

Battery Modelling

An MMSC Case Study on MATHEMATICAL MODELLING

Candidate Number: 1072462

Abstract

This work will attempt to

Contents

1	Introduction	2
2	Problem Formulation	3
2.1	The Isolated Battery	3
2.2	The Equivalent Circuit Model	3
2.3	Battery in an Electric Vehicle (EV)	4
2.4	A Variational Optimisation Problem	5
2.5	Battery Aging	6
3	Numerical Simulation of a Car	7
3.1	Finding ECM Parameters	7
3.2	Forward Euler Simulation	8
4	Metropolis-Hastings and A-Star	8
4.1	Shortest Path Finding	8
4.2	Monte-Carlo Optimisation	8
4.3	Special Case: Granny's House	9
4.4	Routing in Jericho	9
5	Conclusion	9
A	Simulation Code	11

1 Introduction

Clearly, electric batteries are largely important for various industries today and demand for them is ever-growing. This includes, especially, the renewable energy sector due to the unpredictability of energy supplies such as wind and solar power where short-term storage is a necessary evil. Similar relevance may be found in the car industry where one aims for highly (space-)efficient mobile storage of energy. In many countries and/or regions, Electric Vehicles (EVs) still lack a well-enlarged network of charging stations, for various reasons including incompatibilities between charging station suppliers. Using methods introduced in the present report, we aim to optimise the customer experience of an electric vehicle owner through a routing application that takes electric battery peculiarities and battery life-time into account.

In this report, we will consider how to model a Lithium-Ion battery, more specifically the Panasonic 18650PF for which there is some characteristic data available in the public domain ([Kollmeyer 2018](#)). To do this, we will consider an Equivalent Circuit Model (ECM), namely the Thevenin ECM, which may be found in Figure 2. The key idea here is to model the voltage output $V(t)$ given a current profile $I(t)$. This model not only depends on current and time, but also keeps track of internal quantities like the state of charge (SOC, $s \in [0, 1]$) and state of health (SOH, $h \in [0, 1]$). It can be reduced to a system of ordinary differential equations, for which, using given profiles, we find parameters R_0 , R_1 , C_1 and more by a fitting procedure. Later on, we also considered aging effects of the battery through usage over time, which resulted in an additional differential equation in the ODE system.

Based on the model we obtained, verified and validated on new pulse test data, the next step was to apply the model to a real-world application, namely that of electric vehicle routing. For this purpose, we obtain graph data of the road network of choice (users may enter new localities as a text input) from OpenStreetMap ([OpenStreetMap contributors 2023](#)). In order to find the best route from A to B, the algorithm then performs a classical routing algorithm called *A-Star* to find the (geometrically) shortest path between A and B, which may not yet be a feasible or optimal path to the destination but a good start. A good initial condition is all that the remaining algorithm requires, which is a Monte-Carlo iterative method that repeatedly applies small, well-defined, perturbations to the graph route and then obtains a time or cost estimate of the modified trip using the battery (and car) model we introduced. If the

modified route yields an improvement in the metric of choice, it is accepted as the new state and the process repeats until a satisfactory route was found.

Further

2 Problem Formulation

2.1 The Isolated Battery

Let $s \in [0, 1]$ denote the *state of charge* (SOC) of the battery, $h \in [0, 1]$ the *state of health* (SOH), $Q \in \mathbb{R}^+$ the charge, $Q_{00} \in \mathbb{R}^+$ the maximum possible charge at the time of production (in Coulombs), $V \in \mathbb{R}$ the voltage across the battery (in Volts) with $I \in \mathbb{R}$ the corresponding current (in Amperes) where $I > 0$ corresponds to discharging the battery. Then, per common definition, $s := \frac{Q}{Q_0}$ is the amount of charge currently present in the battery as compared to $Q_0 \in \mathbb{R}^+$ the current maximum capacity, which itself is dependent on the state of health, as given by $Q_0 := hQ_{00}$. Further let $T \in [-273.15, \infty)$ denote the temperature of the battery (in degrees Celsius) and let $t \in \mathbb{R}$ represent time (in seconds).

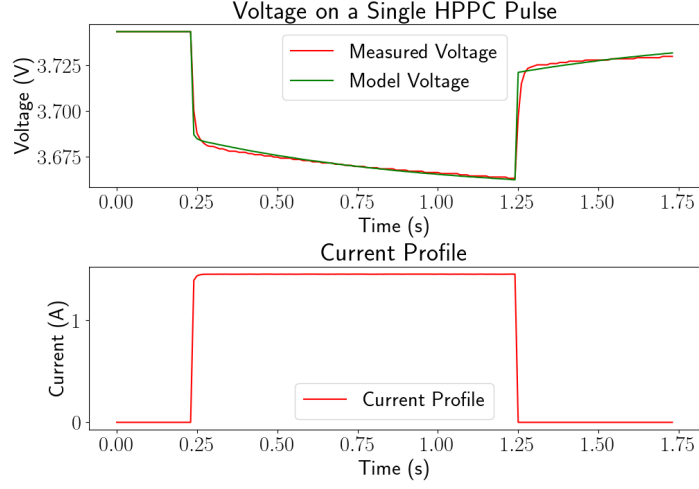
From the definition of current $I := \frac{dQ}{dt}$, we further have that for a single cycle,

$$s = 1 - \frac{1}{Q_0} \int_0^t I(\tau) d\tau,$$

under the assumption that Q_0 , and therefore h , stays constant during that cycle.

2.2 The Equivalent Circuit Model

As mentioned earlier, we want to model the battery as an electrical component which exerts specific behaviour in an electrical circuit. In electrical engineering, such models of more complicated components are frequently represented by equivalent circuits which only consist of basic components, mostly resistors, diodes, transistors, capacitors and inductors.

**Figure 1:** HPPC Pulse

Empirically, we know that a battery's voltage output looks roughly

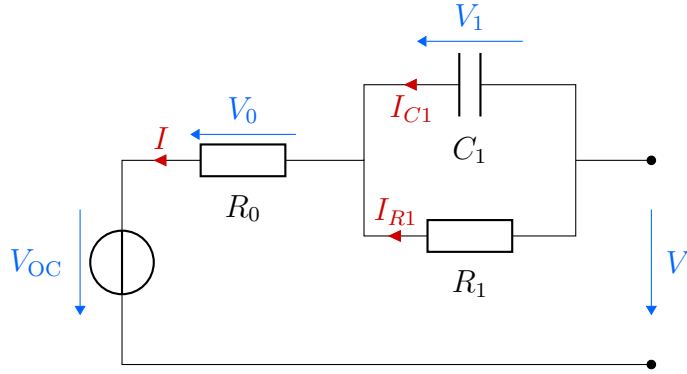


Figure 2: The Thevenin equivalent circuit model (ECM) with parameters $R_0 \in \mathbb{R}^+$, $R_1 \in \mathbb{R}^+$ and $C_1 \in \mathbb{R}^+$ and $V_{OC} \in \mathbb{R}^+$ the *open circuit voltage* which behaves according to a function $V_{OC}(s, h, T)$ dependent on s , h and T .

Kirchhoff's law further tells us that the currents $I_{R1} \in \mathbb{R}$ and $I_{C1} \in \mathbb{R}$ add up to the total current $I = I_{R1} + I_{C1}$, and that the voltages $V_0 \in \mathbb{R}$, $V_1 \in \mathbb{R}$ and V_{OC} sum up to $V = V_0 + V_1 + V_{OC}$. The capacitor behaves according to $I_{C1} = C_1 \frac{dV_1}{dt}$, while the resistors follow Ohm's law $V_0 = R_0 I$ and $V_1 = R_1 I_{R1}$.

2.3 Battery in an Electric Vehicle (EV)

On a graph (V_G, E) with edges $E = \{AB, AC, \dots\} \subseteq V_G \times V_G$ and vertices $V_G = \{A, B, \dots\}$, let $d_{AB} \in \mathbb{R}^+$ denote the distance between two vertices $A \in V_G$ and $B \in V_G$ (in meters), $x = x_{AB} \in [0, d_{AB}]$ the progress (current location) on the

route from vertex A to B (in meters), $v := \frac{dx}{dt}$ denote the current velocity with $v_{\max, AB} \in \mathbb{R}^+$ the maximum allowed velocity on AB (in meters per second). Then let $T_{\text{env}}(x) \in [-273.15, \infty)$ denote the temperature of the environment (in degrees Celsius) at location x .

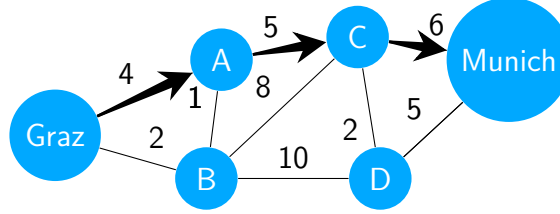


Figure 3: Path

Let $P \in \mathbb{R}$, $P := I \cdot V$ denote the (electrical) power the car draws from the battery (in Watts) so $P > 0$ corresponds to discharging the battery. This power is to be realised into a mechanical component $P_{\text{motor}} \in \mathbb{R}^+$ driving the car forwards, heating for the battery $P_{\text{heat}} \in \mathbb{R}^+$, $P_{\text{heat}} = c(T - T_{\text{env}})$, with $c \in \mathbb{R}^+$ the heat conduction constant describing the relation between the heater and battery, and power dissipation $P_{\text{diss}} \in \mathbb{R}^+$. While driving, $P = P_{\text{motor}} + P_{\text{heat}} + P_{\text{diss}}$. The acceleration of the car $a \in \mathbb{R}$ (in meters per second squared), where $a := \frac{dv}{dt} = \frac{d^2x}{dt^2}$ is decomposed into $a_m \in \mathbb{R}$, which directly impacts $P_{\text{motor}}(a_m)$, and the deceleration due to friction (air, etc.) $a_f(v) \in \mathbb{R}^-$, so that in total $a = a_m + a_f$.

On the graph (V_G, E) there exists a set of EV charging stations $V_{\text{charge}} \subseteq V_G$ where $P_{C, \text{charge}}$ denotes the possible charging power (in Watts) at the charging station vertex $C \in V_{\text{charge}}$ with $K_C \in \mathbb{R}^+$ the occuring costs per energy unit (in Euros per Watt second) and $t_C \in \mathbb{R}^+$ the charging time per charging station B (in seconds).

2.4 A Variational Optimisation Problem

Given source and destination vertices $A, Z \in V_G$ on the graph (V_G, E) , which connected set of edges $E_R \subseteq E$ connecting A to Z , set of visited charging stations $V_C \subseteq V_{\text{charge}}$ and charging times $\{t_C\}_{C \in V_C}$ visited on the route E_R , and driving behaviour $a_m(x, v, t, s, h, T_{\text{env}}, \dots)$, $a_m \in \mathcal{C}^1(\mathbb{P})$ with \mathbb{P} the parameter space ¹ minimises

1. the total travel time $t_{\text{total}} := \int_{V_R} \frac{1}{v} dx + \sum_{C \in V_C} t_C$,
2. the total cost of travel $K := \sum_{C \in V_C} P_{C, \text{charge}} t_C K_C$,

¹to be defined.. TODO

3. $-N$ where N is the highest possible number of repetitions (commutes from A to Z) with the same battery (requiring $h > 0$).

Formulated differently, we aim to minimise the functional $F \in \mathcal{C}(\mathbb{P})^*$, $F : \mathcal{C}(\mathbb{P}) \mapsto \mathbb{R}$ where either $F[\chi] = t_{\text{total}}$ or $F[\chi] = K$.

Charging station + street data could be obtained from [OpenStreetMap](#) by calling `osmfilter england-latest.o5m keep="amenity=charging_station"`².

2.5 Battery Aging

$$Q(t, c, s, I) = Q_0 - \frac{F_{\text{cycle}}(c)}{F_{\text{current}}(I)} - F_{\text{cal}}$$

By $F_{\text{cycle}}(c)$ we denote the *Current Agnostic Cycle Degradation Factor* models cycle aging at a single current. Also $F_{\text{current}}(I)$ represents the *Current Scaling Factor*, a value $0 < F_{\text{current}} \leq 1$ that incorporates behaviour where higher currents are worse for cycle aging. Finally, $F_{\text{cal}}(s, t)$, the *Calendar Degradation Factor*, ages the battery over time and accounts for suboptimal storage in terms of state of charge.

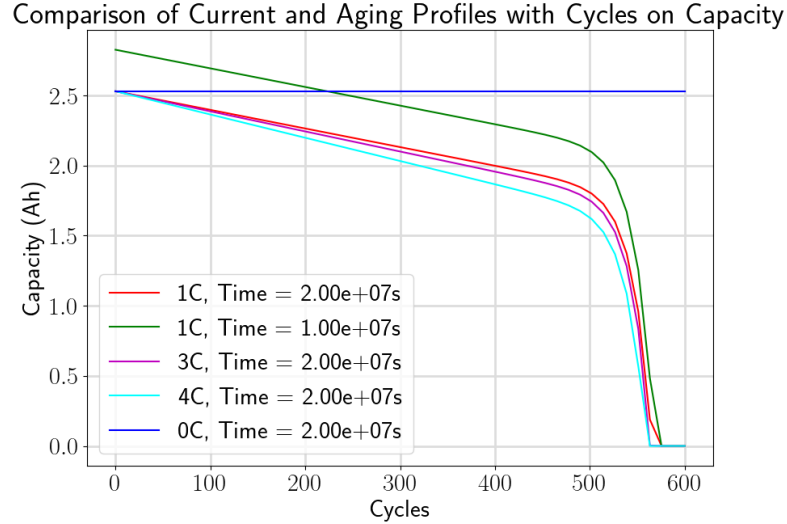


Figure 4: Aging

²OSM's public map data may be obtained from <https://download.geofabrik.de/>

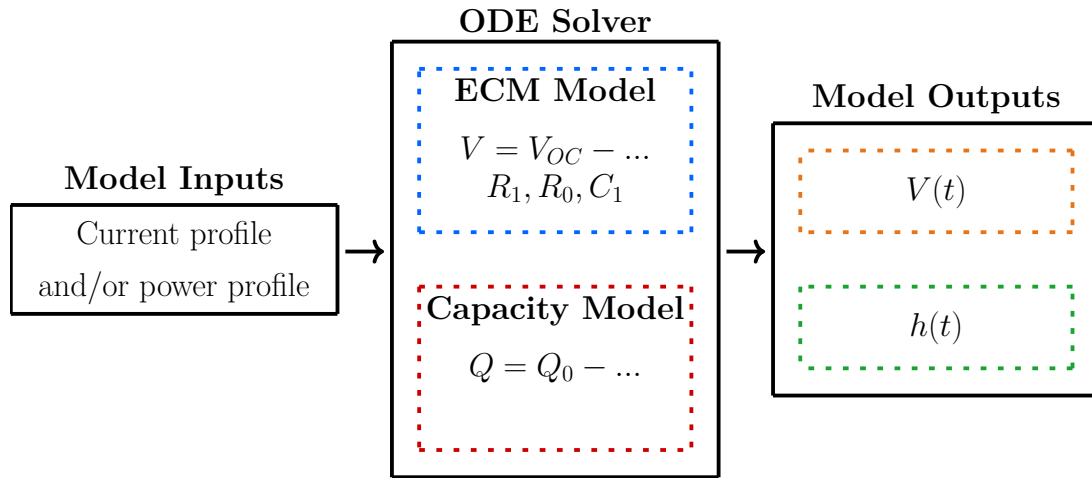
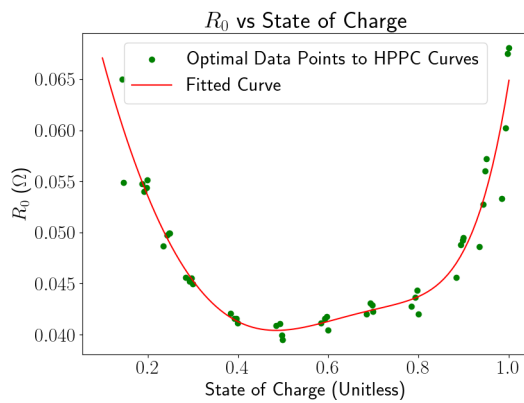
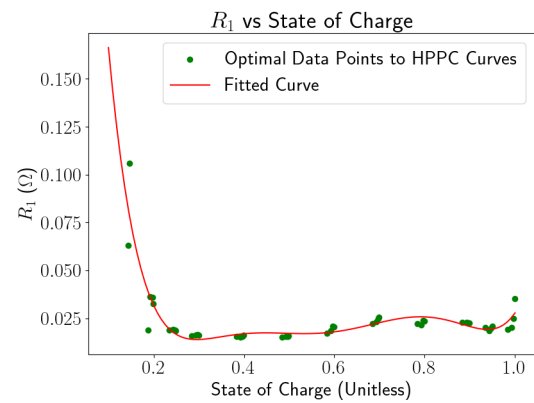


Figure 5: Overview

3 Numerical Simulation of a Car

3.1 Finding ECM Parameters

(a) Fit of R_0 as a function of the SOC (s).(b) Fit of R_1 as a function of the SOC (s).

3.2 Forward Euler Simulation

4 Metropolis-Hastings and A-Star

4.1 Definition: Undirected Graph

A graph $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$ is undirected if and only if $(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E \quad \forall v_i, v_j \in V$.

4.1 Shortest Path Finding

The graph data was retrieved using OSMNX (Boeing 2017) which itself is built on NetworkX (Hagberg, Schult and Swart 2008).

4.2 Monte-Carlo Optimisation

Optimise a large problem (huge state-space). Metric: Time! Could be anything. \Rightarrow Use Monte-Carlo Markov Chain Methods! Slightly perturb the route using a specific alteration technique. Metropolis-Hastings updates the state (route) based on

$$p_{\text{accept}} = \min \left(1, e^{-\beta(T_{\text{next}} - T_{\text{current}})} \right), \quad \text{with } \beta \in \mathbb{R}^+ \text{ a transition factor.}$$

Does a full numerical simulation of the drive. Stop to charge? Explore the state-space to some extent, and return the best route! Larger scales / maps (e.g. England) are not a problem!

```

1 newRoute = self.simulator.batgraph.perturbRoute(self.route)
2 delta = self.measureRoute(newRoute) - self.testedRoutes[self.route]
3 acceptanceProbability = min(1, math.exp(-delta / self.temperature))
4 if random.random() < acceptanceProbability:
5     self.route = newRoute
6     print(f"Accepted new route {newRoute} with delta: {delta:.2f}. Total:
7         ↳ {self.testedRoutes[newRoute]:.2f}.")
7 else:
8     print(f"Rejected route {newRoute} with delta: {delta:.2f}.")

```


4.3 Special Case: Granny's House

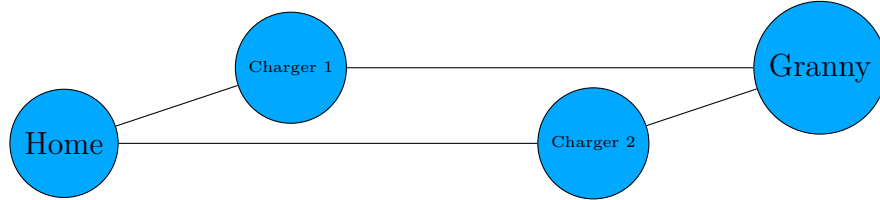


Figure 7: The exemplary problem “Granny’s House”, a special case of Problem (TODO).

4.4 Routing in Jericho

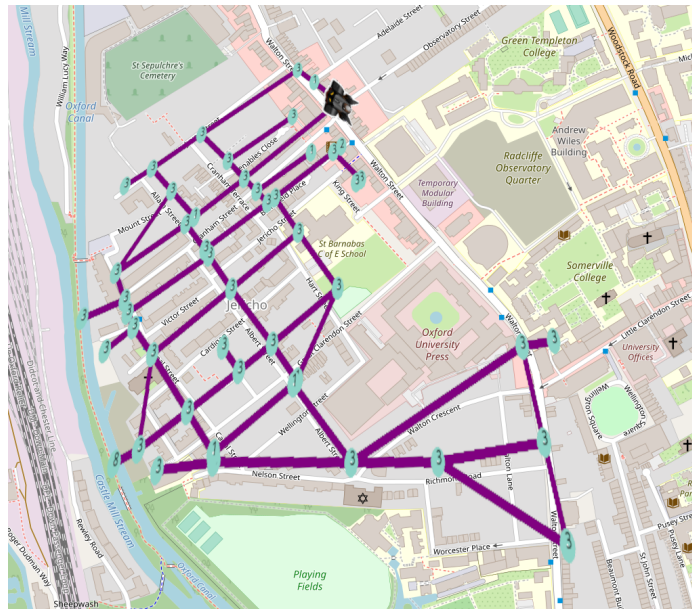


Figure 8: Overlay of the routing graph on a map of Jericho ([OpenStreetMap contributors 2023](#)), without adjusting for the Merkator projection, which leads to a slightly skewed appearance. The underlying data is exactly the same.

5 Conclusion

References

- Boeing, Geoff (2017). ‘OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks’. In: *Computers, Environment and Urban Systems* 65, pp. 126–139. ISSN: 0198-9715. DOI: [10.1016/j.compenvurbsys.2017.05.004](https://doi.org/10.1016/j.compenvurbsys.2017.05.004).
- Hagberg, Aric A., Daniel A. Schult and Pieter J. Swart (2008). ‘Exploring Network Structure, Dynamics, and Function using NetworkX’. In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught and Jarrod Millman. Pasadena, CA USA, pp. 11–15.
- Kollmeyer, Phillip (21st June 2018). ‘Panasonic 18650PF Li-ion Battery Data’. In: 1. Publisher: Mendeley Data. DOI: [10.17632/wykht8y7tg.1](https://doi.org/10.17632/wykht8y7tg.1). URL: <https://data.mendeley.com/datasets/wykht8y7tg/1> (visited on 02/03/2023).
- OpenStreetMap contributors (2023). *Planet dump retrieved from planet.osm.org*. URL: <https://www.openstreetmap.org> (visited on 08/03/2023).

A Simulation Code

```

1  from typing import Optional
2
3  from simulator.battery import Battery
4
5
6  class BatMobile:
7      """Represents the battery car ("Bat-Mobile") driving on an edge of the
8      ↪ BatGraph. Models everything."""
9
10     accelerationPerWatt = 100
11     friction = -120.0
12     currentJump = 0.1
13
14     def __init__(self):
15         self.battery = Battery()
16
17         self.position: float = 0 # meters
18         self.velocity: float = 0 # meters / second
19         self.acceleration: float = 0 # true acceleration that determines
20         ↪ velocity. in meters / second2.
21         self.motorAcceleration: float = 0 # goal acceleration, positively
22         ↪ affects the true acceleration
23         self.dragAcceleration: float = self.friction # acceleration caused by
24         ↪ drag, negatively affects the true accel,
25
26         self.P_motor: float = 0
27
28         self.sourceNode: Optional[str] = None # one of the nodes of the BatGraph
29         self.destinationNode: Optional[str] = None # another node of the
30         ↪ BatGraph
31
32     def iterate(self, dt):
33         self.battery.iterate(dt)
34
35         self.motorAcceleration = self.accelerationPerWatt * self.battery.voltage
36         ↪ * self.battery.current
37         self.simulateDrag()
38
39         self.P_motor = self.battery.voltage * self.battery.current
40
41         self.acceleration = self.motorAcceleration + self.dragAcceleration
42         self.velocity += self.acceleration * dt
43         self.velocity = max(self.velocity, 0) # velocity is at least 0..
44         self.position += self.velocity * dt
45
46     def accelerate(self, currentIncrease):
47         """Increases the current through the motor by the given amount in Amps,
48         therefore accelerating the car based on battery output.
49         """
50         self.battery.current += currentIncrease

```

```
45
46     def simulateDrag(self):
47         """Computes the drag acceleration which increases with  $v^2$ ."""
48         self.dragAcceleration = -6e-3 * self.velocity**2 + self.friction
49
50     def halt(self):
51         self.battery.current = 0
52         self.velocity = 0
53
54     def chargeUp(self):
55         self.battery.soc = 1
56         print("Charged up!")
```