design phase as well as during in-flight operation of a nanosatellite mission. However this is only a stepping stone to where the actual trend of the industry points us to, namely large constellations of inter-communicating CubeSats. In fact, the receding-horizon battery-aware scheduling pipeline has already been improved and adopted, and is being actively used with the GOMX–4 mission [38], a two satellite train constellation. Here, the synthesis step is implemented as a dedicated dynamic program that is expressive enough to capture the non-linear KiBaM, and thus doesn't rely on a synthesis-validation loop, thereby showcasing that the methods presented in this thesis can be adopted for scheduling of small-size satellite constellations. We however observe a tendency towards larger constellations of conceivably thousands of satellites, which exposes a considerable scalability issue with the schemes presented in this work. A prime example of this is the *Starlink* project operated by *SpaceX*, a satellite network consisting of several thousand satellites designed to provide worldwide internet access [37]. Dedicated research has been undertaken, concerning a hypothetical ten satellite train constellation named *Ulloriaq* (the icelandic word for *Star*) derived from the GOMX–4 mission [9]. In this scenario, a linear programming scheme was used to synthesize battery-aware contact plans for such constellations. Further research in making battery-aware planning schemes scale to larger, more complex constellations is conceivable and ultimately necessary since manual control of large networks is becoming increasingly impractical.

The space domain is not the only sphere of interest for the methods presented in this thesis, simply because batteries have become omni-present. In the context of a politically backed emerging market for electric vehicles, energy storage models and dedicated routing, planning or analysis algorithms are vital to further increase the efficiency of battery-powered mobility. The challenges in this domain are, perhaps surprisingly, more numerous and a lot harder compared to the space domain, since uncertainty about energy usage increases greatly. In contrast to the almost complete determinacy of a satellite's orbit, and therefore its energy budget, factors like traffic density, cloudiness, driving style or faulty map data potentially massively increase the complexity of any battery-aware scheme on Earth. Adaptations of battery-aware scheduling schemes to problems of the e-mobility domain seem very worthwhile.

Another domain of interest is the *Internet of Things*, potentially large networks of sensor- and software-assisted physical objects inter-connected via the internet, many of which are battery powered. A better understanding and more accurate modelling of these objects' energy behavior and better, battery-aware communication protocols will undoubtedly have a positive impact on the efficiency of such networks. This is especially relevant in a context where more and more of the internet's connectivity stems from battery-powered satellite networks.

Bibliography

[1] JC Agn and E Bensana. “Exact and approximate methods for the daily management of an earth observation satellite”. In: (1995).

[2] Rajeev Alur and David L. Dill. “A Theory of Timed Automata”. In: *Theoretical Computer Science* 126 (1994), pp. 183–235.

[3] Gerd Behrmann, Kim G Larsen, and Jacob I Rasmussen. “Optimal scheduling using priced timed automata”. In: *ACM SIGMETRICS Performance Evaluation Review* 32.4 (2005), pp. 34–40.

[4] Gerd Behrmann et al. “Minimum-Cost Reachability for Priced Timed Automata”. In: *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*. Ed. by Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli. Vol. 2034. Lecture Notes in Computer Science. Springer, 2001, pp. 147–161. DOI: 10.1007/3-540-45351-2_15. URL: https://doi.org/10.1007/3-540-45351-2_15.

[5] J. Cao, N. Schofield, and A. Emadi. “Battery balancing methods: A comprehensive review”. In: *Vehicle Power and Propulsion Conference, 2008. VPPC '08. IEEE*. Sept. 2008, pp. 1–6. DOI: 10.1109/VPPC.2008.4677669.

[6] UPPAAL CORA. URL: https://uppaal.org/features/#cora (visited on 09/02/2021).

[7] Robert M Corless et al. “On the LambertW function”. In: *Advances in Computational mathematics* 5.1 (1996), pp. 329–359.

[8] *Fortran programs for the simulation of electrochemical systems*. http://www.cchem.berkeley.edu/jsngrp/fortran.html. May 2020. URL: http://www.cchem.berkeley.edu/jsngrp/fortran.html.

[9] Juan A. Fraire et al. “Battery-Aware Contact Plan Design for LEO Satellite Constellations: The Ulloriaq Case Study”. In: *TGCN* 4.1 (2020), pp. 236–245. DOI: 10.1109/TGCN.2019.2954166. URL: https://doi.org/10.1109/TGCN.2019.2954166.

[10] Goran Frehse et al. “SpaceEx: Scalable Verification of Hybrid Systems”. In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 379–395. DOI: 10.1007/978-3-642-22110-1_30. URL: https://doi.org/10.1007/978-3-642-22110-1_30.

[11] Pascal Gilles. “Private Communication”. 2014.

[12] GomSpace. *GomSpace GOMX-1*. https://gomspace.com/gomx-1.aspx. May 2020. URL: https://gomspace.com/gomx-1.aspx.

[13] Ernst Moritz Hahn and Arnd Hartmanns. “A Comparison of Time- and Reward-Bounded Probabilistic Model Checking Techniques”. In: *Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings*. Ed. by Martin Fränzle, Deepak Kapur, and Naijun Zhan. Vol. 9984. Lecture Notes in Computer Science. 2016, pp. 85–100. DOI: 10.1007/978-3-319-47677-3_6. URL: https://doi.org/10.1007/978-3-319-47677-3_6.

[14] Nicholas G Hall and Michael J Magazine. “Maximizing the value of a space mission”. In: *European journal of operational research* 78.2 (1994), pp. 224–241.

[15] Terry Hansen and Chia-Jiu Wang. “Support vector based battery state of charge estimator”. In: *Journal of Power Sources* 141.2 (2005), pp. 351–358.

[16] SA Harrison, ME Price, and MS Philpott. “Task scheduling for satellite based imagery”. In: *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group*. Vol. 78. Uni2 versity of Sanford, UK. 1999, pp. 64–78.

[17] Hongwen He et al. “State-of-charge estimation of the lithium-ion battery using an adaptive extended Kalman filter based on an improved Thevenin model”. In: *IEEE Transactions on Vehicular Technology* 60.4 (2011), pp. 1461–1469.

[18] Chao Hu, Byeng D Youn, and Jaesik Chung. “A multiscale framework with extended Kalman filter for lithium-ion battery SOC and capacity estimation”. In: *Applied Energy* 92 (2012), pp. 694–704.

[19] Texas Instruments Incorporated. *TMS570 Active Cell-Balancing Battery-Management Design Guide*. 2016.

[20] Marijn R. Jongerden and Boudewijn R. Haverkort. “Which battery model to use?” In: *IET Software* 3.6 (2009), pp. 445–457. DOI: 10.1049/iet-sen.2009.0001. URL: http://dx.doi.org/10.1049/iet-sen.2009.0001.

[21] Marijn Remco Jongerden. “Model-based energy analysis of battery powered systems”. PhD thesis. Enschede, Dec. 2010. URL: http://doc.utwente.nl/75079/.

[22] Rudolph Emil Kalman et al. “A new approach to linear filtering and prediction problems”. In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[23] Kim Larsen et al. “As Cheap as Possible: Effcient Cost-Optimal Reachability for Priced Timed Automata”. In: *Computer Aided Verification*. Springer. 2001, pp. 493–505.

[24] Kim G Larsen, Paul Pettersson, and Wang Yi. “UPPAAL in a nutshell”. In: *International Journal on Software Tools for Technology Transfer (STTT)* 1.1 (1997), pp. 134–152.

[25] Edward A Lee and David G Messerschmitt. “Synchronous data flow”. In: *Proceedings of the IEEE* 75.9 (1987), pp. 1235–1245.

[26] Jaemoon Lee, Oanyong Nam, and BH Cho. “Li-ion battery SOC estimation method based on the reduced order extended Kalman filtering”. In: *Journal of Power Sources* 174.1 (2007), pp. 9–15.

[27] Bor Yann Liaw et al. “Correlation of Arrhenius behaviors in power and capacity fades with cell impedance and heat generation in cylindrical lithium-ion cells”. In: *Journal of power sources* 119 (2003), pp. 874–886.

[28] Angelika Mader et al. “Synthesis and stochastic assessment of cost-optimal schedules”. In: *International journal on software tools for technology transfer* 12.5 (2010), pp. 305–318.

[29] James F. Manwell and Jon G. McGowan. “Lead acid battery storage model for hybrid energy systems”. In: *Solar Energy* 50.5 (1993), pp. 399–405.

[30] Joseph C Pemberton and Flavius Galiber. “A constraint-based approach to satellite scheduling”. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 57 (2001), pp. 101–114.

[31] Sabine Piller, Marion Perrin, and Andreas Jossen. “Methods for state-of-charge determination and their applications”. In: *Journal of power sources* 96.1 (2001), pp. 113–120.

[32] Gregory L Plett. “Kalman-filter SOC estimation for LiPB HEV cells”. In: *Proceedings of the 19th International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exhibition (EVS19). Busan, Korea*. 2002, pp. 527–538.

[33] *SENSATION Project (snapshot on the wayback machine)*. URL: `https://web.archive.org/web/20171211083415/http://sensation-project.eu/` (visited on 12/11/2017).

[34] Robin A Sahner and Kishor S. Trivedi. “Performance and reliability analysis using directed acyclic graphs”. In: *IEEE Transactions on Software Engineering* 13.10 (1987), pp. 1105–1114.

[35] Stefan Schupp et al. “HyPro: A C++ Library of State Set Representations for Hybrid Systems Reachability Analysis”. In: *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*. Ed. by Clark Barrett, Misty Davies, and Temesghen Kahsai. Vol. 10227. Lecture Notes in Computer Science. 2017, pp. 288–294. DOI: 10.1007/978-3-319-57288-8_20. URL: `https://doi.org/10.1007/978-3-319-57288-8_20`.

[36] Roberto Sebastiani and Patrick Trentin. "OptiMathSAT: A Tool for Optimization Modulo Theories". In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 447–454. DOI: 10.1007/978-3-319-21690-4_27. URL: https://doi.org/10.1007/978-3-319-21690-4_27.

[37] SpaceX. *Starlink*. URL: https://www.starlink.com (visited on 09/02/2021).

[38] Gregory Stock et al. "Managing Fleets of LEO Satellites: Nonlinear, Optimal, Efficient, Scalable, Usable, and Robust". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.11 (2020), pp. 3762–3773. DOI: 10.1109/TCAD.2020.3012751. URL: https://doi.org/10.1109/TCAD.2020.3012751.

[39] Bart D Theelen et al. "A scenario-aware data flow model for combined long-run average and worst-case performance analysis". In: *Formal Methods and Models for Co-Design, 2006. MEMOCODE'06. Proceedings. Fourth ACM and IEEE International Conference on*. IEEE. 2006, pp. 185–194.

[40] BD Theelen et al. *Scenario-aware dataflow*. Tech. rep. Citeseer, 2008.

[41] Michel Vasquez and Jin-Kao Hao. "A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite". In: *Computational Optimization and Applications* 20.2 (2001), pp. 137–157.

[42] Erik Ramsgaard Wognsen, René Rydhof Hansen, and Kim Guldstrand Larsen. "Battery-aware scheduling of mixed criticality systems". In: *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*. Springer Berlin Heidelberg, 2014, pp. 208–222.

[43] Zhengkui Zhang et al. "Pareto Optimal Reachability Analysis for Simple Priced Timed Automata". In: *Formal Methods and Software Engineering - 19th International Conference on Formal Engineering Methods, ICFEM 2017, Xi'an, China, November 13-17, 2017, Proceedings*. Ed. by Zhenhua Duan and Luke Ong. Vol. 10610. Lecture Notes in Computer Science. Springer, 2017, pp. 481–495. DOI: 10.1007/978-3-319-68690-5_29. URL: https://doi.org/10.1007/978-3-319-68690-5_29.