



**UNIVERSIDADE ESTADUAL DE CAMPINAS**

Faculdade de Engenharia Mecânica

**FÁBIO DE SOUZA MORAES MORI**

**Rede neural profunda para estimar o SOC da  
bateria através de KPIs aplicados em Motorsport**

Campinas

2022

**FÁBIO DE SOUZA MORAES MORI**

# **Rede neural profunda para estimar o SOC da bateria através de KPIs aplicados em Motorsport**

Documento apresentado à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas para o Exame de Qualificação de Mestrado em Engenharia Mecânica.

Orientador: Prof. Dr. Janito Vaqueiro Ferreira

Campinas

2022

## RESUMO

O desenvolvimento de tecnologias em veículos elétricos vem crescendo no mundo de *Motorsports*, onde cada vez mais as categorias elétricas tem tido prestígio e ganham importância para servir como base no desenvolvimento de tecnologias e soluções que serão utilizadas nos veículos elétricos comerciais nas ruas. A bateria de íons de Lítio é a principal fonte de energia para estes veículos e o controle e gerenciamento de seus parâmetros é alvo de constante pesquisa e aprimoramento. Uma previsão correta do Estado de Carga (*SOC -State of Charge*) de uma célula é fator crucial para que se possa extrair o maior desempenho possível do conjunto de baterias e utilizar toda sua autonomia. Em *Motorsports*, ter uma previsão correta deste parâmetro é fundamental para a definição de estratégia de corrida entre piloto e equipe, e pode ser a chave para se obter a vitória.

O enfoque deste trabalho está na inclusão de Indicadores Chaves de Performance (*Key Performance Indicators - KPI*) de pilotagem em um modelo de uma Rede Neural Artificial (RNA) para aprimorar o valor estimado do SOC de uma célula de bateria, que é calculado através de um Filtro de Kalman Sigma Ponto (*Sigma-Point Kalman Filter - SPKF*) baseado nas equações de estado de um modelo de circuito equivalente (*Enhanced Self-Correcting - ESC*) de uma Bateria de Lítio. As características que definem o condutor influem diretamente na curva de descarga do SOC e são mais evidenciadas no mundo do automobilismo profissional, onde observamos diferenças muito maiores em comparação ao condutor comum. Adicionar o KPI ao modelo de previsão através de uma RNA objetiva melhorar a precisão dos resultados e consequentemente a performance da utilização do sistema de baterias. Uma melhora neste cálculo trará benefícios relacionados a autonomia e otimização do conjunto de baterias, que são dois dos fatores que mais geram entraves no desenvolvimento da tecnologia de propulsão elétrica nos automóveis atualmente.

Os resultados apontaram que o modelo híbrido desenvolvido possui ótima precisão, superando o SPKF em algumas situações. O estudo indica que este método poderá obter resultados precisos e adaptáveis as diferentes características observadas quando aplicados em *Motorsports*, onde a diferença das características de pilotagem está imposta como valores de entrada do sistema, diferentemente dos outros modelos já existentes.

**Palavras-chave:** Motorsports, Estado de Carga, Rede Neural Artificial, Aprendizagem Profunda, Filtro de Kalman Sigma Ponto, Circuito Equivalente.

## ABSTRACT

The development of new technologies about electric vehicles is growing up in the Motorsports, where new electrics car categories have been importance as a development basis and innovation will be used on the streets. Lithium ion battery is the most important power supply for electric vehicles and your management and control is constant research target. A correct estimation of Cell State of Charge is primordial to extract the highest performance of a battery pack and utilize all your capacity. In Motorsports this correct estimation is fundamental to define the strategy race between the driver and the engineer and may be the key to victory.

This work includes the driver Key Performance Indicator (KPI) into a Artificial Neural Network (ANN) model to improve the State of Charge (SOC) estimated by Sigma-Point Kalman Filter (SPKF) based on state equations of Enhanced Self-Correcting (ESC) model of Lithium ion battery. The driver characteristics has direct influence on discharge curve of SOC and this is most important and significant with professionals' drivers, compared to ordinary conductors. The SOC estimate model with KPI and ANN help to improve the accuracy and performance about the system. The better estimate will bring advantages in the use of autonomy and optimization of the battery pack, the two factors that need to be improved with new technologies.

This document presents all the development and what can be explored in future works. The work has divided into 3 parts, the first calculates the ESC model parameters of Lithium Ion Cell Battery with the Low Current Open Circuit Voltage and Dynamic Urban Dynamometer Driving Schedule battery laboratory dataset. Then the Sigma-Point Kalman Filter estimates the State of Charge value and the final step extracts the KPIs from the (*Urban Dynamometer Driving Schedule* – UDDS) discharge current profile and input this value with the SOC estimated by SPKF into a ANN to get a new estimated value, with greater precision.

The results showed that the hybrid model has great precision, surpassing the only the SPKF estimated in some situations. This study indicates that this method can get accurate and adaptable results in Motorsports applications, where the drivers characteristics are input data from the modeling system.

**Keywords:** Motorsports, State of Charge, Artificial Neural Network, Deep Learning, Sigma-Point Kalman Filter, Enhanced Self-Correcting Model.

## LISTA DE ILUSTRAÇÕES

Figura 2.1 – Figura extraída do artigo <a href="#">Saxena <i>et al.</i> 2012</a> . . . . .	19
Figura 2.2 – Figura extraída do artigo <a href="#">Xing <i>et al.</i> 2014</a> . . . . .	26
Figura 2.3 – Figura extraída do artigo <a href="#">Guo <i>et al.</i> 2019</a> . . . . .	27
Figura 2.4 – Figura extraída do artigo <a href="#">Hannan <i>et al.</i> 2018</a> . . . . .	28
Figura 2.5 – Figura extraída do artigo <a href="#">He <i>et al.</i> 2018</a> . . . . .	30
Figura 2.6 – Figura extraída do artigo <a href="#">Kharazmi <i>et al.</i> 2019</a> . . . . .	33
Figura 2.7 – Figura extraída do artigo <a href="#">He <i>et al.</i> 2014</a> . . . . .	33
Figura 3.1 – Fluxograma Geral - Incluindo relação OCV-SOC . . . . .	37
Figura 3.2 – Fluxograma Geral - Incluindo Modelo A123 . . . . .	38
Figura 3.3 – Fluxograma Geral - Incluindo SPKF . . . . .	40
Figura 3.4 – Gráfico Pi Toolbox . . . . .	43
Figura 3.5 – Gráfico Pi Toolbox . . . . .	43
Figura 3.6 – Fluxograma Geral - Incluindo KPI . . . . .	44
Figura 3.7 – Fluxograma Geral Final . . . . .	46
Figura 3.8 – Unidade de Rede Neural . . . . .	47
Figura 4.1 – Tensão [V] nos passos 1 e 2 para todas as temperaturas. . . . .	50
Figura 4.2 – OCV - A123 à 35°C . . . . .	51
Figura 4.3 – OCV - A123 à 35°C . . . . .	51
Figura 4.4 – DYN - A123 à 35°C . . . . .	52
Figura 4.5 – DYN - A123 à 35°C . . . . .	53
Figura 4.6 – Relação OCV e SOC à -25°C . . . . .	54
Figura 4.7 – Relação OCV e SOC à 25°C . . . . .	54
Figura 4.8 – Relação OCV e SOC à 35°C . . . . .	55
Figura 4.9 – Parâmetro $Q$ . . . . .	56
Figura 4.10–Parâmetro $\eta$ . . . . .	56
Figura 4.11–Parâmetro $\gamma$ . . . . .	57
Figura 4.12–Parâmetro $M$ . . . . .	57
Figura 4.13–Parâmetro $M_0$ . . . . .	58
Figura 4.14–Parâmetro $R_0$ . . . . .	58

Figura 4.15–Parâmetro $RC$	59
Figura 4.16–Parâmetro $R$	59
Figura 4.17–Estimativa SOC SPKF	60
Figura 4.18–Erro SPKF	60
Figura 4.19–Estimativa SOC SPKF < 50%	61
Figura 4.20–Sinal de Corrente do perfil UDDS	62
Figura 4.21–Sinais de Freio e Acelerador do perfil UDDS	62
Figura 4.22–Velocidade do Freio	63
Figura 4.23–Liberação do Freio	63
Figura 4.24–Agressividade do Freio	64
Figura 4.25–KPI do Freio	64
Figura 4.26–Velocidade do Acelerador	65
Figura 4.27–Liberação do Acelerador	66
Figura 4.28–Agressividade do Acelerador	66
Figura 4.29–KPI do Acelerador	67
Figura 4.30–Resultado da estimativa do SOC pelos modelos	68
Figura 4.31–Resultado da estimativa dos modelos com SOC menor que 50%	69
Figura 4.32–Resultado da estimativa dos modelos com SOC menor que 20%	69
Figura 4.33–Sinal UDDS	70
Figura 4.34–Sinal FUDS	71
Figura 4.35–Resultado Rede Neural Perfil FUDS - 35°C	72
Figura 4.36–Sinal FUDS - KPI de freio	72
Figura 4.37–Sinal FUDS - KPI do acelerador	73
Figura 4.38–Função de perda para o sinal FUDS à 35°C	73
Figura 4.39–Sinal FUDS - KPI de freio com média móvel	74
Figura 4.40–Sinal FUDS - KPI do acelerador com média móvel	74
Figura 4.41–Resultado da RNA com perfil FUDS e KPI com média móvel à 35°C	75
Figura 4.42–Função de perda para o sinal FUDS e KPI com média móvel à 35°C	75
Figura 6.1 – Fórmula E temporada 2019 - 2020	79

## **LISTA DE TABELAS**

Tabela 4.1 – Configuração Rede Neural . . . . .	67
Tabela 4.2 – Configuração Rede Neural . . . . .	71

## **LISTA DE ABREVIATURAS E SIGLAS**

ANFIS	Adaptative Neural Fuzzy Inference System
BMS	Battery Management System
BPNN	Back Propagation Neural Network
BSA	Backtracking Search Algorithm
CALCE	Center for Advanced Life Cycle Engineering
DST	Dynamic Stress Test
DYN	Dynamic Urban Dynamometer Driving Schedule
ECM	Equivalent Circuit Model
EKF	Extended Kalman Filter
ELU	Exponential Linear Unit
EM	Electrochemical Model
EODV	End of Discharge Voltage
EODV	End of Discharge Voltage
ESC	Enhanced Self-Correcting)
FK	Filtro de Kalman
FP	Filtro de Partículas
FUDS	Federal Urban Driving Schedule
HIL	Hardware-in-the-Loop
HPPC	Hybrid Pulse Power Characterisct
KPI	Key Performance Indicador
LS	Least Square

MAEE	Maximum Absolute Estimated Error
NASA	National Aeronautics and Space Administration
OCV	Open Circuit Voltage
P2D	Pseudo-Two-Dimensional
PDE	Partial Differential Equation
PINNs	Physics Informed Neural Networks
RMSE	Root Mean Square Error
RNA	Rede Neural Artificial
RUL	Remaining Usefull Life
SOC	Estado de Carga
SOE	State of Energy
SOH	State of Health
SPKF	Sigma Point Kalman Filter
UDDS	Urban Dynamometer Driving Schedule
UKF	Unscented Kalman Filter
VPINN	Variational Physics Informed Neural Network

## LISTA DE SÍMBOLOS

$\eta$	Eficiência de Coulomb
$DoD$	Percentagem de carga retirada da bateria durante a descarga
$R_p$	Resistência em paralelo
$C_p$	Capacitância em paralelo
$R_0$	Resistência Ôhmica
$rp$	Raio da partícula
$L_{pos}$	Espessura do eletrodo
$\epsilon$	Fração de volume do material ativo
$Q$	Capacidade total da célula
$i$	Corrente
$v$	Tensão
$z$	Estado de carga
$V_{ajus}$	Tensão ajustada
$T$	Temperatura
$i_R$	Equação de estado do algoritmo de teste dinâmico
$s$	Equação de estado do algoritmo de teste dinâmico
$h$	Equação de estado do algoritmo de teste dinâmico
$rms$	Raiz do erro quadrático médio
$\gamma$	Parâmetro de histerese do modelo de circuito equivalente da bateria
$M$	Parâmetro de histerese do modelo de circuito equivalente da bateria
$M_0$	Parâmetro de histerese do modelo de circuito equivalente da bateria

$X_0$	Incerteza do estado inicial do SPKF
$V$	Sensor de tensão na equação de saída do SPKF
$W$	Sensor de corrente na equação de estado do SPKF
$N_x$	Elementos do vetor de estado do SPKF
$N_x$	Elementos do vetor de medição do SPKF
$N_u$	Elementos do vetor de entrada do SPKF
$N_w$	Ruído do processo do sistema do SPKF
$N_w$	Ruído do sensor do sistema do SPKF
$N_a$	Número de elementos no vetor de estado aumentado do SPKF
$X_{hat}$	Valores de estimativa de estado do SPKF
$y_{hat}$	Valores de previsão de tensão do SPKF
$L$	Matriz de ganho do SPKF
$\Sigma XY$	Matriz de covariância cruzada do SPKF
$\Sigma Y$	Matriz de covariância de inovação do SPKF
$\Sigma X$	Matriz de covariância de erro do SPKF
$X_a$	Matriz de estado aumentado do SPKF
$h_k$	Matriz de histerese do SPKF
$r$	Medição residual
$\Delta t$	Intervalo de tempo
$zk_{bnd}$	Bordas de erro do SOC definidas na simulação
$V_{min}$	Tensão mínima
$V_{max}$	Tensão máxima
$I_{discharged}$	Corrente de descarga da bateria
$I_{charged}$	Corrente de recarga da bateria

# SUMÁRIO

<b>1</b>	<b>Introdução</b>	<b>15</b>
1.1	Introdução ao tema	15
1.2	Desenvolvimentos Propostos	16
1.3	Resumo da Tese	17
<b>2</b>	<b>Revisão do Estado da Arte Atual</b>	<b>18</b>
<b>3</b>	<b>Metodologia</b>	<b>34</b>
3.1	Teste 1: Tensão de Circuito Aberto	34
3.2	Teste 2: Perfil Dinâmico de Carga e Descarga	36
3.3	Algoritmo da Tensão de Circuito Aberto	37
3.4	Algoritmo do Teste Dinâmico	38
3.5	Filtro de Kalman Ponto Sigma	40
3.6	Indicadores de Performance do Piloto	42
3.6.1	Agressividade e velocidade no pedal de freio	44
3.6.2	Agressividade e velocidade no pedal do acelerador	45
3.7	Rede Neural de Aprendizagem Profunda	46
<b>4</b>	<b>Resultados e Comentários</b>	<b>50</b>
4.1	Teste 1: Tensão de Circuito Aberto	50
4.2	Teste 2: Perfil Dinâmico de Carga e Descarga	52
4.3	Algoritmo da Tensão de Circuito Aberto	53
4.4	Algoritmo do Teste Dinâmico	55
4.5	Filtro de Kalman Ponto Sigma	59
4.6	Indicadores de Performance do Piloto	61
4.6.1	Agressividade e velocidade no pedal de freio	61
4.6.2	Agressividade e velocidade no pedal do acelerador	65
4.7	Rede Neural de Aprendizagem Profunda	67
4.8	Validação da RNA	70
<b>5</b>	<b>Conclusão</b>	<b>76</b>
<b>6</b>	<b>Trabalhos Futuros</b>	<b>77</b>

<b>Referências . . . . .</b>	<b>80</b>
<b>Anexos</b>	<b>83</b>
<b>ANEXO A Algoritmos em Matlab . . . . .</b>	<b>84</b>
A.1 Algoritmos da Tensão de Circuito Aberto . . . . .	84
A.1.1 runProcessOCV.m . . . . .	84
A.1.2 processOCV.m . . . . .	86
A.2 Algoritmos do Teste Dinâmico . . . . .	95
A.2.1 runProcessDynamic.m . . . . .	95
A.2.2 processDynamic.m . . . . .	97
A.2.3 setupDynData.m . . . . .	115
A.3 Algoritmos do Filtro de Kalman Ponto Sigma . . . . .	116
A.3.1 runSPKF.m . . . . .	116
A.3.2 initSPKF.m . . . . .	120
A.3.3 iterSPKF.m . . . . .	123
A.4 Algoritmos dos Indicadores de Performance do Piloto . . . . .	128
A.4.1 KPIDYN.m . . . . .	128
A.4.2 signals.m . . . . .	134
A.4.3 bspeed.m . . . . .	135
A.4.4 brelease.m . . . . .	136
A.4.5 bmeanrelease.m . . . . .	136
A.4.6 baggresssion.m . . . . .	138
A.4.7 bmeanaggresssion.m . . . . .	139
A.4.8 tspeed.m . . . . .	140
A.4.9 trelease.m . . . . .	141
A.4.10 tmeanrelease.m . . . . .	142
A.4.11 taggression.m . . . . .	143
A.4.12 tmeanaggression.m . . . . .	144
<b>ANEXO B Algoritmos em Python . . . . .</b>	<b>146</b>
B.1 Algoritmos da Rede Neural de aprendizagem profunda . . . . .	146
B.1.1 SOC.py . . . . .	146
B.1.2 train.py . . . . .	152



# 1 INTRODUÇÃO

A aplicação da inteligência artificial pode ser uma solução inovadora para um melhor desempenho de veículos elétricos em competições automobilísticas. Ao passo que a energia elétrica em veículos favorece o crescimento sustentável em regiões onde a matriz energética também é (já que a geração de energia será necessária para recarregar as baterias), por outro lado, gera um certo obstáculo relacionado ao potencial de velocidade e autonomia dos carros. Avanços recentes nas equações que regem a eletroquímica nas baterias, circuitos elétricos equivalentes mais precisos ou nos algoritmos como Redes Neurais e Filtros de Kalman ainda não foram suficientes em obter com maior precisão o valor de carga existente nas baterias para aplicabilidade a nível de uma competição automobilística. Posto isso, cabe o questionamento, se a tecnologia das baterias de íons de lítio atingiu o seu limite, o que desafia e, de certa forma, pode inclusive ameaçar a categoria da competição em questão, ou se ainda podemos melhorar a estimativa dos algoritmos embarcados através da inclusão de novos parâmetros. Esta é uma pergunta instigante a ser respondida, cuja temática norteia esta dissertação.

## 1.1 Introdução ao tema

As mudanças climáticas avançam a cada ano e têm sido pauta de discussões entre líderes mundiais e entidades governamentais e não governamentais. Diante dessa problemática atual, a busca pela sustentabilidade deve ser priorizada nos diversos setores e serviços. Montadoras de veículos urbanos estão investindo largamente na produção de veículos movidos a energia elétrica com objetivo de torná-los mais acessíveis e adaptados à sociedade. No contexto do automobilismo, o desenvolvimento de uma categoria 100% elétrica atende a premissa de sustentabilidade no que tange à redução da emissão de gás carbônico ( $\text{CO}_2$ ) na atmosfera. Entretanto, existem desafios para a categoria de carros elétricos do tipo fórmula, um deles é a bateria, em função da sua vida útil limitada e seu posterior descarte, que se ocorrido de forma indevida é nociva ao meio ambiente.

O outro é o desempenho do carro em competições automotivas, pois a autonomia do veículo é definida pela quantidade de carga disponível em uma célula de bateria de lítio e aumentar o número de células dentro de um veículo se torna inviável diante do aumento da massa, volume e custo que seria adicionado ao carro. É por isso, inclusive, que as baterias

utilizadas são de lítio e não de chumbo. Avanços tecnológicos da célula de bateria, bem como o desenvolvimento de redes neurais nos algoritmos de predição de carga, permitiram o uso da bateria próximo ao limite de corrente e tensão e viabilizaram a consolidação dos veículos elétricos e da categoria automobilística em questão. Ainda assim, a potência do veículo pode oscilar, sendo necessária a adoção de uma estratégia onde a carga da bateria seja administrada até o término da prova pelo piloto.

Para mensurar a carga disponível nas células são utilizadas a tensão, em Volts, e a corrente, em Amperes, mas a relação da carga disponível com os valores medidos não é direta e o uso de algoritmos é necessário, sendo assim também o desenvolvimento deles torna o cálculo mais eficiente. O valor de carga disponível em uma célula é denominado SOC e existem diversos algoritmos desenvolvidos para esta aplicação, embora hoje em dia muito mais eficientes, ainda existe margem para erro nesta previsão principalmente nos valores finais.

Portanto, um questionamento acerca da possibilidade de incluir a personificação de cada piloto dentro do algoritmo é uma hipótese a ser testada, afinal, assim como em veículos à combustão, não somente o mapa de injeção define o consumo, mas a performance de cada piloto é fator relevante para prever o real consumo durante o trajeto. O presente trabalho, propôs um modelo híbrido inovador de algoritmo utilizando Filtro de Kalman (FK), um modelo de circuito equivalente (*Equivalent Circuit Model - ECM*) da bateria e RNAs para inclusão dos parâmetros de pilotagem e cálculo do SOC com aplicação em *Motorsport*.

## 1.2 Desenvolvimentos Propostos

Há um grande esforço em otimizar algoritmos e modelos de bateria, reduzir a ordem de equações diferenciais para aplicá-las em sistemas embarcados e melhorar a precisão do sistema, desenvolver modelos híbridos para melhorar o desempenho do algoritmo, dentre outros. Algoritmos de estimativa para cálculo do SOC utilizam modelos eletroquímicos ou baseados em dados bastante desenvolvidos e comuns de serem vistos em aplicações embarcadas. Porém, para que se possa chegar a um nível ainda maior de precisão, é necessário considerar fatores que não estão vinculados as características da bateria, mas sim ao agente principal que vai determinar o consumo de energia: o piloto.

Neste contexto, os KPIs da pilotagem são parâmetros que agregam dados relevantes para maior precisão na estimativa do valor de SOC. A agressividade no pedal do acelerador e suavidade no momento de tirar o pé do pedal, estão diretamente relacionados com a quantidade

de carga a ser drenada das baterias. No caso do pedal de freio temos a mesma situação. A agressividade do pedal de freio e a suavidade no momento de tirar o pé estão diretamente relacionados com a quantidade de corrente a ser utilizada para recarregar as baterias no momento da regeneração da energia.

Este trabalho elaborou um algoritmo híbrido para estimativa do SOC em veículos de competição, a partir de um modelo proposto e desenvolvido por Gregory Plett, University of Colorado Colorado Springs - UCCS, disponibilizado e protegido por direitos autorais, podendo ser utilizado apenas para fins educativos e informativos, tal como este trabalho, que utiliza um modelo de ECM e um KF para estimar o SOC da bateria. Para acrescentar o desenvolvimento baseado nos dados de pilotagem e criar um modelo híbrido foi desenvolvido em conjunto uma RNA com parâmetros externos, os KPIs, que também alteram a curva de consumo de SOC, parametrizando cada piloto e estimando novamente o SOC, com maior precisão e de forma mais personalizada.

### **1.3 Resumo da Tese**

O trabalho se inspirou no estudo desenvolvido por Gregory Plett, onde foi estimado o SOC da bateria aplicando um KF. Para a realização de testes com os algoritmos é necessário um banco de dados de testes de células de baterias. Para isso optou-se pelo banco de uma célula A123 de LiFeP04 de 3,4 V e um perfil de carga e descarga de um veículo urbano, ambos disponibilizados pelo CALCE. A escolha foi embasada por modelo semelhante ao que é utilizado no automobilismo e também pelo perfil dinâmico ser o que mais se assemelha à aplicação deste trabalho.

Após aplicar o modelo de Gregory Plett para a célula A123, foi iniciado o desenvolvimento do modelo híbrido proposto. Conforme os dados do teste do perfil dinâmico de carga e descarga da bateria, foram aplicadas as fórmulas para o cálculo dos KPIs dos pedais de freio e de acelerador, correlacionando estes valores com os sinais de carga e de descarga da bateria. Com os valores extraídos do teste, foi utilizada uma RNA para uma nova estimativa do SOC, com o objetivo de otimizar a precisão do cálculo anterior do KF. Desta forma, um modelo híbrido desenvolvido tanto da parte teórica da bateria quanto da aplicação do problema pode configurar um modelo ideal no desenvolvimento de uma inteligência artificial embarcada para uma melhor precisão dos algoritmos de estimativa do SOC.

## 2 REVISÃO DO ESTADO DA ARTE ATUAL

Foram pesquisados e estudados trabalhos relacionados ao desenvolvimento de algoritmos para obtenção do valor do SOC de uma célula de bateria. Um dos principais ramos de pesquisa dos trabalhos são aqueles que focam na eletroquímica da célula (*Electrochemical Model - EM*), visando melhorar o equacionamento que prevê o funcionamento das células através das reações químicas internas que ocorrem durante a sua utilização. Temos também, os trabalhos que concentram sua pesquisa em melhorar o circuito elétrico equivalente que representa estes fenômenos internos e que, de certa forma, replica estas variações para podermos assim calcular o valor do SOC. Por fim, temos as pesquisas que tem como objetivo encontrar o melhor modelo de algoritmo que retorne com maior precisão o cálculo do SOC, independente da forma como ele é calculado, seja por modelos EM ou modelos ECM.

Pela sua importância no desenvolvimento de novas tecnologias, o estudo relacionado a baterias é muito amplo no meio científico. Os algoritmos baseados em dados são mais populares em aplicações embarcadas devido ao seu desempenho relativamente mais rápido em comparação a outros modelos, em teoria mais complexos. [Saxena et al. 2012](#) ressaltam as incertezas inerentes a este sistema mostrando a análise preliminar do desenvolvimento de um algoritmo baseado em dados (ver fluxograma na Figura 2.1) e suas lições aprendidas utilizando um banco de dados, disponibilizado pela *National Aeronautics and Space Administration* (NASA), obtido em laboratório de carga e descarga de células de íons de Lítio. Os resultados mostraram que os ruídos das medições utilizadas como dados interferem muito na eficácia dos modelos, assim como os transientes de carga e descarga obtidos em laboratórios que podem ser diferentes das aplicações reais do problema e também a extração dos parâmetros da célula, que exige muitas aproximações sensíveis ao resultado encontrado.

[Ning et al. 2016](#), da mesma forma, ressaltam que a variação dos parâmetros internos pode aumentar drasticamente o erro, fazendo uma crítica aos modelos com parâmetros internos estáticos e expondo suas desvantagens, tal como a necessidade de um complexo modelo para obtenção dos parâmetros iniciais que não serão corrigidos durante a aplicação. Sendo assim, propõe a utilização de um modelo de bateria adaptável utilizando um algoritmo de modo deslizante (*Sliding Mode Algorithm*) para reduzir os erros conforme os parâmetros internos da bateria variam. Os resultados foram obtidos através de dados com perfil UDDS e a estimativa

do SOC obteve resultados de erro inferiores a 2%.

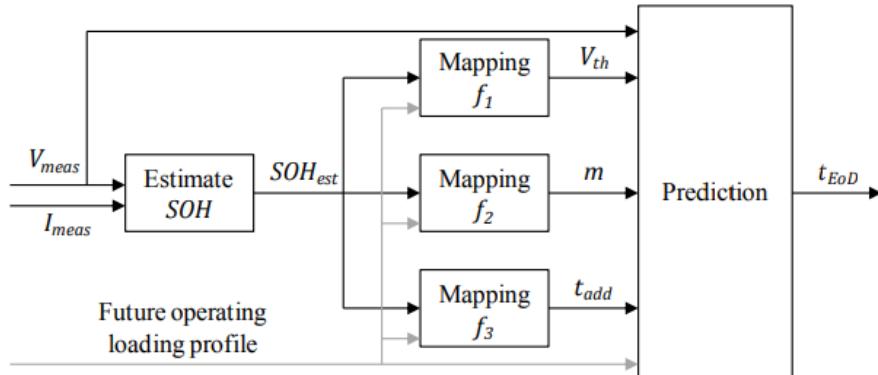


Figura 2.1 – Figura extraída do artigo [Saxena et al. 2012](#)

[Zhang et al. 2014](#) apresentam um trabalho mostrando a degradação dos parâmetros internos envelhecendo uma célula de LiCoO<sub>2</sub> em ciclos com alta temperatura e monitorando seus parâmetros. O mecanismo de degradação foi discutido usando um modelo multi-físico e parâmetros-chave, razões para o enfraquecimento da capacidade e o aumento da resistência interna foram analisados em detalhes. Todas as evidências indicaram que a reação de formação do filme da interface do eletrólito sólido (*Solid Electrolyte Interface*) é a principal causa de degradação da bateria em alta temperatura ambiente.

Dentre muitos estudos que vão sendo desenvolvidos relacionados a medição de parâmetros de uma célula de Lítio, [Chang 2013](#) discorre que o SOC é um parâmetro muito importante não apenas para sabermos a carga disponível na célula, mas conhecer seu valor exato nos permite melhorar o desempenho, proteger a célula, evitar sobrecargas e também criar estratégias de controle de sua utilização mais racionalmente para conseguir economizar energia, consequentemente aumentar sua vida útil. Ele faz uma revisão detalhada sobre diversos métodos matemáticos utilizados para estimativa do SOC, separando os métodos em: medição direta, contagem de Coulomb, sistemas adaptativos e métodos híbridos.

Os modelos de medição direta apresentados são o de tensão de circuito aberto (*Open Circuit Voltage - OCV*), tensão do terminal (*Terminal Voltage Method*), impedância (*Impedance Method*) e impedância espectroscópica (*Impedance Spectroscopy Method*). São apresentados dois modelos de contagem de Coulomb, o padrão e outro modificado (*Modified Coulomb Counting Method*). Os modelos adaptativos são a rede neural de retro-propagação (*Back Propagation Neural Network*), rede neural de função de base radial (*Radial Basis Function Neural Network*), lógica Fuzzy (*Fuzzy Logic Method*), máquina de vetores (*Support Vector Machine*), rede neu-

ral Fuzzy (*Fuzzy Neural Network*) e o Filtro de Kalman. Os modelos híbridos apresentados são o de contagem de Coulomb com medição da forma eletromotriz (*Coulomb Counting and Electromotive Force Combination*), contagem de Coulomb com FK (*Coulomb Counting and Kalman Filter*) e um sistema por unidade com um FK extendido (*Per-Unit System and Extended Kalman Filter Combination*). Após a apresentação dos métodos, é dito que o futuro desta tecnologia deve estar nos modelos híbridos, pesquisas em como adicionar características reais da aplicação ao modelo de predição, considerar os efeitos de envelhecimento e alteração de parâmetros internos da bateria com modelos adaptativos, além de um contínuo desenvolvimento nos algoritmos de redes neurais baseado em dados para se obter uma maior precisão.

[Stefanopoulou e Kim 2015](#) descrevem os conceitos teóricos como o SOC, o estado de saúde (*State of Health – SOH*) e o estado de energia (*State of Power*), todos relacionados ao gerenciamento de carga de uma bateria e que são controlados durante sua utilização pelo sistema de gerenciamento (*Battery Management System – BMS*). O gerenciamento térmico também foi abordado, já que se evidenciou o desempenho ruim das baterias de íons de Lítio em temperaturas baixas.

[How et al. 2019](#) apresentam um trabalho de revisão de vários trabalhos com diferentes métodos de estimativa do SOC, destacando os métodos baseados em modelagem e em dados. Os métodos baseados em modelagem tentam modelar o comportamento físico e químico de uma célula de bateria através de equações diferenciais matemáticas complexas para poder estimar o SOC com maior precisão. Já os modelos baseados em dados, necessitam de uma grande base de dados para poder treinar seus algoritmos e torná-los o mais preciso possível.

É descrito primeiramente o método de consulta da tabela OCV-SOC, que fornece uma relação direta entre esses dois parâmetros. A seguir é mostrado o tradicional método de contagem de Coulomb (*Coulomb Counting*), antes de se aprofundar nos modelos de modelagem e dados. Na descrição dos métodos de modelagem são apresentadas as diversas equações diferenciais que regem o comportamento da bateria durante sua utilização e o objetivo é desenvolver um algoritmo que solucione essas equações. Por fim, os modelos baseados em dados são apresentados: redes neurais (*Neural Network*), aprendizagem profunda (*Deep Learning*), máquina de vetores (*Support Vector Machine*), lógica Fuzzy (*Fuzzy Logic Method*) e modelos híbridos.

O principal desafio apresentado na estimativa do SOC é melhorar a precisão, eficiácia e robustez dos algoritmos com baixa complexidade computacional para que o mesmo possa

ser implementado em um hardware de BMS de baixo custo. Os erros na estimativa do SOC são gerados muitas vezes pelos próprios sensores de corrente e tensão utilizados, modelos matemáticos imprecisos da bateria, SOC inicial, seleção dos parâmetros internos, dentre outros. Um dos grandes desafios que precisam ser solucionados é a inconsistência dos valores de SOC em cada célula que compõe o grupo de baterias, já que dentro de um grupo de baterias que formam os pacotes de alta tensão existem diversas células conectadas em série e paralelo, o que faz com que a estimativa do SOC para o grupo de células se torne mais difícil. Altas temperaturas também prejudicam muito a estimativa, já que provocam efeitos que alteram o comportamento das células como a fuga térmica que pode ser originada tanto por motivos mecânicos, como colisões ou acidentes, motivos elétricos, como sobrecarga e também por altas temperaturas.

Conforme ocorre a utilização destas células, existe uma redução da sua capacidade de carga total e este também é um dos fatores que prejudicam a estimativa já que alteram, por exemplo, os valores iniciais e máximos a serem considerados nos cálculos do algoritmo. Outro problema muito importante está no material composto da bateria de íons de Lítio que, embora tenha ótimas características, podem variar seu desempenho de forma significante dependendo de seus eletrodos positivo e negativo. Monitorar e garantir que a bateria esteja funcionando corretamente e de forma segura, sem que efeitos de sobretensão, sobrecorrente, super aquecimentos ou baixas temperaturas aconteçam, também é um desafio e está diretamente relacionado à precisão do cálculo do SOC, já que estes efeitos alteram as características das células de bateria.

Existem desafios relacionados também a bancada de testes que é utilizada, relacionado aos ruídos, interferência magnética dos sinais aquisitados e precisão do equipamento, já que a obtenção desses dados está diretamente relacionada com a qualidade dos dados que poderão ser utilizados no desenvolvimento de um algoritmo. O último desafio mencionado é justamente conseguir implementar um algoritmo de estimativa do SOC em um BMS de baixo custo, que não necessite de muito armazenamento de memória e que seja rápido computacionalmente para que possa ser utilizado em uma aplicação embarcada.

A conclusão desta revisão é que tanto os algoritmos baseados em modelagem quanto os baseados em dados produziram resultados e avanços significativos na estimativa do SOC, sendo que, para uma aplicação onde o modelo do sistema seja conhecido previamente, os algoritmos baseados nas equações diferenciais são mais eficientes, mas para aplicações onde o sistema não é totalmente conhecido, o modelo baseado em dados pode ter desempenho superior.

Métodos como o de contagem de Coulomb são de malha aberta e não são considerados eficientes pois acumulam muito erro ao longo de sua aplicação, assim como o modelo de consulta da tabela OCV–SOC. Os modelos eletroquímicos têm um custo computacional alto para solucionar as equações diferenciais parciais (*Partial Differential Equation – PDE*) e este fator é um limitante para sua aplicação embarcada. Os modelos ECM precisam de um tempo substancial para estimar os parâmetros e obter um equilíbrio entre acurácia e complexidade computacional. O FK contém complexas operações matriciais que dificultam sua aplicação em microcontroladores de baixo custo, além de que seu desempenho degrada conforme as incertezas da configuração do modelo da bateria, nível de ruído, parâmetros físicos e condições iniciais. As redes neurais artificiais são eficientes para estimar o SOC sob diversas características de aplicação, mas seu desempenho está limitado a duração, qualidade dos dados e definição de hiperparâmetros utilizados em seu treinamento. A lógica Fuzzy precisa de parâmetros muito específicos e difíceis de se obter devido as variações dos parâmetros da bateria sob diferentes perfis de carga e descarga. O sistema de inferência *neuro-fuzzy* adaptativo (*Adaptative Neural Fuzzy Inference System – ANFIS*) necessita de uma grande quantidade de espaço de memória para armazenamento dos dados, inviabilizando sua aplicação embarcada, enquanto nos modelos híbridos o algoritmo genético (*Genetic Algorithm*) tem uma resposta lenta devido a complexidade dos cálculos realizados e o algoritmo de otimização de partículas (*Particle Swarm Optimization*) tem baixa velocidade de convergência de seus resultados durante o processo iterativo.

Devido a estas preocupações as orientações dadas para o desenvolvimento futuro destes algoritmos são fazer uma modelagem precisa do modelo eletroquímico da bateria em relação a variação dinâmica de seus parâmetros, pesquisa e desenvolvimento de algoritmos para serem embarcados nos hardwares de BMS, controle para equilibrar o SOC das células dentro de um agrupamento de alta tensão, ser criterioso na escolha dos parâmetros do modelo da bateria e os hiperparâmetros de um modelo baseado em dados, tal como desenvolver outras técnicas de otimização para estes modelos e considerar a importância da aplicação real ao problema, já que a dinâmica real da aplicação não pode ser simulada no laboratório e, desta forma, a estimativa do valor do SOC pode ter uma série de incertezas. Sendo assim os autores deste trabalho acreditam que essas sugestões fariam uma grande contribuição ao contínuo desenvolvimento destes algoritmos no futuro.

Os modelos mais simples, como de Contagem de Coulomb (*Coulomb Counting*), não apresentam robustez quanto aos ruídos do sistema ou falhas na medição do sensor e normal-

mente vem sendo aplicados em conjunto com outros métodos. [Singh et al. 2020](#) modificaram um circuito equivalente fornecido no *Matlab* adicionando 3 pares resistor-capacitor (RC) em série com a resistência interna e utiliza a função “*Isqnonlin*” para otimizar os valores destes parâmetros. Para estimar o SOC foi feito uma combinação do método de contagem de Coulomb e tensão de circuito aberto, o resultado ainda é corrigido utilizando algoritmos baseados em ANFIS considerando o efeito da temperatura na variação do SOC.

Os resultados foram comparados em 3 situações diferentes, primeiro sem a adição dos pares RC, segundo com a adição dos pares RC e, por fim, com a correção do erro através do algoritmo ANFIS. O circuito com 3-RC envolve mais perdas devido a impedância, enquanto no outro modelo apenas as perdas devido a resistência interna são consideradas, mas os testes mostraram que o circuito com 3-RC possui resultados mais próximos dos dados reais simulados.

Após a otimização dos parâmetros com o algoritmo ANFIS, os resultados foram muito próximos dos dados reais com baixo desvio, verificando a utilidade do projeto proposto para sistemas em tempo real. O sistema completo foi validado em tempo real usando configuração de laboratório *hardware in-the-loop* e neste trabalho todo o sistema de bateria foi testado com 420V ao invés de uma única célula, obtendo resultados satisfatórios.

Para extrair os parâmetros que representam as reações eletroquímicas da bateria, o método mais tradicional entre as aplicações embarcadas é o ECM. Os modelos são continuamente desenvolvidos para caracterizar com maior precisão os parâmetros da bateria que irão servir como dados de entrada para a estimativa do SOC. Eles apresentam boa precisão nos cálculos e têm seu custo computacional compatível com aplicações reais embarcadas. No entanto, sua utilização é em grande maioria em conjunto com algoritmos de estimativa baseados em dados, como Redes Neurais Artificiais e Filtros de Kalman, que usam esses parâmetros estimados para calcular o SOC.

[Wang et al. 2016](#) apresentam um estimador adaptativo baseado em probabilidade para obter com precisão valores do SOC, através de um modelo de circuito equivalente resistor capacitor (RC) em combinação com um modelo eletroquímico e do estado de energia (*State of Energy* – SOE) via uma rede neural de janela deslizante (*Sliding Window Neural Network*) para obter a relação entre a tensão do terminal e as entradas do modelo. Uma bancada de testes foi estruturada e diferentes perfis dinâmicos de corrente foram testados para obter os resultados deste modelo.

O teste com o perfil de caracterização de potência de pulso híbrido (*Hybrid Pulse*

*Power Characterisct - HPPC) mostrou que o modelo baseado em dados e o modelo de rede neural obtiveram erros de tensão de  $\pm 10$  mV e  $\pm 20$  mV, indicando que uma previsão precisa pode ser fornecida por estes modelos. O teste de estresse dinâmico (*Dynamic Stress Test – DST*) mostrou que o erro do modelo proposto foi de  $\pm 8$  mV. O erro máximo absoluto estimado (*Maximum Absolute Estimated Error – MAEE*) e a raiz do erro quadrático médio (*Root Mean Square Error – RMSE*) para a estimativa do SOC com base no modelo proposto são de 0,97% e 0,53%, respectivamente, em contraste com 3,19% e 2,68% apresentados pelo FK extendido (*Extended Kalman Filter – EKF*). Para o cálculo do SOE o modelo proposto também foi superior com valores de MAEE e RMSE abaixo de 2%, enquanto o EKF apresentou resultados de 3,96% e 2,75%, respectivamente.*

Finalmente, um perfil UDDS foi executado e seus resultados mostraram que para a estimativa do SOC o modelo proposto apresentou o MAEE e o RMSE de 1,31% e 0,88%, respectivamente, enquanto os resultados com base no EKF foram de 3,62% e 3,05%, respectivamente. Para estimativa do SOE a partir do perfil UDDS o modelo proposto apresentou MAEE e RMSE inferiores a 2,5% e 2%, respectivamente, enquanto o modelo com base no EKF teve resultados de 4,11% e 3,38%.

[Eddahech et al. 2012](#) demonstram um modelo para uma célula com base em medições de Espectroscopia de Impedância Eletroquímica baseado em uma abordagem de ECM e em seguida uma RNA Recorrente (*Recurrent Neural Network*) para estimar o SOH da bateria. Os parâmetros são obtidos através de um modelo de 1<sup>a</sup> ordem seguido de duas RNAs para estimar o SOC. A função de ativação da rede neural é uma sigmóide com 7 neurônios na camada de entrada e 6 neurônios ocultos, baseados no algoritmo de retro-propagação e com erro quadrático médio (*Mean Square Root*) de 0,462 e 0,296 para valores normalizados de capacidade e resistência, respectivamente. Esses resultados mostram uma contribuição para o desenvolvimento de sistemas automatizados em tempo real para monitoramento dos parâmetros de saúde da bateria.

[Yang et al. 2017](#) utilizam um ECM de primeira ordem para estimar os parâmetros de entrada de uma RNA (*Back Propagation Neural Network - BPNN*), os parâmetros podem ser usados para treinar a BPNN de três camadas e a mesma é utilizada para estimar o SOH. Os testes com perfil de corrente estático e dinâmico verificaram que a precisão do modelo proposto é adequada para estimar o SOH com baixo custo de computação. Os resultados mostraram que a resistência ôhmica  $R_0$  aumenta quando reduzimos o SOH e que durante o intervalo de 20% a

90% do SOC,  $R_0$  aumenta conforme o SOC diminui. Os parâmetros  $R_p$  e  $C_p$  variam de forma não linear e não foi possível concluir com clareza uma relação direta. Para estimar o SOH foi utilizado 5 baterias para o treinamento da RNA e outras 5 de validação da estimativa, que obteve erro máximo de 7,2% e que poderia ser menor caso fossem utilizados mais dados de treinamento. Este modelo proposto apresentou vantagens como o baixo custo de computação, baixo custo de memória e fácil entendimento. A precisão poderia ser melhor, mas depende do algoritmo utilizado, quantidade de dados de treinamento e precisão do modelo ECM escolhido.

[Chi et al. 2013](#) utilizam os parâmetros extraídos de um ECM e algoritmos de Filtro de Partículas (FP) e um FK (*Unscented Kalman Filter - UKF*) para prever o SOC, tensão final de descarga (*End of Discharge Voltage - EODV*) e o tempo de vida restante (*Remaining Usefull Life - RUL*) da bateria durante o voo de um veículo elétrico não tripulado. O resultado mostrou uma melhoria considerável nas implementações baseadas em filtros de partículas e prognósticos do EODV, tanto em relação a precisão do modelo quanto a estabilidade durante o voo.

[Xing et al. 2014](#) desenvolveram um algoritmo de UKF que otimiza o erro do sistema em conjunto com um ECM de resistência interna simplificado para estimar o SOC de acordo com uma tabela OCV-SOC e um parâmetro que considera a influência da temperatura ambiente, fator que altera muito a precisão da estimativa do SOC, o diagrama de blocos do trabalho pode ser visto em (Figura 2.2).

Foram feitos dois testes dinâmicos, o DST e o de direção urbana federal (*Federal Urban Driving Schedule – FUDS*), para validar o desempenho do modelo proposto. Os resultados dentro da faixa de 25% a 85% do SOC indicaram um erro de raiz quadrada média menor que os métodos que não corrigem o modelo levando em consideração a temperatura ambiente, portanto o modelo proposto poderia ser efetivamente aplicado em sistemas de gerenciamento de baterias para veículos elétricos. Além disso, o trabalho também observou que o erro do SOC estimado no modelo sem correção para temperaturas em 10°C é maior que a 40°C, causado pela imprecisão da tabela OCV-SOC em baixa temperatura.

Por fim, os resultados foram satisfatórios com RMSE abaixo de 5% nos testes com temperaturas de 10°C e 40°C, ainda que para a temperatura de 0°C tenha sido observado RMSE de até 16% para SOC inicial de 75% para o modelo corrigido e 25% para o modelo sem correção de temperatura.

[Guo et al. 2019](#) realizaram uma série de testes em um ECM (Figura 2.3) para fornecer suporte de dados para a identificação dos parâmetros da bateria e em seguida aplica-

um FK estendido para estimar o SOC e utiliza uma técnica de Hardware-in-the-Loop (HIL) para validar o sistema. A simulação HIL e o teste em bancada mostraram um erro de estimativa do SOC do BMS menor que 5% no caso de instabilidade na tensão de circuito aberto da bateria, com uma boa robustez e confiabilidade.

[Fang et al. 2019](#) propõem um ECM com um fator de esquecimento recursivo dos mínimos quadrados (*Forgetting Factor Recursive Least Squares*) para realizar a identificação dos parâmetros do modelo e desenvolve um estimador de conjunto utilizando um FK duplo (*Double Extended Kalman Filter*) para estimar o SOC e o SOH. O resultado dos experimentos realizados mostrou que o erro máximo da estimativa de SOC é de 1,08% e do SOH é de 1,52%, além de que a curva OCV-SOC tem a maior influência no erro de estimativa em comparação aos valores da capacidade da bateria e seus parâmetros. Como um sistema de bateria consiste em uma dinâmica rápida e outra lenta, o método clássico dos mínimos quadrados (*Least Square – LS*), que estima todos os parâmetros do modelo, sofre com problemas e baixa precisão.

[Zhang et al. 2018](#) sugerem um novo método de LS (*Decoupled Weighted Recursive Least Squares*) que estima separadamente os parâmetros de dinâmica rápida e lenta da bateria. Uma simulação foi feita para comparar os resultados do modelo proposto contra a técnica LS e os resultados mostraram que o método proposto pode melhorar não apenas a precisão da modelagem, mas também o desempenho de estimativa do SOC, já que apresentou RMSE de 0,86% contra 2,98% do tradicional LS.

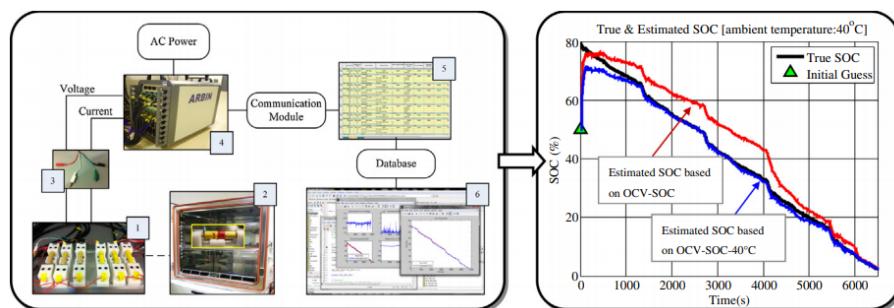


Figura 2.2 – Figura extraída do artigo [Xing et al. 2014](#)

Os métodos orientados a dados vêm sendo amplamente desenvolvidos nos trabalhos de pesquisa, as RNA e os algoritmos de filtragem como o FK e o FP são os mais populares devido à sua precisão, capacidade de adaptação e treinamento aos diversos casos de aplicação do problema ao estimar os parâmetros da bateria. [Nunes et al. 2018](#) utilizaram uma RNA para estimar o SOC da bateria em duas partes, primeiro estimando a tensão a partir dos dados de corrente e temperatura e depois estimando o SOC a partir dos dados de tensão, corrente,

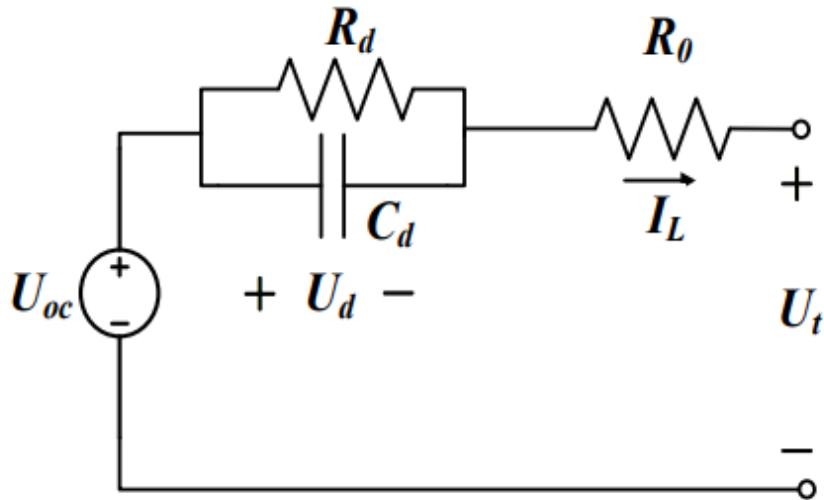


Figura 2.3 – Figura extraída do artigo [Guo et al. 2019](#)

temperatura e realimentação do valor do SOC. A RNA foi desenvolvida com capacidade de aprendizagem e adaptação aos ciclos de carga, descarga e descanso de uma bateria de íons de Lítio, se mostrando muito precisa nos testes realizados.

[Zahid e Li 2016](#) compararam algoritmos de filtragem como o EKF, UKF e o FP para estimar o SOC de baterias de LiFePO<sub>4</sub> para um veículo elétrico. Os parâmetros da bateria são identificados a partir de um modelo equivalente de Thevenin. Os resultados mostraram que quanto maior a ordem da função polinomial, melhores resultados são obtidos na estimativa. Durante os testes percebeu-se que todos os métodos utilizados, tiveram erros grandes quando o SOC estava entre 45% a 65% e que, quando um SOC incorreto era dado inicialmente, o FP era mais afetado em relação aos demais, demorando mais para recuperar o erro de inicialização. O UKF provou-se melhor que o EKF, embora também tenha sofrido com a recuperação do erro de inicialização e o FP superou o EKF quando o modelo da bateria foi afetado por um ruído gaussiano ou quando um modelo de ordem superior foi utilizado, neste último caso superando até mesmo o UKF. Estes resultados mostraram que ambos os métodos podem ser aplicados e cada um tem características melhores que o outro dependendo do problema de aplicação.

[Zhao et al. 2019](#) basearam seu trabalho em modelos de aprendizagem profunda (*Deep Learning*) e revisa sistemas como autoencoders (*Autoencoders*), máquina Boltzmann restrita (*Restricted Boltzmann Machine*), uma classe de RNA (*Deep Belief Network - DBN*), outra classe da máquina Boltzmann (*Deep Boltzmann Machine*), RNA Convolucional (*Convolutional Neural Network*), RNA Recorrente (*Recurrent Neural Network*) e novas tendências de monitoramento. O trabalho conclui que o aprendizado profundo aplicado em sistemas de moni-

toramento de saúde da máquina (*Machine Health Monitoring Systems*) depende fortemente da quantidade e qualidade dos dados utilizados e a extração de parâmetros dos modelos monitorados pode reduzir a complexidade das camadas ocultas.

[Hannan et al. 2018](#) tiveram como objetivo melhorar a capacidade de uma BPNN utilizando um algoritmo de busca (*Backtracking Search Algorithm - BSA*) a fim de melhorar a precisão e robustez do modelo da RNA, encontrando o melhor valor de neurônios da camada oculta, taxa de aprendizado, dentre outros (Figura 2.4). Para testar o modelo em diferentes temperaturas foi utilizado um perfil DST, um FUDS e os resultados obtidos mostraram que o modelo BSA baseado em BPNN supera os outros modelos de rede na estimativa de SOC com alta precisão em diferentes perfis e temperaturas. Observou-se também, que conforme a temperatura aumenta, o erro da estimativa do SOC diminui e a faixa de erro a 45°C do modelo BPNN está entre -3,5% e 4,3% para o ciclo FUDS e -2,4% e 3,3% para o ciclo DST. Já para o modelo BPNN-BSA, o erro na estimativa é de -2,4% e 3,5% e para o ciclo FUDS está entre -2,1% e 3,2%, já para valores de temperatura mais baixos, como 0 °C o erro chega até quase 10%.

[Zhang et al. 2019](#) treinaram um modelo de mínimos quadrados e máquina de vetores de suporte (*Least-Squares Support Vector Machine*) com um pequeno conjunto de amostras para determinar as características dinâmicas da bateria e utilizou um UKF baseado no modelo para estimar o SOC. Os resultados da simulação com os dados de teste HPPC e FUDS confirmam que a abordagem proposta pode produzir um modelo mais simplificado e ainda mais preciso. O erro absoluto da estimativa do SOC após o UKF nos testes realizados varia de -0,02% a 0,06% quando o SOC é entre 10% e 90% para o teste HPPC, da mesma forma o erro varia entre -0,01% a 0,02% para o teste FUDS.

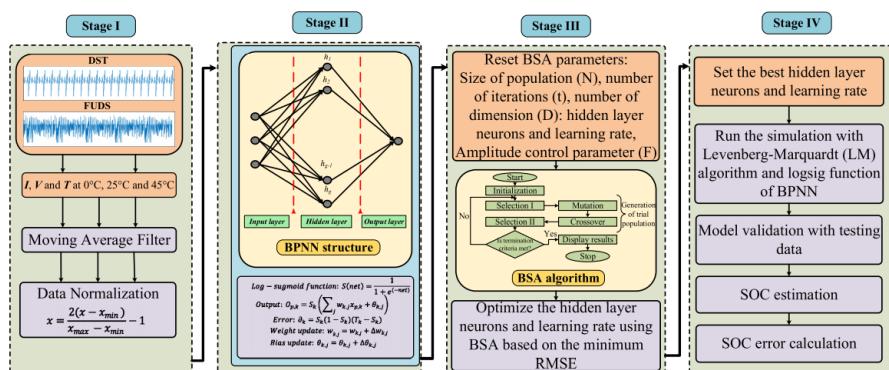


Figura 2.4 – Figura extraída do artigo [Hannan et al. 2018](#)

Os EM são os métodos mais precisos para determinar os parâmetros da bateria pois

em sua maioria se baseiam em Equações Diferenciais Parciais (EDPs) para descrever o comportamento dinâmico da bateria. Apesar de serem muito precisos os cálculos de EDPs podem ter alto custo computacional, o que acaba limitando sua aplicação em sistemas embarcados. [Li et al. 2015](#) apresentaram um modelo tridimensional que abrange a conservação de carga, massa, energia e a reação eletroquímica de uma bateria de Fosfato de Ferro e Lítio, investigando as propriedades eletroquímicas internas. Os resultados indicaram que os gradientes máximos das propriedades estão na região de transição entre as guias e as placas de eletrodo. Para a bateria de LiFePO<sub>4</sub>/Grafite o ânodo desempenha um papel mais importante que o cátodo no potencial e é provável que seja crucial na diminuição acentuada da tensão de saída no processo de descarga da célula, além de que a quantidade de material do ânodo pode fornecer uma maior capacidade para a bateria. Este modelo forneceu um eficiente método para observar as informações detalhadas sobre a distribuição espacial e temporal dos comportamentos eletroquímicos internos em uma bateria de íons de Lítio, como distribuição de potencial elétrico transitório, distribuição de sobre potencial, distribuição do SOC, local de distribuição de densidade de corrente e distribuição de concentração de íons de lítio, que são difíceis de se obter por experimentos de métodos tradicionais.

[Hosseinzadeh et al. 2017](#) apresentaram um EM térmico unidimensional de um par de eletrodos de uma bateria de íons de Lítio desenvolvido no software *Comsol Multiphysics* e validado para uma célula de fosfato de Lítio para realizar análises estatísticas dos parâmetros mais influentes que determinam o desempenho da célula. Os parâmetros analisados foram: raio de partícula ( $rp$ ), espessura do eletrodo ( $L_{pos}$ ), fração de volume do material ativo ( $\epsilon_{s,pos}$ ), taxa C, energia específica e potência específica. Sendo assim, observou-se que a energia ótima pode ser alcançada quando  $rp \leq 40 \text{ nm}$ ,  $75 \mu\text{m} \leq L_{pos} \leq 100 \mu\text{m}$ ,  $0,4 \leq \epsilon_{s,pos} \leq 0,6$  e enquanto a taxa C estiver abaixo de 4C. Por outro lado, a potência ótima é obtida para um eletrodo fino quando  $L_{pos} \leq 30 \mu\text{m}$ , alta porosidade e taxa C. Como não podemos alcançar o ideal de energia e potência ao mesmo tempo, a bateria deve ser projetada de modo que a relação potência/energia seja satisfatória para a aplicação específica.

[Chen et al. 2019](#) mostraram um EM simplificado de eletrodo que adota uma aproximação polinomial e um método de três variáveis, propondo um novo método de identificação de parâmetros considerando temperatura e corrente para reduzir o desvio causado por diferentes condições. Os parâmetros do modelo são identificados pelo algoritmo genético offline em diferentes temperaturas e correntes para criar tabelas de pesquisa para estimativas online. As

células escolhidas para validação do modelo simplificado e método de estimativa proposto são do tipo 18650 3,5 Ah NCM. Os resultados indicam que o esquema proposto é preciso, simples e flexível para mudanças de corrente e temperatura para diferentes condições de operação. O modelo simplificado combinado com o método de identificação de parâmetros proposto atualiza os parâmetros com um erro de tensão de aproximadamente 0,056V, erro de tensão relativo entre 0 a 0,7% e uma média do erro de tensão relativo de 0,2%.

He *et al.* 2018 propuseram um modelo eletroquímico para estimativa de SOC de bateria de íons de Lítio envolvendo as propriedades internas físicas e químicas da bateria como, por exemplo, as concentrações de lítio (Figura 2.5). Para resolver computacionalmente as complexas PDEs de difusão em fase sólida no modelo, um método eficiente com base na projeção com funções de base otimizadas é apresentado. Então, um algoritmo de filtragem de janela móvel (*Moving Window Filtering*) é desenvolvido para melhorar a taxa de convergência dos filtros de estado. Os resultados mostraram que o modelo eletroquímico desenvolvido gera 20 vezes menos equações em comparação com métodos baseados em diferenças finitas, sem perder a precisão. Além disso, a proposta baseada em projeções do método de solução é três vezes mais eficiente do que os métodos convencionais de filtragem de estado, como o FK.

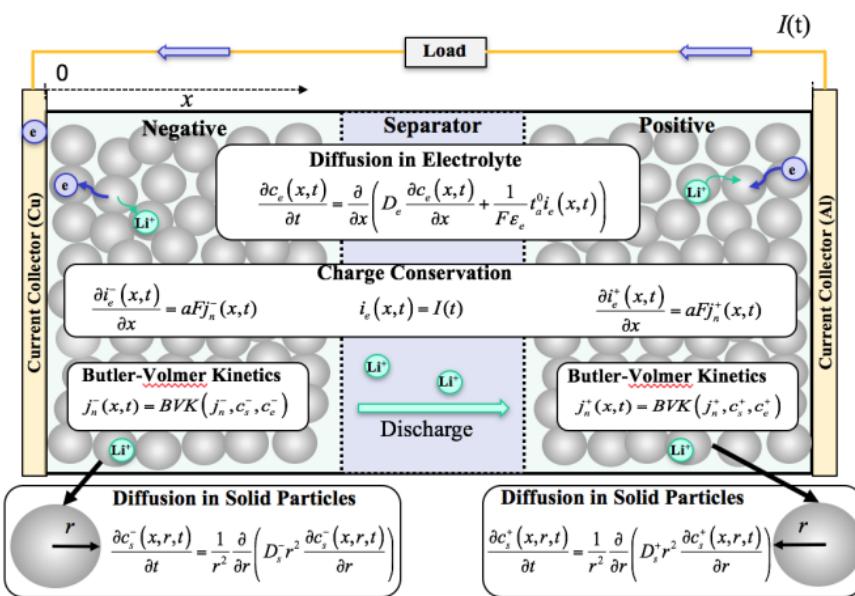


Figura 2.5 – Figura extraída do artigo He *et al.* 2018

Os métodos híbridos, contendo 2 ou mais sistemas integrados para melhorar o desempenho da estimativa e controle dos parâmetros da bateria, vem sendo cada vez mais presentes nas pesquisas e aplicações para muitos tipos de problema, como estimar os parâmetros de saúde de uma bateria. Conseguir absorver as qualidades de cada método, criando um mo-

delo integrado que tenha as melhores características de cada um tem sido uma boa estratégia de aplicação e gerando bons resultados, sendo apontada como maior potencial de desempenho em projetos futuros. Combinar a precisão de EM baseados em equações diferenciais com a otimização dos cálculos computacionais dos métodos orientados a dados, por exemplo, pode ser um caminho para solucionar muitos problemas embarcados em tempo real, que muitas vezes requerem maior precisão e adaptação a diferentes situações que as oferecidas por modelos orientados somente por dados.

[Tulsky et al. 2016](#) demonstraram um algoritmo original para estimativa do SOC usando um modelo pseudo bi-dimensional (*Pseudo-Two-Dimensional* – P2D). As equações diferenciais parciais são discretizadas usando algoritmos em um modelo de espaço de estado não linear. Este modelo discreto de alta dimensão, consistindo de dezenas a centenas de estados, contém equações algébricas não lineares implícitas. A incerteza no modelo é caracterizada por ruído gaussiano aditivo. Explorando a estrutura especial do modelo pseudo-bidimensional, um novo algoritmo de FP que varre no tempo e nas coordenadas espaciais de forma independente, foi desenvolvido. O SOC e outras propriedades da bateria que dependem de variáveis de estado não medidas, como as concentrações e potenciais são estimados através do FP desenvolvido. O algoritmo usa uma nova técnica para reduzir a complexidade computacional e uma simulação mostra que o algoritmo proposto fornece estimativa precisa do SOC na presença de estado significativo e ruído de medição. O modelo P2D é um dos modelos mais sofisticados, no entanto, não leva em conta a saúde da bateria, pois ela muda com a carga e ciclos de descarga. O trabalho conclui que o algoritmo proposto pode ser facilmente adaptado para integrar as equações do estado de saúde no modelo P2D.

[Farfán et al. 2020](#) abordaram o problema através de uma aplicação diferente que o da obtenção dos parâmetros de saúde da bateria, mas conceitualmente a ideia segue o mesmo caminho e é muito importante para o desenvolvimento dos modelos híbridos. Foi avaliado a precisão de dois modelos físicos, dois modelos baseados em RNA e proposto um sistema híbrido usando as séries temporais geradas pelos modelos individuais como entrada de uma nova RNA. Os resultado indicaram que a técnica híbrida, utilizando equações e dados, foi capaz de melhorar o desempenho individual dos modelos físicos e baseados apenas em RNA.

[Karpatne et al. 2017](#) apresentaram uma nova estrutura para combinar o conhecimento científico de modelos baseados em física com redes neurais (*Physics-guided Neural Network*), utilizando a saída do modelo baseado em física junto com recursos observacionais

para gerar previsões usando uma arquitetura de RNA e aplicando o método para um problema de modelagem da temperatura de um lago. Além disso, este artigo apresenta uma nova estrutura para o uso de funções de perda baseadas na física para o aprendizado de redes neurais, garantindo que as previsões do modelo não apenas mostrem erros menores no conjunto de treinamento, mas também sejam cientificamente consistentes com a física conhecida do problema em questão. Através do conhecimento científico para guiar a construção e aprendizagem de redes neurais, a estrutura proposta garantiu melhor generalização e consistência científica dos resultados apresentados. Este resultado nos mostra que estes algoritmos podem possuir tanto as vantagens dos métodos baseados nas características físicas quanto nos modelos baseados em dados, mantendo a eficácia do sistema mesmo para aplicações onde os dados reais são muito divergentes dos utilizados durante o treinamento.

[Kharazmi et al. 2019](#) desenvolveram uma versão de uma RNA baseada na física (*Physics Informed Neural Networks - PINNs*) de Petrov-Galerkin com base na aproximação linear de uma RNA profunda (*Deep Neural Network*) selecionando o espaço para as redes neurais e polinômio de Legendre. O residual variacional das PDEs é formulado usando a aproximação de DNNs, incorporando a forma variacional do problema na função de perda da rede neural e construindo uma forma variacional informada pela física da rede neural (*Variational Physics-Informed Neural Network - VPINN*) (Figura 2.6). Desta forma, a ordem dos operadores diferenciais representados pelas redes neurais diminuiu, reduzindo efetivamente o custo de treinamento em VPINNs e aumentando sua precisão quando comparada a PINNs. Foi demonstrado vários exemplos onde foi observado claras vantagens das VPINNs sobre as PIINs em questão de precisão da estimativa e velocidade de desempenho.

[Raissi et al. 2017](#) introduziram o conceito de redes neurais informadas pela Física, que são treinadas para resolver tarefas de aprendizado supervisionado respeitando uma lei da Física descrita por PDEs não lineares. Foi apresentado dois desenvolvimentos, uma solução orientada a dados e uma descoberta orientada a dados de PDEs. As redes neurais resultantes vão formar uma nova classe de aproximadores de função universal eficientes em dados que codificam naturalmente quaisquer leis Físicas. Essas redes podem ser utilizadas para inferir soluções para PDEs e obter modelos substitutos informados pela Física totalmente diferenciáveis.

Para ser possível o desenvolvimento de novos modelos no campo de pesquisa é necessário a obtenção de dados de teste em ambientes controlados para utilização desses dados no desenvolvimento do algoritmo. Para isso, é necessário equipamentos especiais e de alto custo

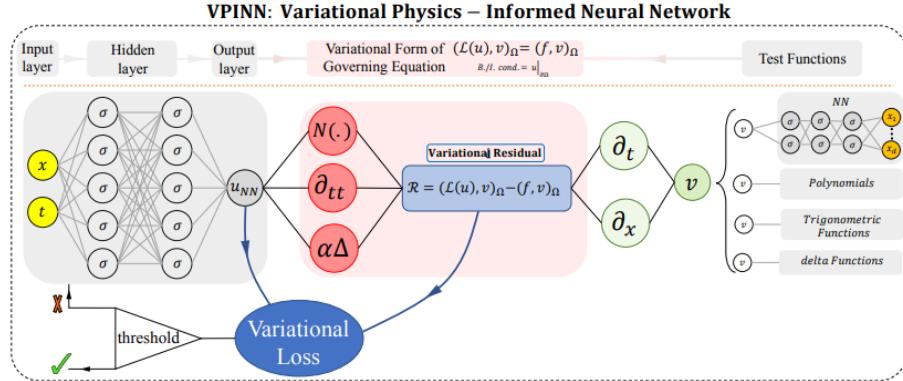


Figura 2.6 – Figura extraída do artigo [Kharazmi et al. 2019](#)

para poder testar as células em condições programadas e perfis de carga e descarga desejados. Este fator poderia impossibilitar muitos pesquisadores a evoluírem seus modelos devido ao fato de que este tipo de laboratório não é de acesso fácil para a grande maioria dos desenvolvedores. Sendo assim, algumas instituições importantes disponibilizam banco de dados de simulações de diferentes células de bateria sob diferentes perfis de descarga para fins educativos e de pesquisa, como listado no trabalho [How et al. 2019](#). Este presente trabalho utilizou os dados disponibilizados por *Gregory L. Plett da University of Colorado Colorado Springs* (UCCS), mas existem dados semelhantes disponibilizados pela NASA e pelo *Center for Advanced Life Cycle Engineering* (CALCE), dentre outros, que podem ser adquiridos de forma gratuita em suas plataformas na internet para fins de estudo e desenvolvimento.

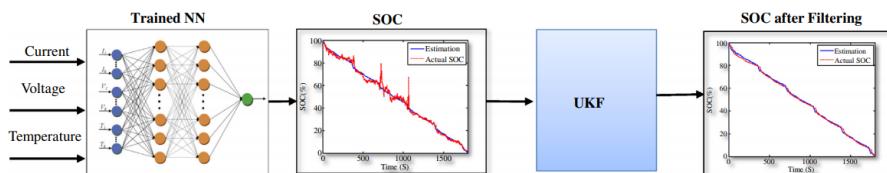


Figura 2.7 – Figura extraída do artigo [He et al. 2014](#)

Todos os trabalhos referenciados auxiliaram na construção deste presente trabalho para direcionar a metodologia e quais as tendências nas quais acredita-se serem positivas para contribuir no desenvolvimento de tecnologia.

### 3 METODOLOGIA

Neste capítulo está descrito todos os passos para obtenção da estimativa do SOC pelo algoritmo híbrido proposto neste trabalho. Iniciou-se os testes laboratoriais nas células de bateria para obtenção dos parâmetros internos, posteriormente o desenvolvimento do KF que realiza a primeira estimativa do SOC, até a obtenção dos KPIs e desenvolvimento da RNA para realizar a segunda estimativa do SOC. Para obtenção dos resultados foi utilizado um banco de dados com testes em uma bateria A123 de LiFePO<sub>4</sub> de 3,6V com capacidade de 2230 mAh, pois é o mesmo modelo utilizado em diversos veículos elétricos e o mais próximo a nossa aplicação fornecido para fins educativos e de pesquisa através da CALCE.

Para a obtenção dos parâmetros internos, a célula A123 passa por 2 testes laboratoriais, o Teste de Circuito Aberto (*Open Circuit Voltage - OCV*) e o teste com perfil dinâmico de carga e descarga (*Dynamic Urban Dynamometer Driving Schedule - DYN*).

#### 3.1 Teste 1: Tensão de Circuito Aberto

O objetivo foi encontrar a relação do OCV com o SOC, tendo em vista que eles possuem uma relação direta dependente da temperatura e essa é necessária para calcular os parâmetros dinâmicos da célula no próximo teste.

Para obtenção dos dados de corrente e tensão, a conexão com a célula é feita a partir de 4 fios, separando a medição da corrente com a tensão, para que não exista influência de uma queda de tensão no cabo de medição devido a corrente, caso o sistema fosse ligado apenas por 2 fios.

O equipamento de ensaio utilizado em laboratório, assim como os equipamentos da *Arbin Instruments* nos EUA ou da CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) no Brasil, tem como princípio de funcionamento aplicar diversos ciclos de carga/descarga definidos pelo usuário, sendo aplicável a várias células simultaneamente, sob uma temperatura específica e controlada, gerando como resposta os valores de tensão, corrente e temperatura em um banco disponibilizado após o teste. Parâmetros de temperatura e umidade são controlados de acordo com as definições de teste, para obtenção dos dados em uma faixa de temperatura, bem como aplicar choques térmicos na célula, se desejado.

O teste foi executado em 10 passos:

Descarregar a célula até a tensão mínima em temperatura de trabalho .

1. Inserir célula no ambiente de teste com a temperatura de trabalho especificada e permanecer ao menos 2 horas para que toda célula atinja temperatura uniforme;
2. Aplicar corrente de descarga constante até que a tensão medida seja igual a mínima estabelecida;

Descarregar a célula até atingir SOC = 0% à 25°C.

3. Mudança de temperatura no ambiente de teste para 25°C e manter célula em repouso ao menos 2 horas, para que ela atinja temperatura uniforme;
4. Aplicar corrente de descarga até que a tensão medida seja igual a mínima estabelecida;
5. Aplicar um perfil de corrente oscilatório utilizado em desmagnetização para minimizar o efeito da histerese;

Carregar a célula até a tensão máxima em temperatura de trabalho.

6. Mudança de temperatura no ambiente de teste para a de trabalho especificada, e manter célula em repouso ao menos 2 horas para que ela atinja temperatura uniforme;
7. Aplicar corrente de carga até que a tensão medida seja igual a máxima estabelecida;

Carregar a célula até atingir SOC = 100% à 25°C.

8. Mudança de temperatura no ambiente de teste para 25°C e manter célula em repouso ao menos 2 horas, para que ela atinja temperatura uniforme;
9. Aplicar corrente de carga até que a tensão medida seja igual a máxima estabelecida;
10. Aplicar um perfil de corrente oscilatório utilizado em desmagnetização para minimizar o efeito da histerese.

Os gráficos gerados neste teste de tensão de circuito aberto para uma célula de bateria de íons de Lítio A123, são apresentados na sessão de Resultados 4.1.

### 3.2 Teste 2: Perfil Dinâmico de Carga e Descarga

O objetivo foi aplicar um perfil de carga/descarga semelhante a aplicação real para gerar os dados utilizados como entrada do algoritmo, que neste trabalho extraiu os parâmetros do modelo ESC da bateria. O perfil de direção urbana (*Urban Dynamometer Driving Schedule* - UDDS) foi escolhido por ser o perfil disponível mais próximo de uma aplicação em *Motorsports*. O ambiente de teste seguiu o mesmo padrão do primeiro, sendo a única diferença entre eles os passos e padrões de corrente que foram aplicados à bateria.

O teste foi executado em 9 passos:

1. Inserir célula no ambiente de teste com a temperatura de trabalho especificada e permanecer ao menos 2 horas, para que ela atinja temperatura uniforme;
2. Aplicar corrente de descarga a uma taxa de C/1 por 6 minutos para evitar uma sobrecarga nos próximos passos;
3. Executar o perfil UDDS no intervalo de 90% a 10% de SOC;
4. Mudança de temperatura no ambiente de teste para 25°C e manter célula em repouso ao menos 2 horas, para que ela atinja temperatura uniforme;

Descarregar a célula até atingir SOC = 0% à 25°C.

5. Aplicar corrente de descarga a uma taxa de C/30, até que a tensão medida seja igual a mínima estabelecida;

6. Aplicar um perfil de corrente oscilatório utilizado em desmagnetização para minimizar o efeito da histerese;

Carregar a célula até atingir SOC = 100% à 25°C.

7. Aplicar corrente de carga a uma taxa de C/30, até que a tensão medida seja igual a máxima estabelecida;

8. Manter a tensão constante até que a corrente caia abaixo de C/30;

9. Aplicar um perfil de corrente oscilatório utilizado em desmagnetização para minimizar o efeito da histerese.

Os gráficos gerados neste teste de perfil dinâmico de carga e descarga para uma célula de bateria de íons de Lítio A123, são apresentados na sessão de Resultados 4.2.

### 3.3 Algoritmo da Tensão de Circuito Aberto

Para obter a relação de OCV e SOC da célula a partir do teste laboratorial, foi utilizado um código desenvolvido no software *MatLab* baseado no modelo de *Grogory L. Plett, 2015*.

A fórmula para calcular o SOC é dada por:

$$z[k] = z[0] - \frac{1}{Q} \sum_{j=0}^{k-1} \eta[j] i[j]. \quad (3.1)$$

Para calcular a eficiência de Coulomb:

$$\eta = \frac{I_{discharged}}{I_{charged}}. \quad (3.2)$$

Onde  $I_{discharged}$  é a corrente consumida da bateria e  $I_{charged}$  é a corrente que foi utilizada para recarregar a bateria. Para calcular a capacidade total da célula foi utilizado:

$$Q = \sum_{j=0}^{k-1} \eta[j] i[j]. \quad (3.3)$$

A porcentagem de carga retirada da bateria durante a descarga (*Depth of Discharge* - DoD) em cada ponto é calculado por:

$$DoD[t] = (TotalAh I_{discharged}) - \eta * (TotalAh I_{charged}). \quad (3.4)$$

Finalmente, com a equação 3.1 e 3.4 foi possível definir o cálculo do SOC a cada ponto por:

$$SOC[t] = 1 - \frac{DoD(t)}{Q}. \quad (3.5)$$

A representação do processo de obtenção da relação OCV e SOC pelo fluxograma está ilustrada a seguir (Figura 3.1):

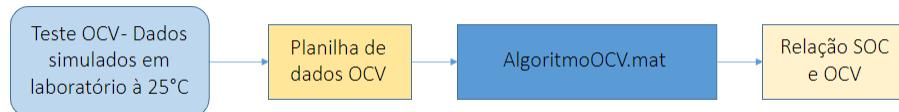


Figura 3.1 – Fluxograma Geral - Incluindo relação OCV-SOC

Obtendo os valores compensados de SOC e OCV foi possível estabelecer uma relação entre eles, sendo utilizada como dado de entrada para o algoritmo que calculou os parâmetros dinâmicos a partir do perfil UDDS. Resumidamente, o código calcula primeiro a  $\eta$ , ajusta as curvas de carga e descarga compensando o valor da resistência interna  $R_0$  e compila as relações de SOC e OCV, para a temperatura de trabalho escolhida de 25°C.

O algoritmo utilizado neste projeto, referente a essa metodologia apresentada, encontra-se ilustrado no anexo A.1, e os resultados gerados compilando o programa podem ser vistos na sessão 4.3.

### 3.4 Algoritmo do Teste Dinâmico

O Teste 2 forneceu os dados de entrada do perfil UDDS para o código desenvolvido que determinou os parâmetros dinâmicos do modelo ESC da célula.

O processo de obtenção dos parâmetros dinâmicos da célula A123 está representado pelo fluxograma a seguir (Figura 3.2):

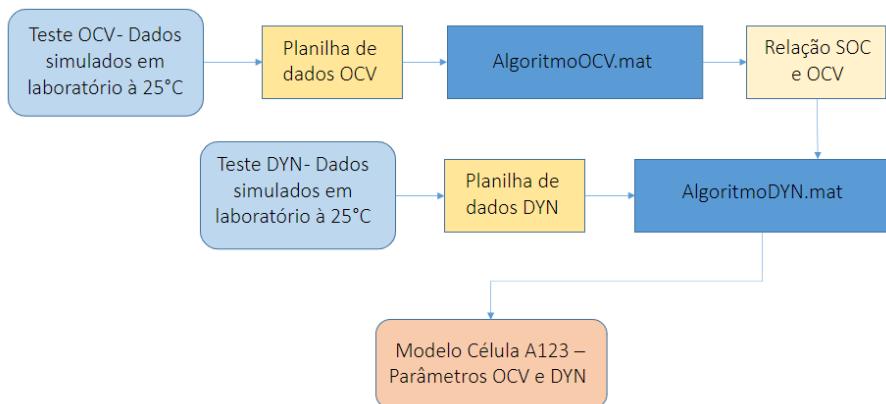


Figura 3.2 – Fluxograma Geral - Incluindo Modelo A123

Os dados de entrada são  $i[k]$ ,  $v[k]$  e a tabela  $OCV$  vs  $SOC$  (obtidos no algoritmo de OCV). Novamente são calculados  $\eta$  e  $Q$  através das equações 3.2 e 3.3 em seguida obtido  $z[k]$  a partir da equação 3.1 com  $z[0]=0$ , e calculada a tensão ajustada  $v_{ajus}[k]$  através da equação:

$$v_{ajus}[k] = v[k] - OCV(z[k], T[k]) \quad (3.6)$$

Em que  $v[k]$  é a tensão medida e  $OCV(z[k], T[k])$  é a relação de OCV e SOC calculada no algoritmo OCV.

Para calcular o parâmetro  $R - C$ , o algoritmo utilizou uma técnica de identificação de sistema de subespaço e calculou as equações de estado  $i_R[k]$ ,  $s[k]$  e  $h[k]$  através das equações 3.7, 3.8 e 3.9:

$$i_R[k] = A_{AR} i_R[k - 1] + B_{RC} i_R[k - 1], \quad (3.7)$$

$$s[k] = \text{sgn}(i[k]), |i[k]| > 0, \quad (3.8)$$

$$h[k] = A_H h[k - 1] + (A_H h[k - 1] - 1)\text{sgn}(i[k - 1]). \quad (3.9)$$

O parâmetro de saída,  $v_{est}[k]$ , foi obtido através da equação:

$$v_{est}[k] = OCV(z[k], T[k]) + Mh[k] + Mv_0s[k] - \sum_j R_j i_{Rj}[k] - R_0 i[k]. \quad (3.10)$$

Para calcular o desvio do parâmetro estimado foi utilizado a raiz do erro quadrático médio:

$$rms = \sqrt{\frac{1}{N} \sum_{j=1}^n (v[j] - v_{est}[j])^2}. \quad (3.11)$$

Todos os parâmetros dinâmicos do modelo ESC calculados foram otimizados e atualizados até a obtenção do menor erro quadrático médio da raiz. Por fim, o código retornou uma matriz contendo os dados:

1. QParam - Capacidade  $Q$  em Ah
2. etaParam - Eficiência de Coulomb  $\eta$
3. GParam - Parâmetro  $\gamma$  de Histerese
4. MPParam - Parâmetro M de Histerese[V]
5. M0Param - Parâmetro  $M_0$  de Histerese [V]
6. R0Param - Resistência Interna [Ohm]
7. RCPParam - Constante de tempo  $R - C$  [s]

## 8. RParam - Resistência do parâmetro $R - C$ [Ohm]

O algoritmo utilizado neste projeto, referente a essa metodologia apresentada, encontra-se no anexo A.2, e os resultados gerados compilando o programa encontram-se na sessão 4.4.

### 3.5 Filtro de Kalman Ponto Sigma

Após a obtenção dos dados do modelo ESC, foi implementado um Filtro de Kalman Ponto Sigma (*Sigma Point Kalman Filter - SPKF*) para estimar o SOC da célula A123 durante a aplicação de um perfil UDDS de corrente de carga/descarga, assim como na simulação com perfil dinâmico de carga/descarga. A resposta deste sistema serviu como dado de entrada para a RNA. Os códigos utilizados foram baseados pelo modelo de *Grogory L. Plett, 2015*.

O diagrama de blocos do processo incluindo a obtenção do SOC pelo SPKF está representado a seguir (Figura 3.3):

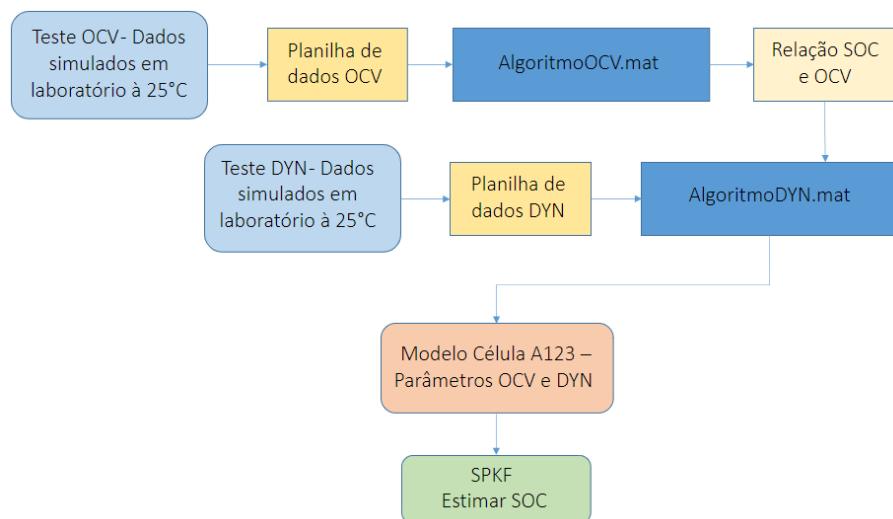


Figura 3.3 – Fluxograma Geral - Incluindo SPKF

A metodologia abordada sobre *SPKF* foi retirada do conteúdo do livro [Plett 2015](#). Para aplicar a abordagem de *Sigma Point* da estatística de propagação, através de uma função não linear para o problema de estimativa de estado, a abordagem deve modelar conjuntamente toda aleatoriedade de incerteza do estado, do ruído do processo e do ruído do sensor. O processo é dividido em 2 passos, de predição/estimativa e correção, e cada passo é subdividido em 3 partes. De forma genérica, o EKF e o SPKF seguem os mesmos passos do KF.

Passo 1a: Atualização da Estimativa de Estado

$$x_k^- = E[x_k | Y_{k-1}] = E[f(x_{k-1}, u_{k-1}, w_{k-1}) | Y_{k-1}] \quad (3.12)$$

Implementado computacionalmente nos seguintes passos:

$$x_k^- = E[f(x_{k-1}, u_{k-1}, w_{k-1}) | Y_{k-1}] \quad (3.13)$$

$$x_k^- = \sum_{i=0}^p \alpha_i^{(m)} f(\chi_{k-1,i}^{x,+}, u_{k-1}, \chi_{k-1,i}^{w,+}) \quad (3.14)$$

$$x_k^- = \sum_{i=0}^p \alpha_i^{(m)} \chi_{k,i}^{x,-} \quad (3.15)$$

E, finalmente, foi possível calcular a estimativa do estado através de uma simples multiplicação de matriz:

$$x_k^- = [\chi_k^{x,-}] [\alpha^{(m)}] \quad (3.16)$$

Passo 1b: Atualização do Erro de Covariância

$$\bar{\sum}_{x,k} = E[(x_k^-)(x_k^-)^T] = E[(x_k - x_k^-)(x_k - x_k^-)^T] \quad (3.17)$$

Implementada computacionalmente nos seguintes passos:

$$\bar{\sum}_{x,k} = \sum_{i=0}^p \alpha_i^{(c)} (\chi_{k,1}^{x,-} - x_k^-) (\chi_{k,i}^{x,-} - x_k^-)^T \quad (3.18)$$

A covariância estimada foi calculada por:

$$\bar{\sum}_{x,k} = [\chi_k^{x,-} - x_k^-] [diag(\alpha^{(c)})] [\chi_k^{x,-} - x_k^-]^T \quad (3.19)$$

Passo 1c: Estimativa da Saída do Sistema

$$y_k = E[y_k | Y_{k-1}] = E[h(x_k, u_k, v_k) | Y_{k-1}] \quad (3.20)$$

Implementada computacionalmente nos seguintes passos:

$$y_k = E[h(x_k, u_k, v_k) | Y_{k-1}] \quad (3.21)$$

$$y_k = \sum_{i=0}^p \alpha^{(m)} h(\chi_{k,i}^{x,-}, u_k, \chi_{k-1,i}^{v,+}) \quad (3.22)$$

A saída foi estimada novamente por uma simples multiplicação de matriz:

$$y_k = \sum_{i=0}^p \alpha^{(m)} Y_{k,i} \quad (3.23)$$

O passo 2 representa o processo de correção de estados e também é subdividido em 3 partes.

Passo 2a: Estimar a matriz de ganho

$$L_k = \sum_{xy,k}^- \sum_{y,k}^{-1} \quad (3.24)$$

Onde:

$$\sum_{y,k}^- = \sum_{i=0}^p \alpha_i^{(c)} (Y_{k,i} - y_k) (Y_{k,i} - y_k)^T \quad (3.25)$$

$$\sum_{xy,k}^- = \sum_{i=0}^p \alpha_i^{(c)} (\chi_{k,i}^{x,-} - x_k^-) (Y_{k,i} - y_k)^T \quad (3.26)$$

Passo 2b: Atualização da estimativa de estado medido

$$x_k^+ = x_k^- + L_k (y_k - y_k) \quad (3.27)$$

Passo 2c: Atualização do erro de covariância medido

$$\sum_{x,k}^+ = \sum_{x,k}^- - L_k \sum_{y,k} L_k^T \quad (3.28)$$

Através dos 6 passos o SPKF utilizou o método *Sigma-Point* para propagar a incerteza da entrada para a saída do estado e as equações de saída. A utilização dos vetores e matrizes é uma prática computacional adequada para armazenar e realizar os cálculos com os *Sigma-Points*, calculando as médias e as covariâncias. Desta forma, o modelo SPKF descrito acima foi derivado para ser aplicado em um algoritmo desenvolvido em *MatLab*.

O algoritmo utilizado neste projeto, referente a essa metodologia apresentada, encontra-se na sessão de anexos A.3, e os resultados gerados compilando o programa podem ser vistos na sessão 4.5.

### 3.6 Indicadores de Performance do Piloto

Os KPIs foram utilizados para correção na estimativa adquirida pelo SPKF, sendo incluídas características de pilotagem que influenciam diretamente no consumo de carga da

bateria. A depender do piloto, essas características também se diferem, conforme ilustrado nas Figuras 3.4 e 3.5, com dados de dois pilotos diferentes, nas cores azul e vermelho.

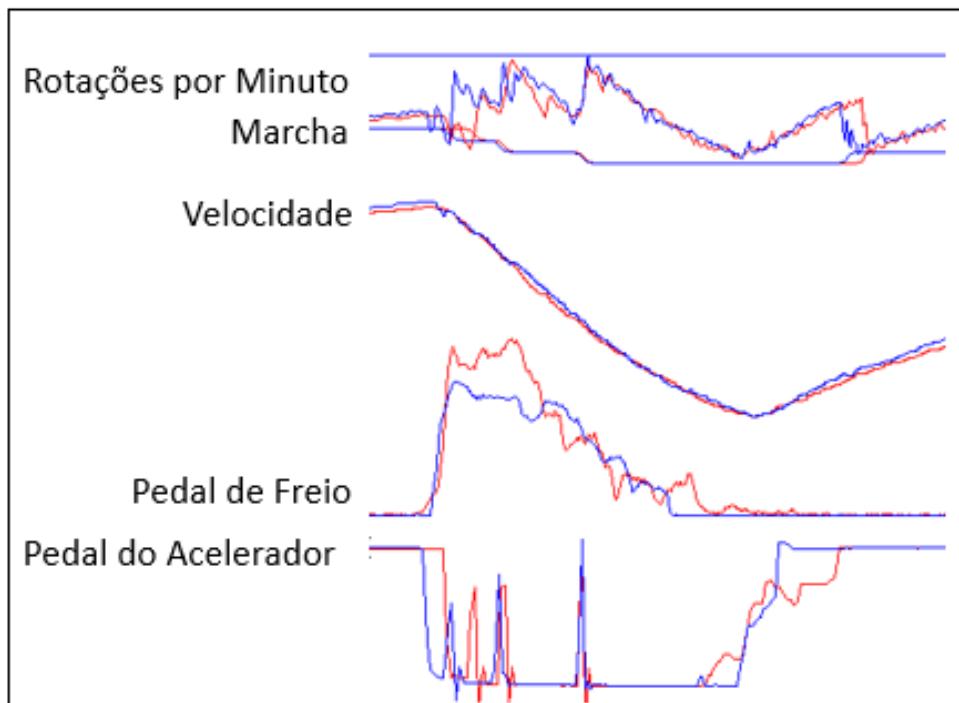


Figura 3.4 – Gráfico Pi Toolbox

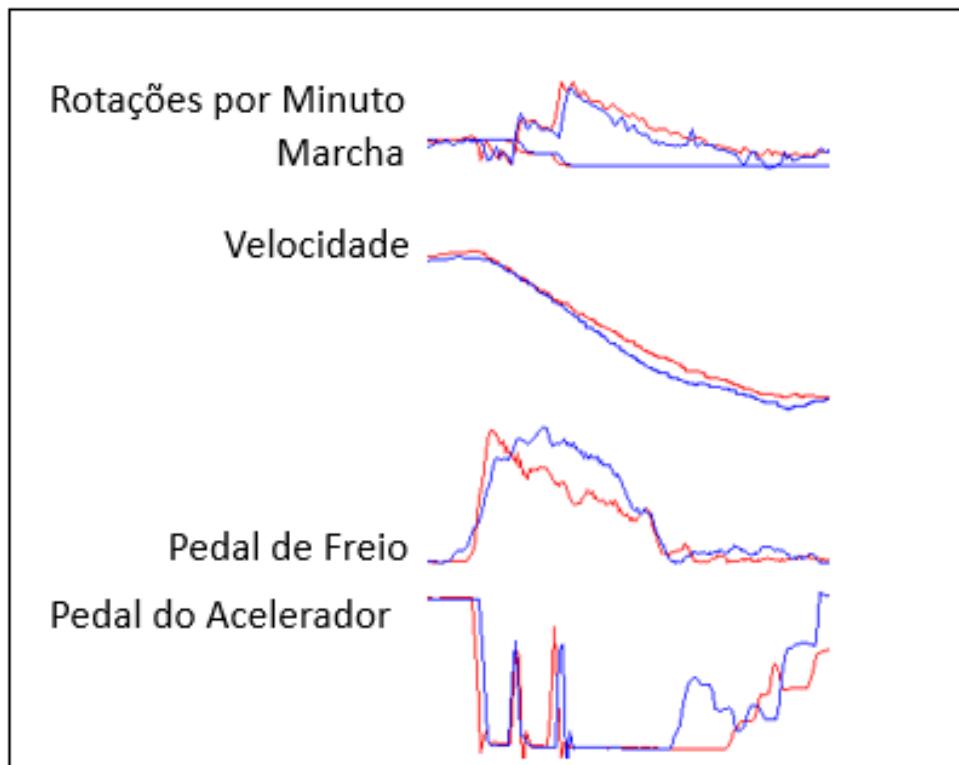


Figura 3.5 – Gráfico Pi Toolbox

Para caracterizar o modelo de pilotagem, foram utilizados os dados de corrente da simulação dinâmica da célula A123 a partir do perfil UDDS (Figura 4.20). Apesar de não haver os dados do pedal de acelerador e freio do veículo, foi considerada uma relação linear com o pedal do acelerador no momento em que a corrente é positiva (descarga) e com o pedal de freio quando a corrente é negativa (regeneração, corrente de recarga) que pode ser observada na Figura 4.21. Sendo assim, estes dados foram usados no cálculo dos KPIs, que caracterizam um modelo de pilotagem da simulação utilizada.

Com os dados de acelerador e freio, foi aplicado o conceito da derivada para mensurar a velocidade de acionamento dos pedais, e então separamos os sinais positivos e negativos de cada um, relacionando com o momento de acionamento e liberação, que foram divididos como KPIs de agressividade (acionamento) e liberação, tanto do freio como do acelerador. Por fim, foi calculado uma média desta derivada durante a amostragem do sinal UDDS para cada uma dessas 4 situações, gerando assim 4 sinais de KPIs que caracterizaram o estilo de pilotagem do sinal UDDS.

O diagrama de blocos do processo, incluindo a obtenção do KPI através dos dados UDDS é representado a seguir (Figura 3.6):

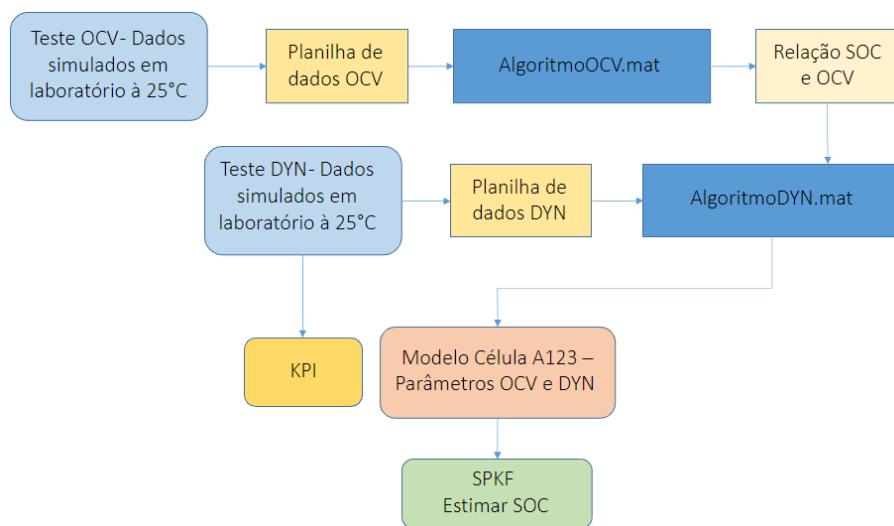


Figura 3.6 – Fluxograma Geral - Incluindo KPI

### 3.6.1 Agressividade e velocidade no pedal de freio

Com os dados exclusivos referentes ao pedal de freio foram calculados os parâmetros de agressividade de freio e liberação do pedal em 5 passos:

1. Derivar o sinal de freio para calcular o parâmetro referente a velocidade do pedal (Figura 4.22);
2. Filtrar o sinal nos pontos de interesse relativos a liberação do pedal (Figura 4.23);
3. Filtrar o sinal nos pontos de interesse relativos a agressividade do freio (Figura 4.24);
4. Calcular a média acumulada do sinal da liberação do pedal para transformá-lo em um KPI (Figura 4.25);
5. Calcular a média acumulada do sinal da agressividade de freio para transformá-lo em um KPI (Figura 4.25).

### 3.6.2 Agressividade e velocidade no pedal do acelerador

Da mesma forma como feito com o sinal de freio, foram calculados os parâmetros agressividade do acelerador e liberação do pedal em 5 passos:

1. Derivar o sinal de acelerador para calcular o parâmetro referente a velocidade do pedal (Figura 4.26);
2. Filtrar o sinal nos pontos de interesse relativos a liberação do pedal (Figura 4.27);
3. Filtrar o sinal nos pontos de interesse relativos a agressividade do acelerador (Figura 4.28);
4. Calcular a média acumulada do sinal da liberação do pedal do acelerador para transformá-lo em um KPI (Figura 4.29);
5. Calcular a média acumulada do sinal da agressividade do acelerador para transformá-lo em um KPI (Figura 4.29).

O algoritmo utilizado neste projeto, referente a essa metodologia apresentada, encontra-se na sessão de anexos A.4, e os resultados gerados compilando o programa podem ser vistos na sessão 4.6.

### 3.7 Rede Neural de Aprendizagem Profunda

Uma RNA totalmente conectada (*Fully Connected*) foi desenvolvida em *Python* para estimar o SOC da célula A123 baseado nos dados do SPKF e dos KPIs. O desenvolvimento da RNA utilizou a biblioteca *Pytorch*, desenvolvida pelo *Facebook*, além da biblioteca *Numpy* e funções específicas para aplicação de redes neurais.

O diagrama de blocos do processo, incluindo a obtenção do SOC final, é demonstrado abaixo (Figura 3.7):

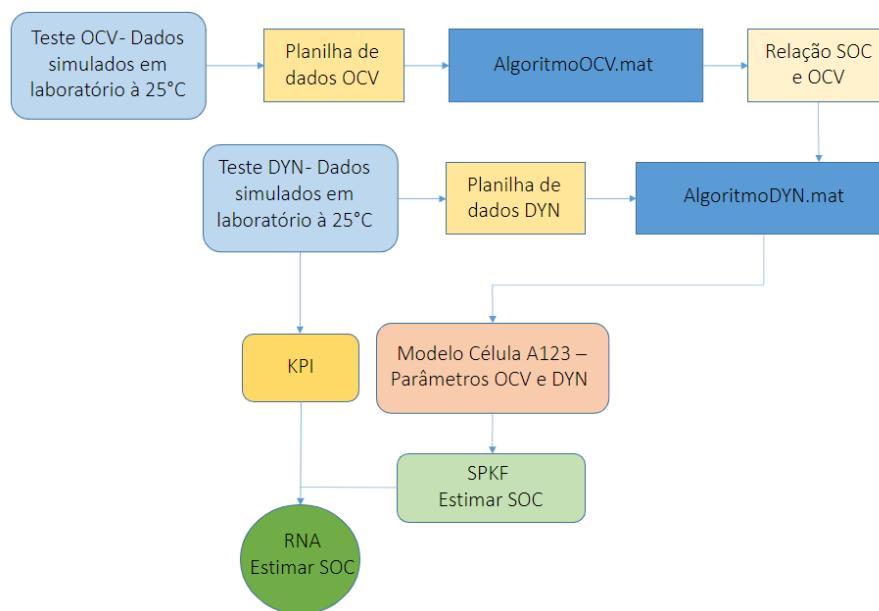


Figura 3.7 – Fluxograma Geral Final

Uma Unidade de Rede Neural consiste apenas em uma camada de entrada e uma camada de saída, representada pela Figura 3.8:

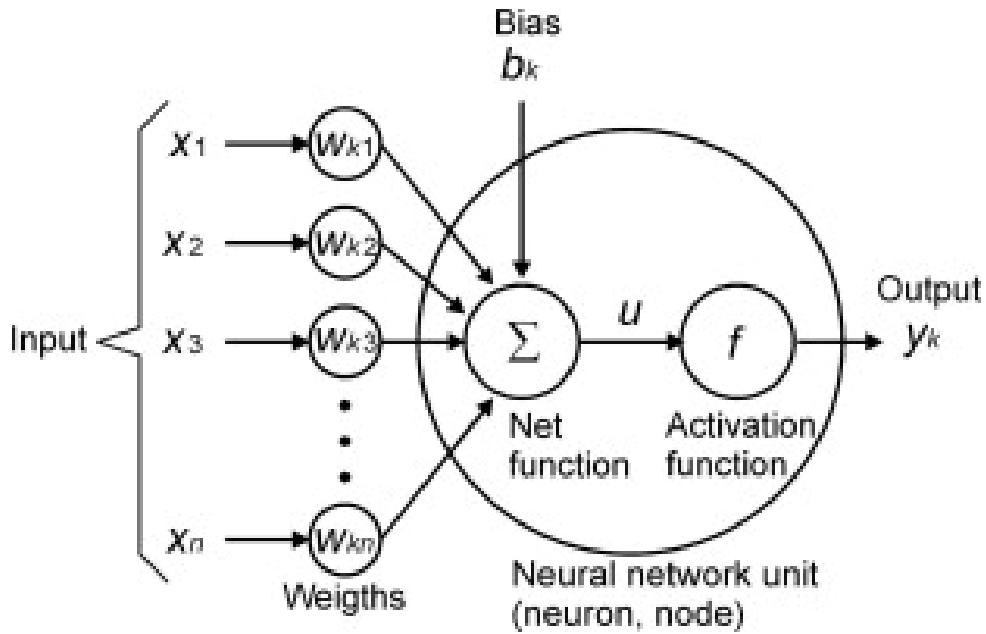


Figura 3.8 – Unidade de Rede Neural

Pode ser equacionada da seguinte forma:

$$y = f(z) \quad (3.29)$$

$$z = b_0 + \sum_{i=1}^d x_i w_i \quad (3.30)$$

Onde  $w_i$  é o peso,  $b_0$  o bias,  $x_i$  o vetor de entrada,  $z$  a saída e  $f$  a função de ativação.

Neste trabalho, os dados foram organizados e normalizados em uma tabela de *Excel* para serem salvos em arquivos de texto *.txt* e importados pelo algoritmo em *Python*. A matriz de entrada para os dados de treinamento *Xtrain* contém as informações de SOC calculada pelo SPKF (Figura 4.17), indicador de performance da agressividade do freio e indicador de performance da liberação do pedal de freio (Figura 4.25), indicador de performance da agressividade do acelerador e indicador de performance da liberação do pedal de acelerador (Figura 4.29), nas colunas 0, 1, 2, 3 e 4 respectivamente, relativo aos dados dinâmicos da célula A123 simulados aplicando o perfil de corrente UDDS (Figura 4.20). Os dados de resposta do treinamento *ytrain* contém o SOC real, o mesmo utilizado para validar o SPKF. Os dados utilizados para o teste foram variados para poder testar o desempenho da RNA em diferentes situações, representando possíveis variações de aplicação *Motorsports*.

A técnica de separação dos dados em lotes foi aplicada para melhorar o desempenho do algoritmo, o modelo da RNA profunda foi definido como sequencial e as funções de ativação

utilizadas foram *Hardtanh* (variação da tangente hiperbólica) e Unidade Exponencial Linear (*Exponential Linear Unit - ELU*).

As RNAs têm a capacidade de aprendizado durante a fase de treinamento, onde o usuário fornece os dados de entrada e informa qual a saída correspondente. Para treinar uma rede devem ser fornecidos os dados de entrada  $x_n$  e sua saída real correspondente  $y(x)$ , a rede é treinada para que aprenda os valores dos pesos e bias a fim de aproximar a saída da rede  $f(z)$  ao valor de  $y(x)$ . Durante o treinamento de uma RNA e, também para calcular seu desempenho no conjunto de dados de teste, é necessário utilizar um algoritmo que permita encontrar esses pesos e bias ideais e quantifique o quanto bem este objetivo está sendo atingido. Esta função pode ser definida como a função de custo, função de perda ou objetivo, representada por:

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2 \quad (3.31)$$

Onde  $w$  são todos os pesos da rede,  $b$  os bias,  $n$  o número total de entradas de treinamento e  $a$  é o vetor de saída da rede.  $C$  também é conhecido como erro quadrático médio ou *MSE*. Valores de  $C(w,b)$  próximos de 0 indicam que o algoritmo de treinamento está obtendo bons resultados e  $C(w,b)$  maiores indicam que existe um desvio grande entre a saída da rede e o resultado esperado, portanto o algoritmo tem como objetivo minimizar o custo  $C(w,b)$  em função dos pesos e bias.

Para implementar a minimização desta função é necessário aplicar uma técnica chamada descida de gradiente na função de custo, ela pode ser vista como uma maneira de dar pequenos passos na direção que  $C$  diminui, ou seja, a função de custo vai de encontro ao seu valor mínimo.

As funções *trainmodel* e *runeepoch* foram criadas para serem executadas durante o treinamento e teste da RNA. O otimizador escolhido foi o gradiente descendente estocástico (*Stochastic Gradient Descent*) e os parâmetros de perda (*loss*), acurácia (*accuracy*) e erro (*error*) calculados a cada época durante o treinamento, baseado no valor estimado de SOC pela RNA. A função *runeepoch* é responsável por inserir os dados no modelo sequencial, obter a resposta do SOC estimado e retornar os valores médios de erro (*error*) e acurácia (*accuracy*). Com esse algoritmo foi possível calcular os valores estimados de SOC.

Em resumo, a metodologia utilizada pode ser apresentada por uma RNA totalmente conectada, onde todas as camadas são interligadas entre si, com um algoritmo de retropropagação de gradiente descendente estocástico, minimizando uma função de custo de erro quadrático

médio. Os dados de entrada são o SOC e os KPIs, que caracterizam a influência do piloto no consumo de carga da bateria.

O algoritmo utilizado neste projeto, referente a essa metodologia apresentada, encontra-se na sessão de anexos B.1, e os resultados gerados compilando o programa podem ser vistos na sessão 4.7.

## 4 RESULTADOS E COMENTÁRIOS

### 4.1 Teste 1: Tensão de Circuito Aberto

Os parâmetros de tensão da célula extraídos do teste foram  $V_{min} = 2$  V e  $V_{max} = 3.75$  V. Os testes foram feitos para temperaturas de -25°C à 45°C (Figura 4.1) e o modelo desenvolvido utilizou os dados de 35°C, por se assemelhar ao caso real de aplicação. Foi observado, também, que a maior não linearidade ocorre principalmente em temperaturas menores que 0°C, mas esta, não contempla os casos de aplicação do nosso problema de interesse. Os resultados dos passos do teste foram divididos em 4 partes para uma melhor visualização e estão ilustrados nas Figuras 4.2 e 4.3.

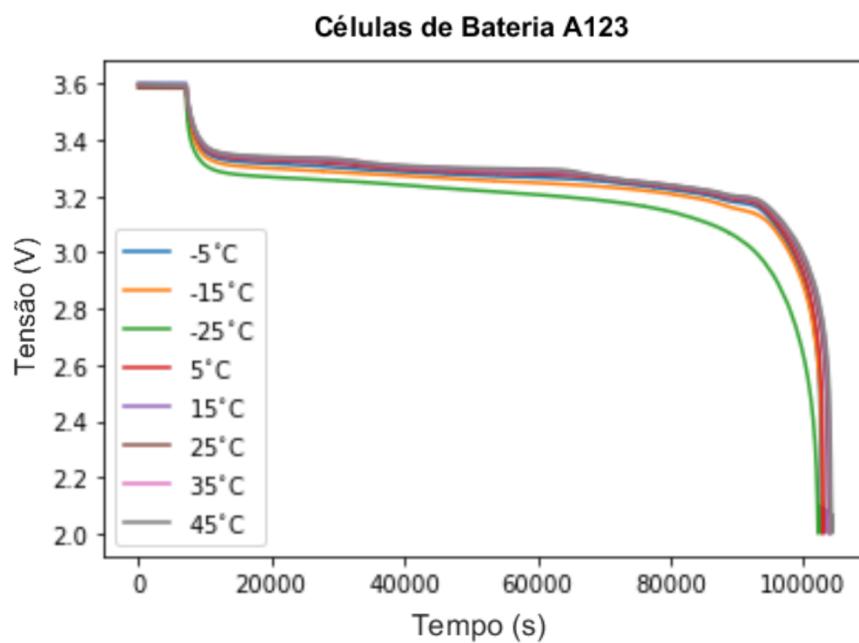


Figura 4.1 – Tensão [V] nos passos 1 e 2 para todas as temperaturas.

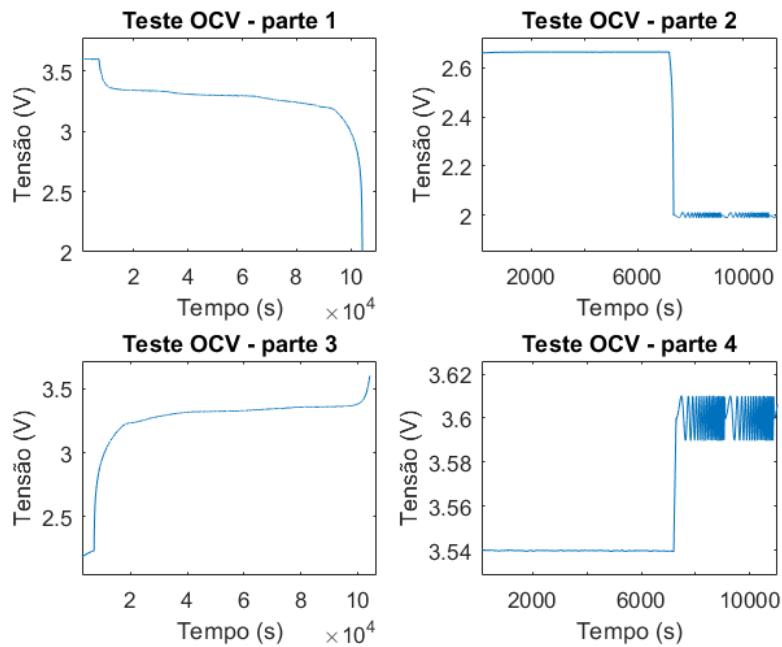


Figura 4.2 – OCV - A123 à 35°C

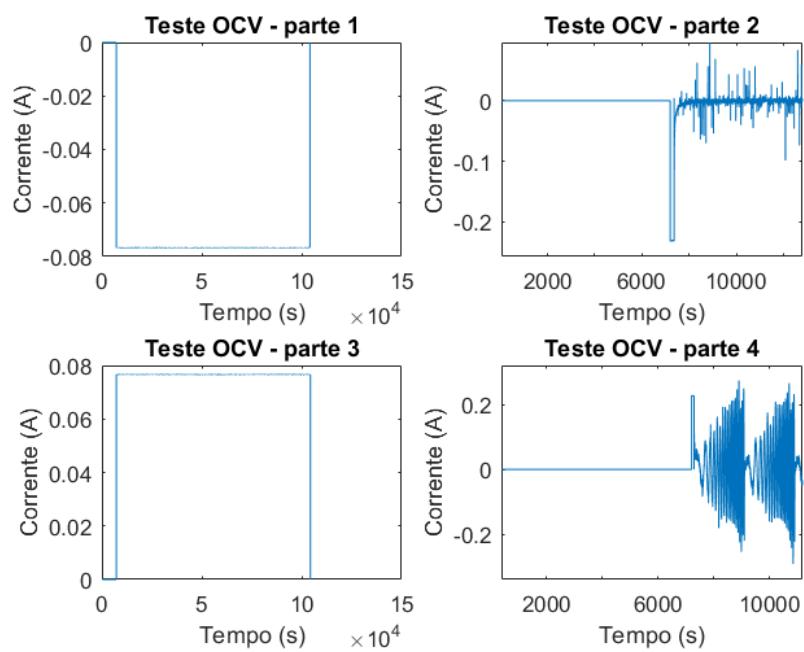


Figura 4.3 – OCV - A123 à 35°C

#### 4.2 Teste 2: Perfil Dinâmico de Carga e Descarga

Os parâmetros de tensão da célula extraídos do teste foram  $V_{min} = 2$  V e  $V_{max} = 3.75$  V. Novamente, os testes foram realizados nas temperaturas entre -25°C e 45°C para a célula A123 e os dados de tensão (Figura 4.4) e corrente (Figura 4.5) à 35°C foram utilizados como dados de entrada do algorítimo, no qual foram obtidos os parâmetros dinâmicos do modelo ESC.

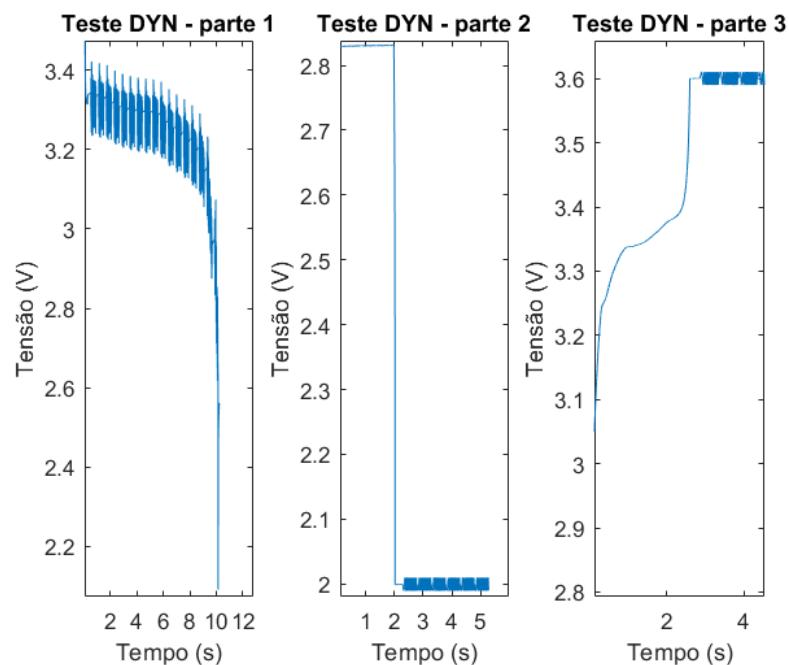


Figura 4.4 – DYN - A123 à 35°C

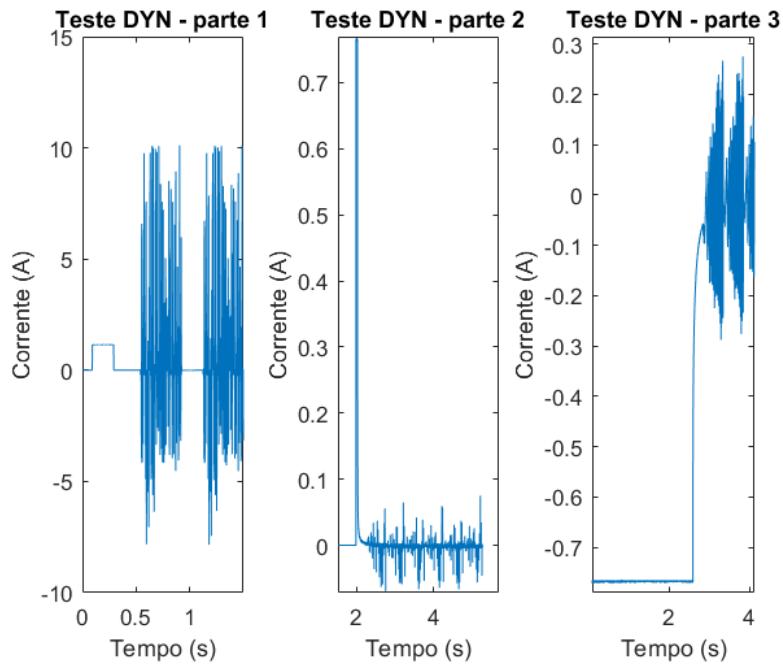


Figura 4.5 – DYN - A123 à 35°C

### 4.3 Algoritmo da Tensão de Circuito Aberto

Os resultados gerados pelo algoritmo A.1 apresentaram a relação entre a tensão de circuito aberto (OCV) e o estado de carga da bateria (SOC) para as temperaturas entre -25°C e 45°C, com passos de 10°C em cada teste. Pode-se observar na Figura 4.6, para temperaturas mais baixas, a não linearidade do sistema de carga da célula A123 é maior, mas conforme as temperaturas aumentam, os resultados se aproximam mais. Neste trabalho foi considerado um intervalo entre 25°C a 35°C, para estes valores, a estimativa do algoritmo da tensão de circuito aberto mostrou bons resultados, como observado nas figuras 4.7 e 4.8.

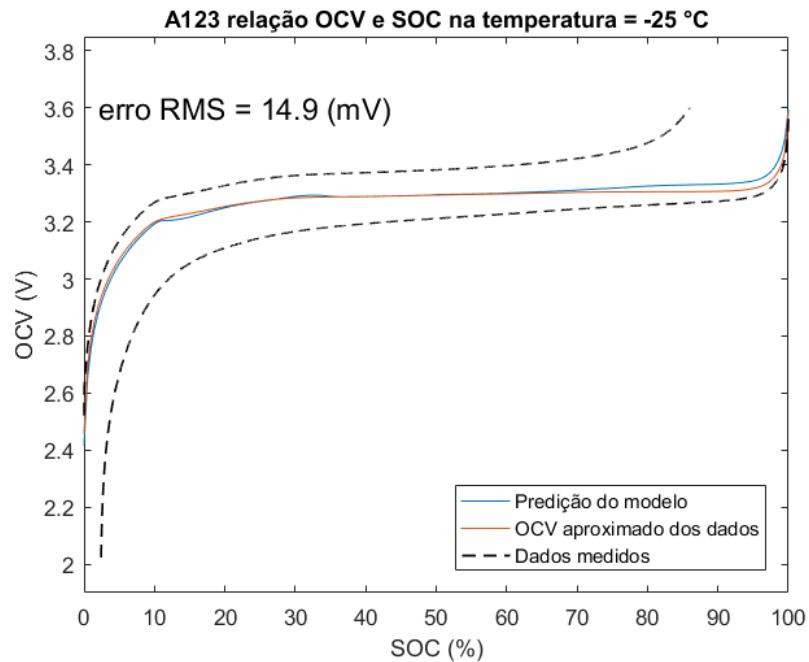


Figura 4.6 – Relação OCV e SOC à -25°C

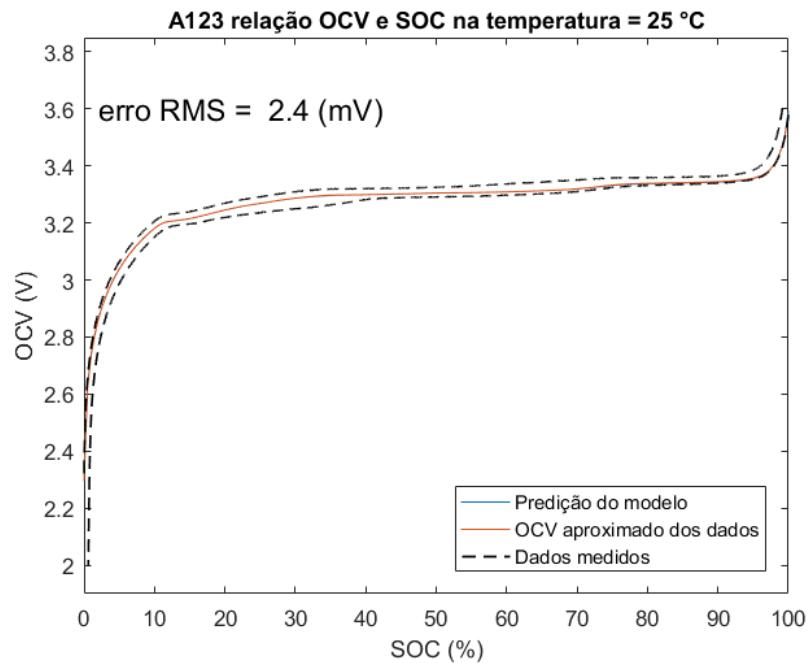


Figura 4.7 – Relação OCV e SOC à 25°C

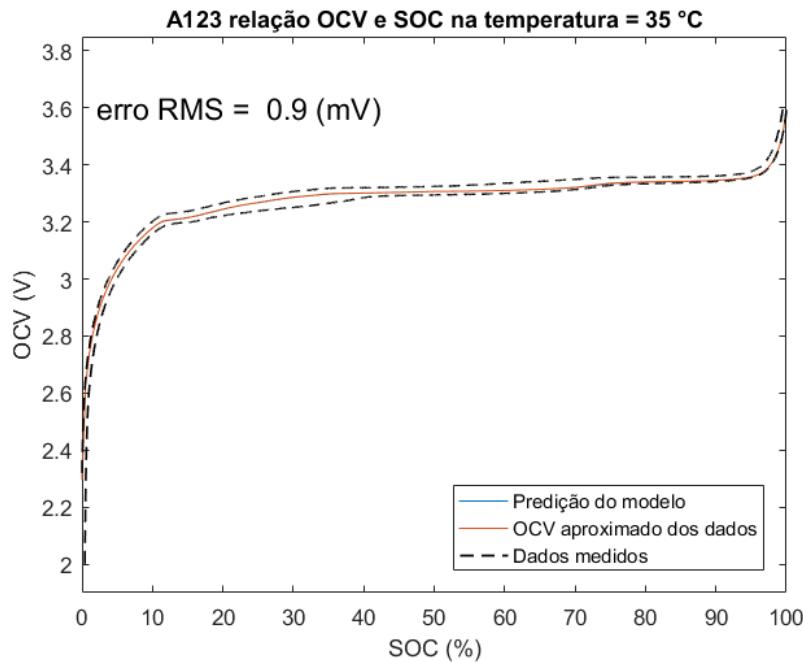


Figura 4.8 – Relação OCV e SOC à 35°C

#### 4.4 Algoritmo do Teste Dinâmico

Os gráficos gerados pelo algoritmo A.2 apresentaram os parâmetros do modelo ESC da bateria A123 para as temperaturas de -25°C a 45°C, com passos de 10°C em cada teste. Assim como descrito na metodologia, eles são necessários para continuidade e aplicação no filtro de Kalman Ponto Sigma para estimativa do SOC. A lista de parâmetros e seus respectivos resultados são apresentados a seguir:

1.  $Q$  - Capacidade  $Q$  em Ah (Figura 4.9)
2.  $\eta$  - Eficiência de Coulomb (Figura 4.10)
3.  $G$  - Parâmetro  $\gamma$  de Histerese (Figura 4.11)
4.  $M$  - Parâmetro M de Histerese[V] (Figura 4.12)
5.  $M_0$  - Parâmetro  $M_0$  de Histerese [V] (Figura 4.13)
6.  $R_0$  - Resistência Interna [Ohm] (Figura 4.14)
7.  $RC$  - Constante de tempo  $R - C$  [s] (Figura 4.15)
8.  $R$  - Resistência do parâmetro  $R - C$  [Ohm] (Figura 4.16)

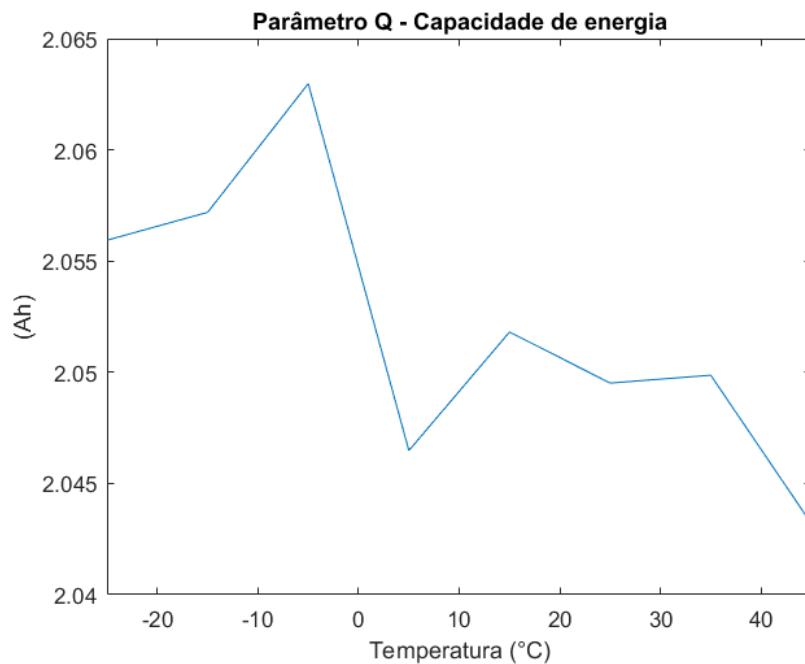


Figura 4.9 – Parâmetro  $Q$

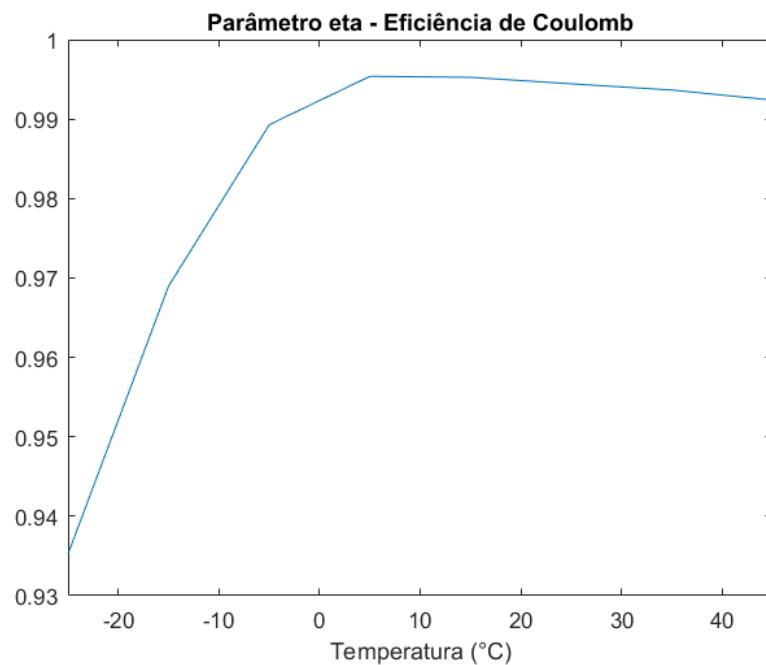


Figura 4.10 – Parâmetro  $\eta$

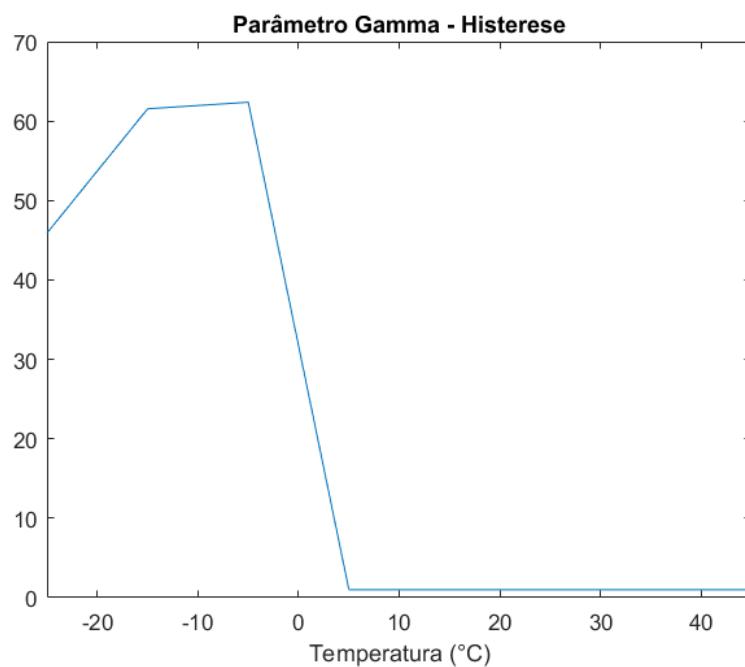


Figura 4.11 – Parâmetro  $\gamma$

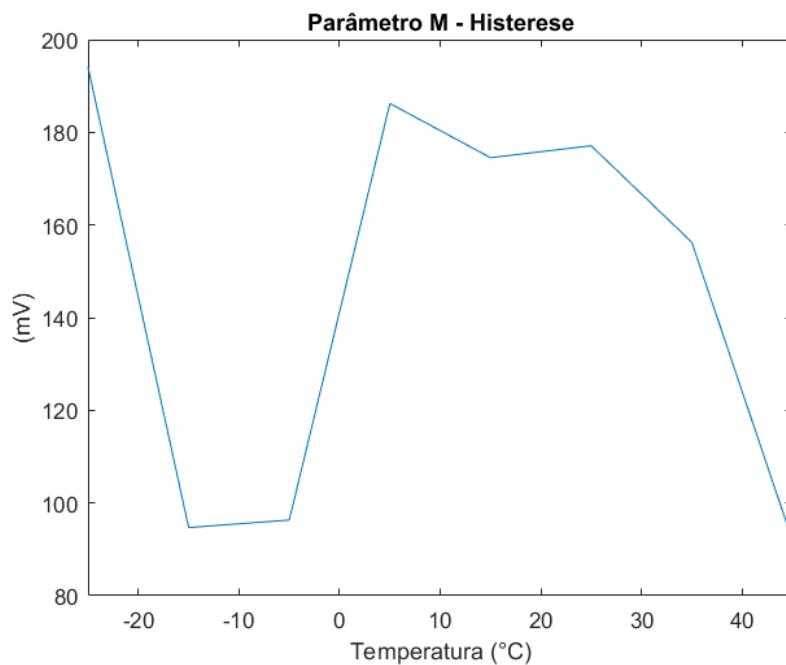


Figura 4.12 – Parâmetro  $M$

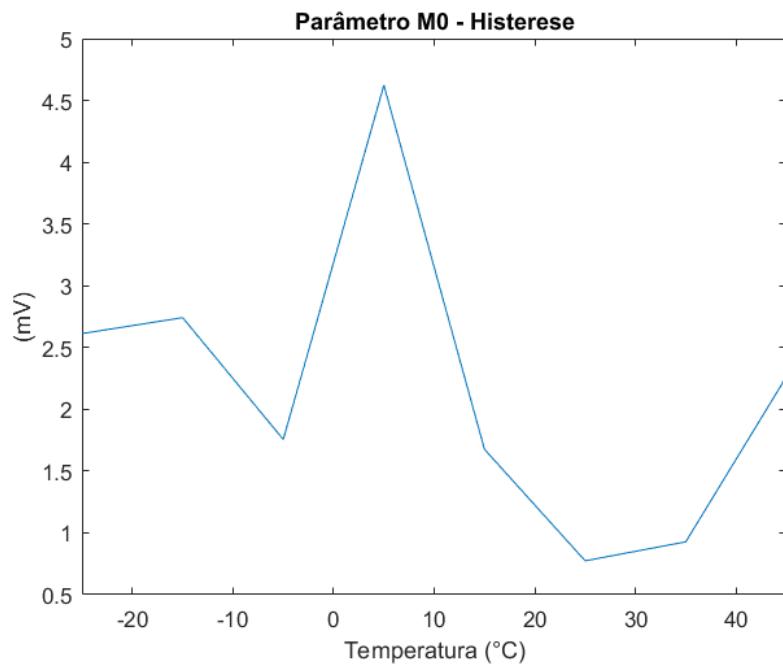


Figura 4.13 – Parâmetro  $M_0$

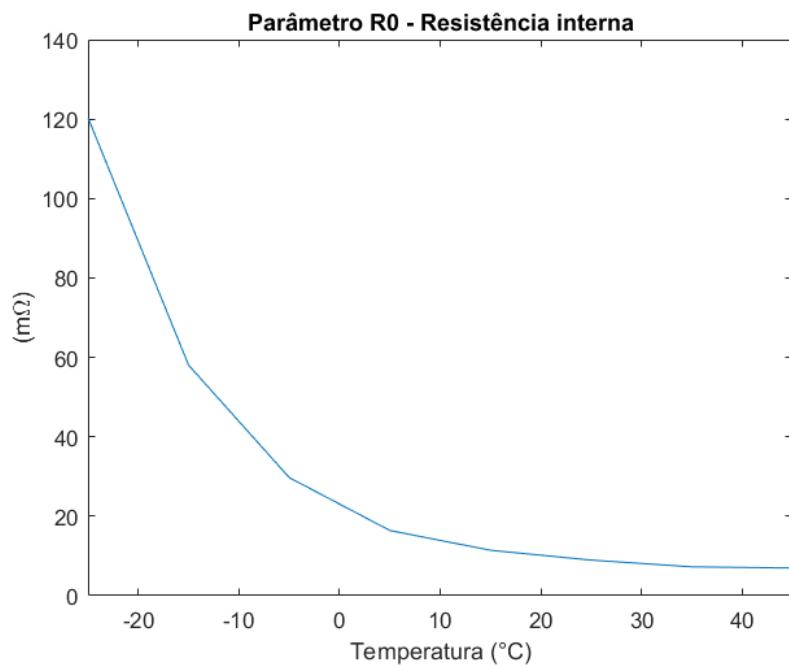
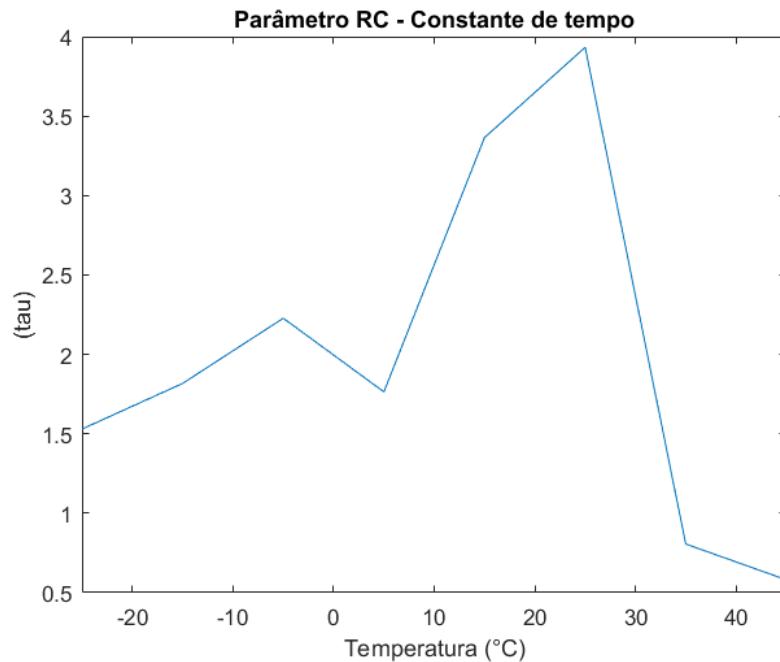
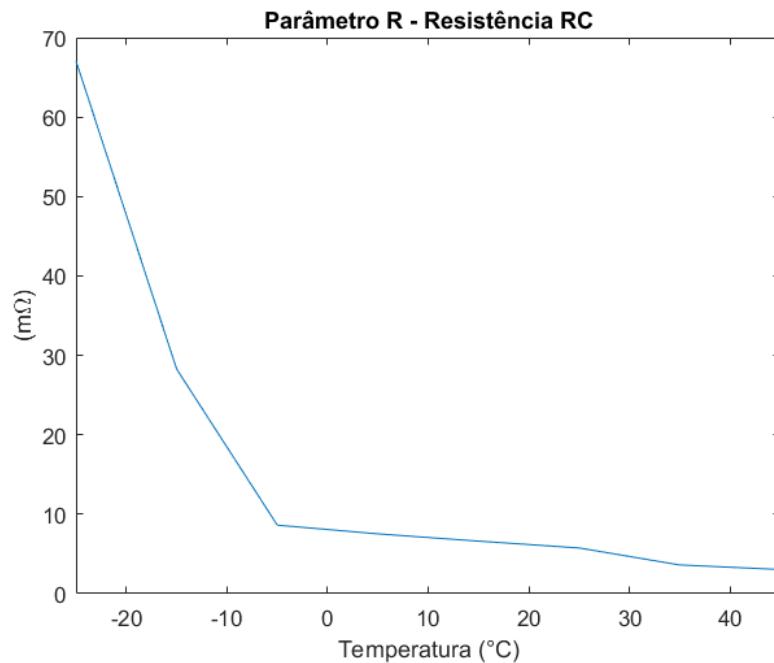


Figura 4.14 – Parâmetro  $R_0$

Figura 4.15 – Parâmetro  $RC$ Figura 4.16 – Parâmetro  $R$ 

#### 4.5 Filtro de Kalman Ponto Sigma

Os resultados simulados do algoritmo SPKF A.3 com os dados do modelo ESC e perfil de corrente UDDS geraram a primeira estimativa de SOC (Figura 4.17).

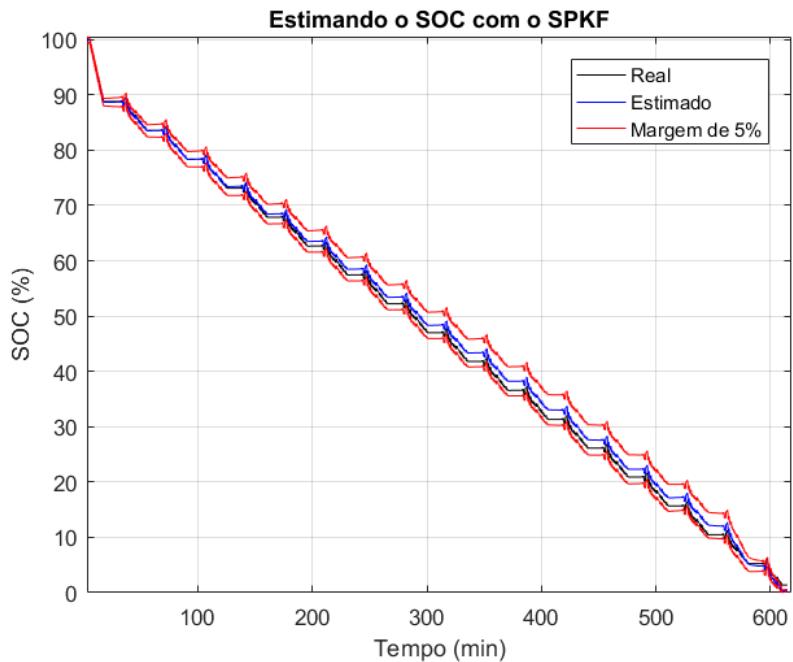


Figura 4.17 – Estimativa SOC SPKF

O resultado obteve um RMSE= 1.11% e 1.93% de tempo fora da borda delimitada de 5% de erro (Figura 4.18).

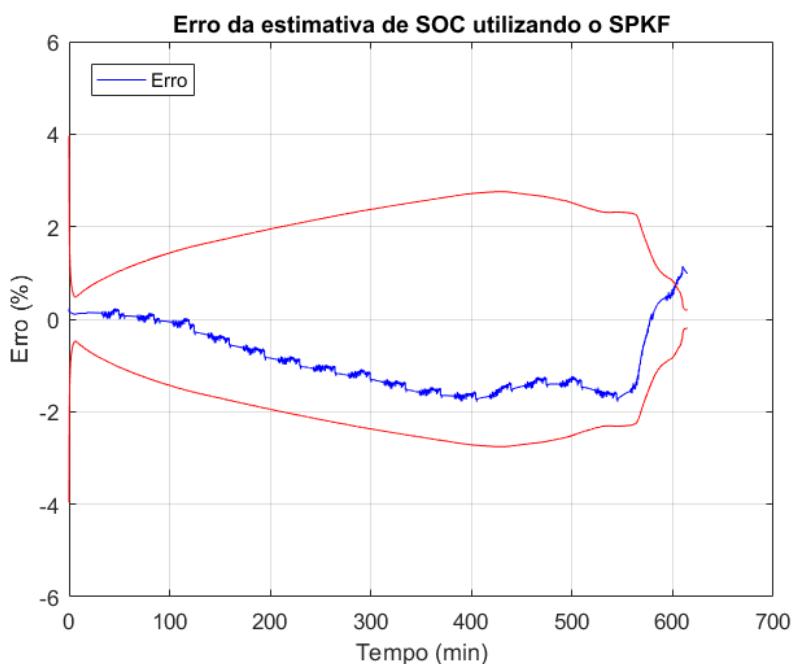


Figura 4.18 – Erro SPKF

Embora o resultado deste algoritmo seja bom em comparação aos outros métodos e, dentro de uma margem de erro aceitável, este trabalho propôs uma nova maneira de otimizar a precisão dos cálculos utilizando uma abordagem específica para a aplicação em *Motorsports*.

Houve melhora na precisão, principalmente para valores de baixo SOC (Figura 4.19), onde na prática são os momentos mais importantes durante uma corrida, além da inclusão de uma personificação do piloto no algoritmo.

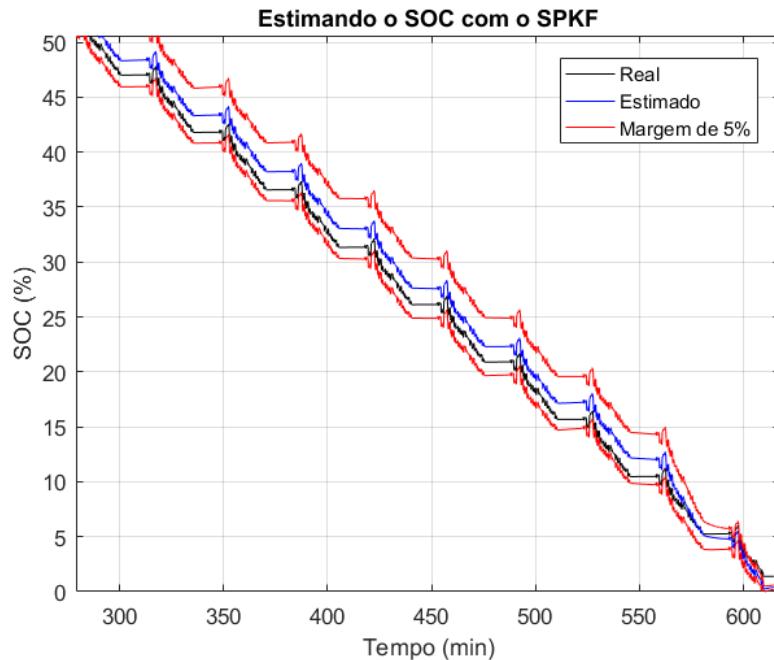


Figura 4.19 – Estimativa SOC SPKF < 50%

## 4.6 Indicadores de Performance do Piloto

Para obtenção desses indicadores, o primeiro passo foi inicializar o sistema, carregando os dados e parametrizando as variáveis que seriam utilizadas e, em seguida, extraídos os sinais de freio e acelerador (Figura 4.21) do sinal UDDS (Figura 4.20). Com os sinais separados de freio e acelerador, iniciou-se o processo de cálculos para obter os indicadores desejados. Os sinais dos KPIs de agressividade e liberação do pedal de freio podem ser visualizados na Figura 4.25, e os sinais dos KPIs de agressividade e liberação do pedal do acelerador na Figura 4.29.

### 4.6.1 Agressividade e velocidade no pedal de freio

O processo da derivada do sinal de freio pode ser observado na Figura 4.22, tal como a separação deste sinal em liberação (Figura 4.23) e agressividade (Figura 4.24), antes de calcularmos os indicadores. Os parâmetros da filtragem, apresentados na sessão de metodo-

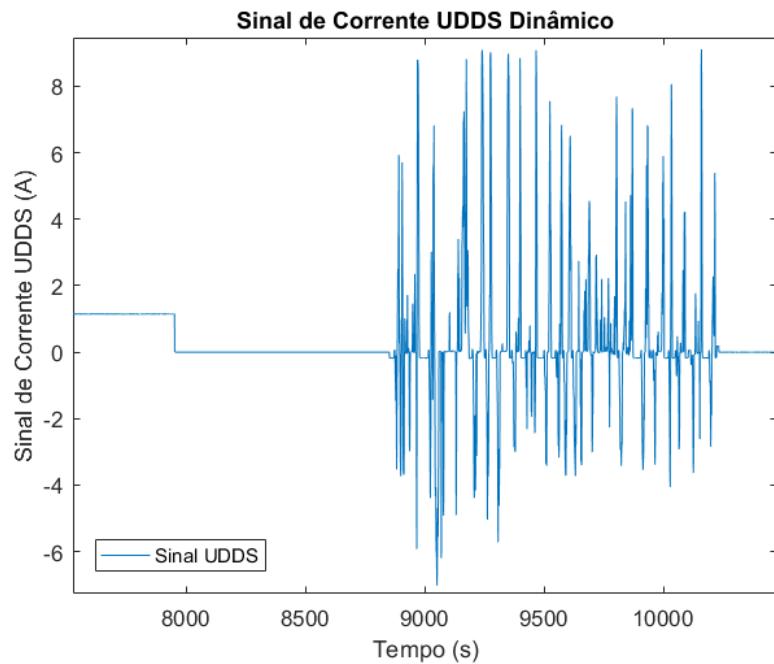


Figura 4.20 – Sinal de Corrente do perfil UDDS

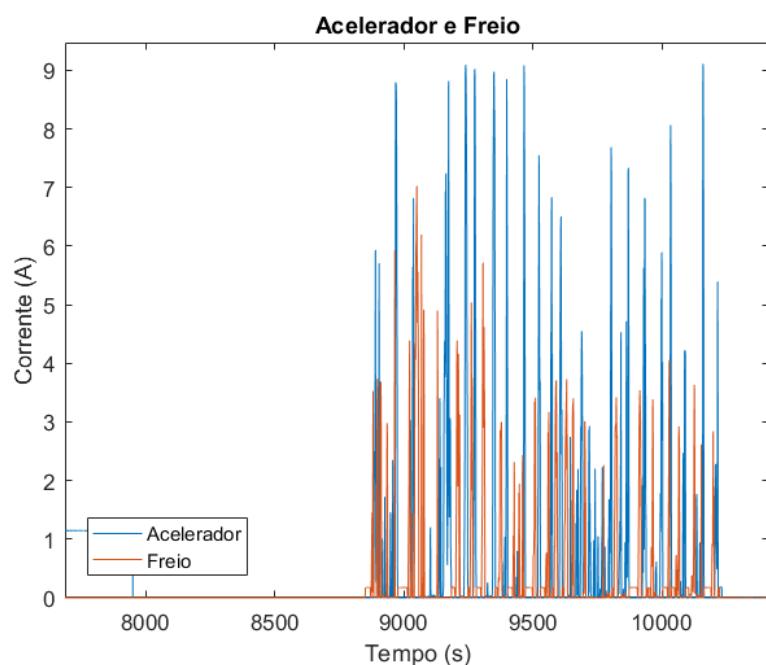


Figura 4.21 – Sinais de Freio e Acelerador do perfil UDDS

dologia 3.6, foram escolhidos para melhor ajuste do sinal e geração dos KPIs de liberação e agressividade do pedal de freio (Figura 4.25).

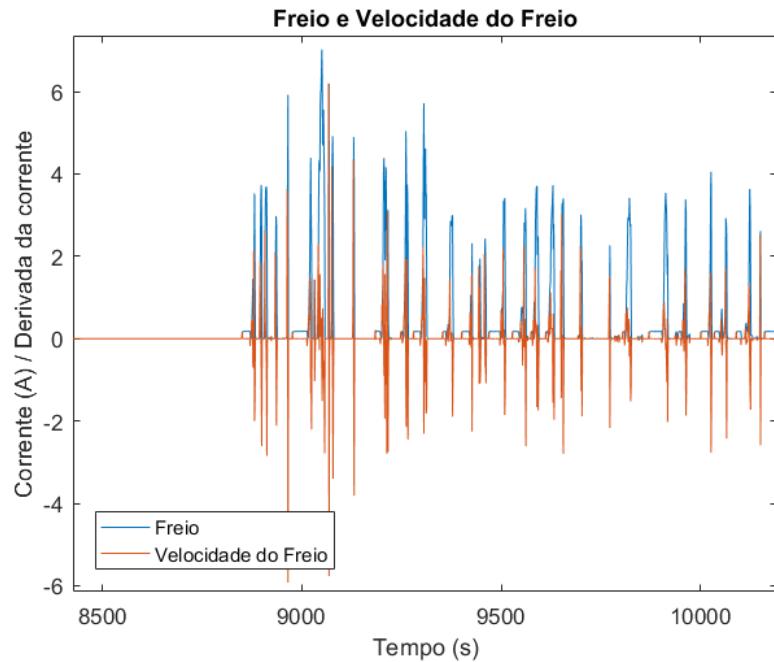


Figura 4.22 – Velocidade do Freio

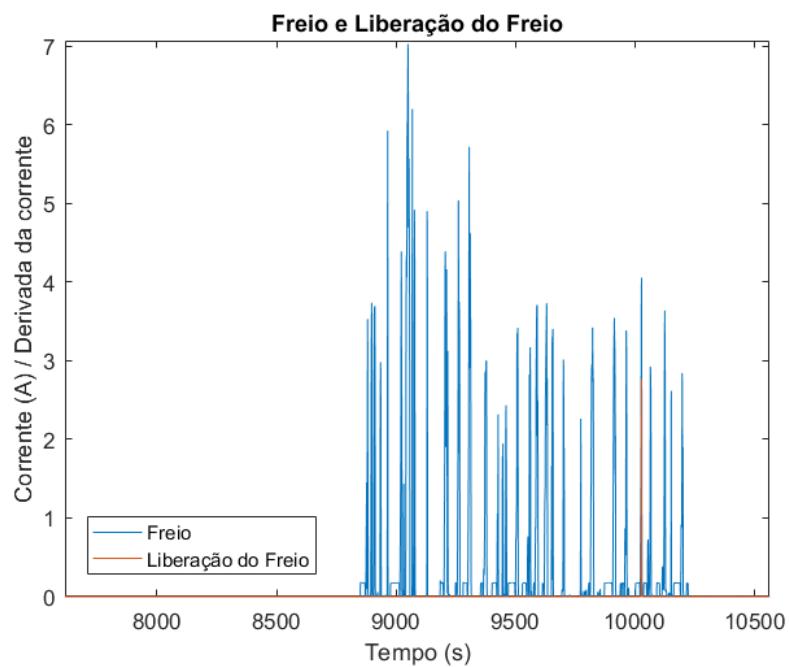


Figura 4.23 – Liberação do Freio

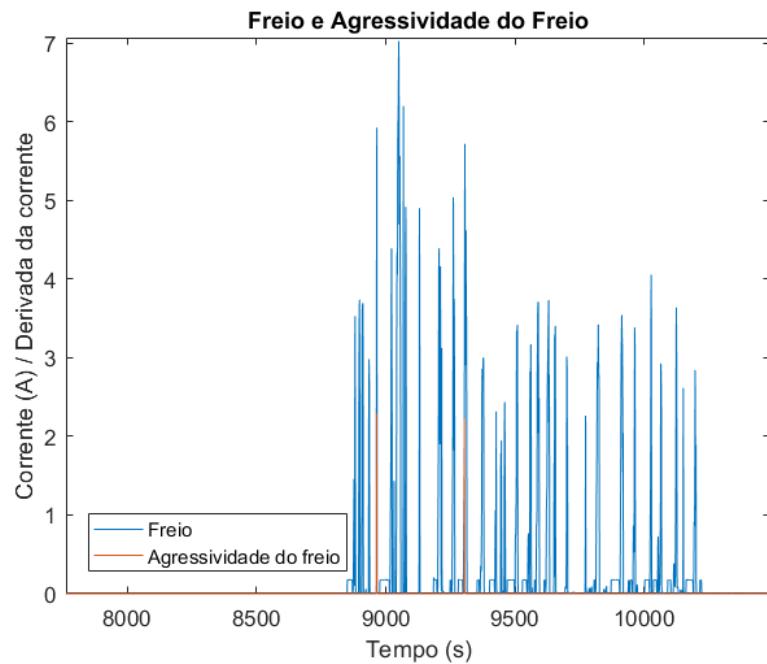


Figura 4.24 – Agressividade do Freio

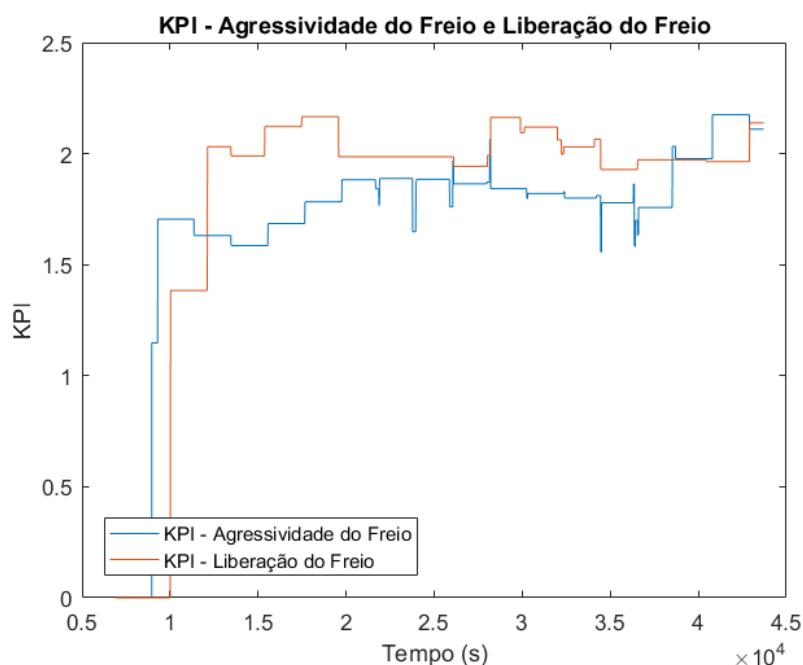


Figura 4.25 – KPI do Freio

#### 4.6.2 Agressividade e velocidade no pedal do acelerador

O processo da derivada do sinal do acelerador pode ser observado na Figura 4.26, tal como a separação deste sinal em liberação (Figura 4.27) e agressividade (Figura 4.28), antes de calcularmos os indicadores. Novamente, os parâmetros da filtragem, foram escolhidos para melhor ajuste do sinal e geração dos KPIs de agressividade e liberação do acelerador (Figura 4.29).

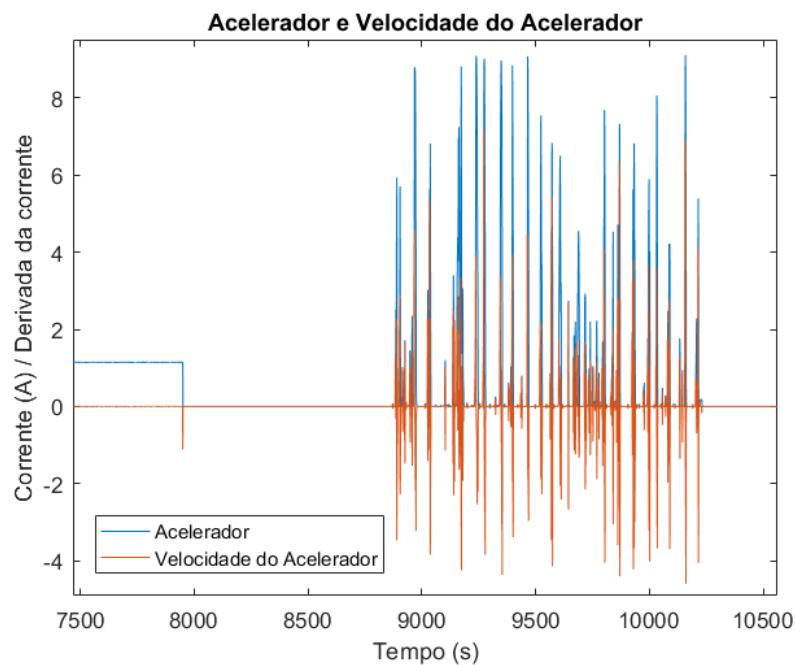


Figura 4.26 – Velocidade do Acelerador

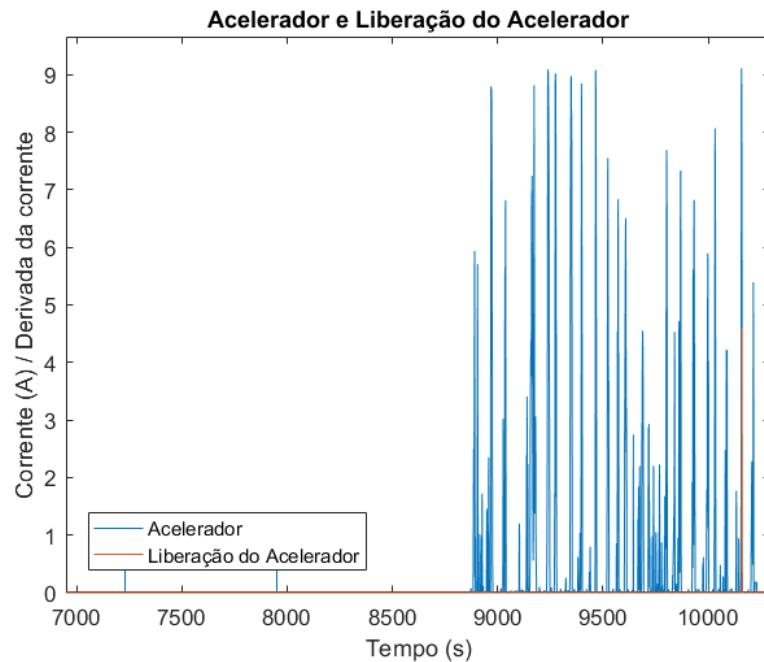


Figura 4.27 – Liberação do Acelerador

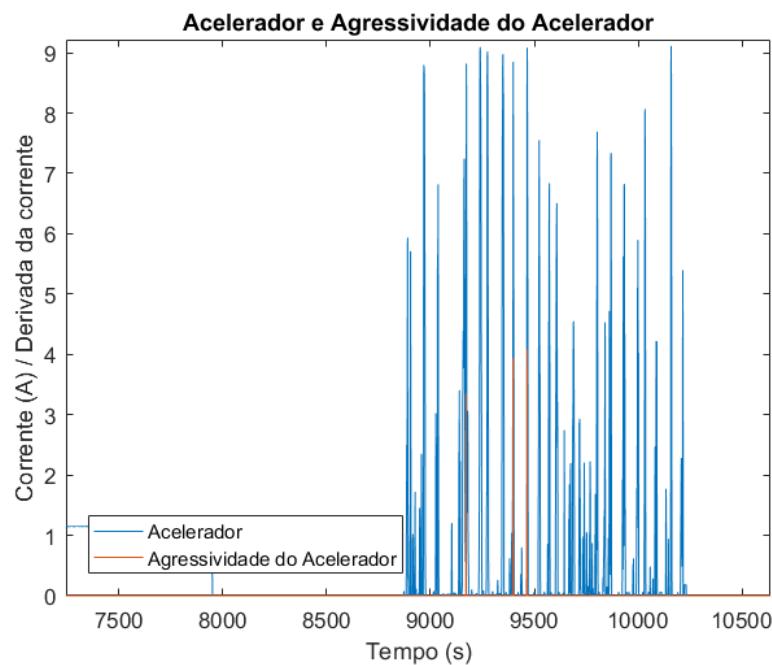


Figura 4.28 – Agressividade do Acelerador

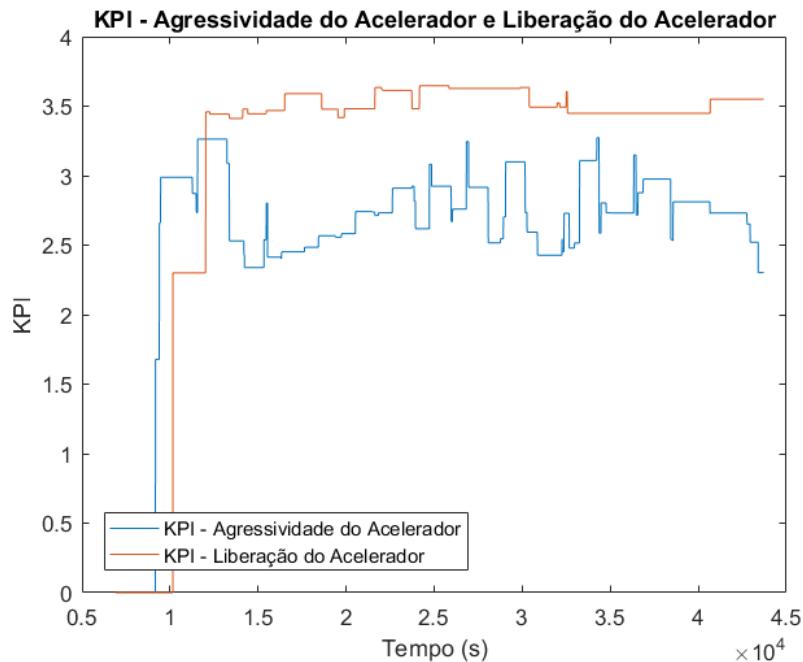


Figura 4.29 – KPI do Acelerador

#### 4.7 Rede Neural de Aprendizagem Profunda

Os resultados iniciais foram gerados com os dados de teste e treinamento à 35°C. Estes dados foram os mesmos utilizados para obtenção do SOC pelo SPKF e geração de KPI. Como relatado na Sessão 4.5, o SPKF obteve boa precisão, com um RMSE= 1,1% e uma percentagem de 1,93% de tempo fora da borda delimitada de 5% de erro para mais ou para menos.

A RNA desenvolvida apresenta os seguintes hiper parâmetros, indicados na Tabela 4.1:

Tabela 4.1 – Configuração Rede Neural

Hiperparâmetro	Valor
Camadas ocultas	2
Neurônios por camada	10
Função de ativação	ELU
Taxa de aprendizado	0.01
Agrupamentos de treinamento	8
Épocas de treinamento	100

E os parâmetros de avaliação de desempenho foram:

- Acurácia: Representa o quanto os dados estimados estão dentro da borda delimitada de 5% de erro para mais ou para menos

- Erro: Mede o erro quadrado médio, norma L2 ao quadrado, entre cada elemento na entrada (SOC estimado) e o destino (SOC real)

O resultado da simulação pode ser analisado na Figura 4.30.

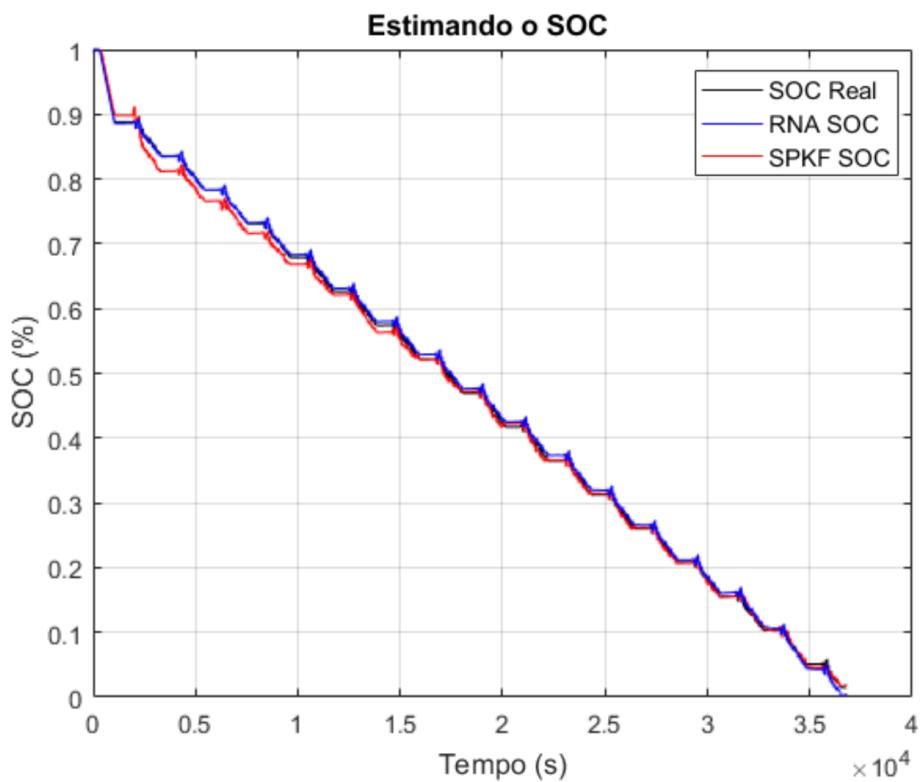


Figura 4.30 – Resultado da estimativa do SOC pelos modelos

O modelo para 35°C apresentou uma avaliação de acurácia= 97.91% e erro= 0.0027% enquanto o SPKF apresentou uma acurácia= 92.28% e um erro= 0.0029%. O resultado apresentado pelo SPKF também apontou que o erro da estimativa de SOC foi maior nos valores finais, onde a não linearidade dos dados era mais evidenciada e o resultado estimado apresentou o maior desvio em relação ao valor real de SOC. Porém, o modelo com a RNA obteve maior precisão no cálculo em relação ao SPKF para valores menores de SOC, como ilustrado nas Figuras 4.31 e 4.32 para SOC menor que 50% e 20%, respectivamente.

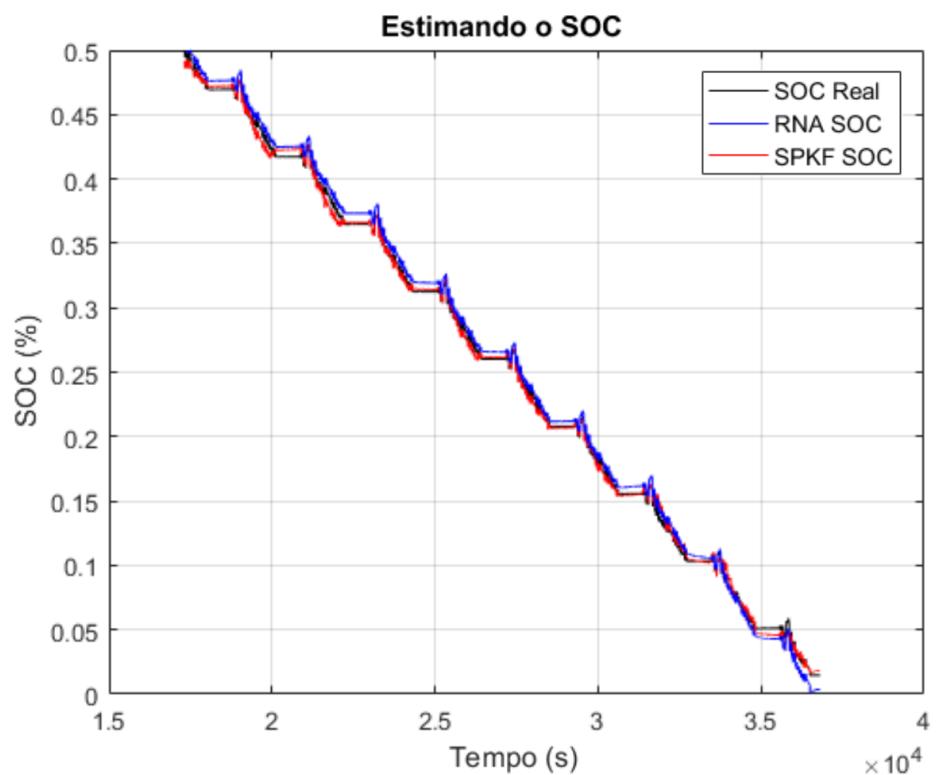


Figura 4.31 – Resultado da estimativa dos modelos com SOC menor que 50%

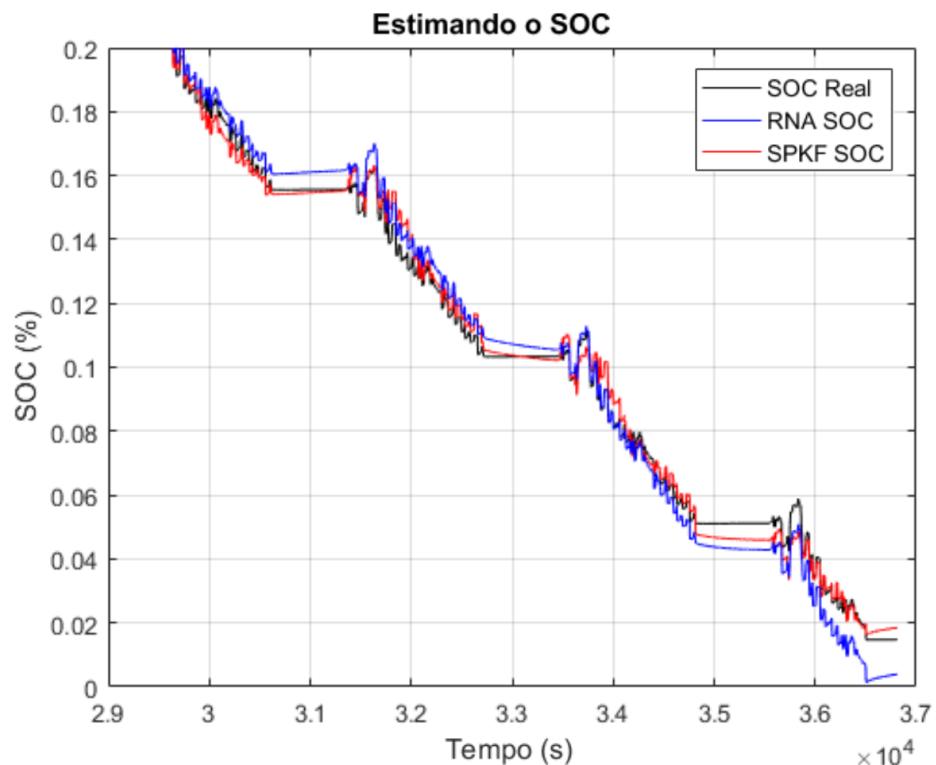


Figura 4.32 – Resultado da estimativa dos modelos com SOC menor que 20%

#### 4.8 Validação da RNA

Após demonstrado um bom desempenho quando utilizado os dados de treinamento como teste da RNA e definidas as melhores configurações de hiper parâmetros para cada temperatura de teste, foi testado o desempenho para dados diferentes de treinamento e validação.

Foram utilizados 2 tipos de perfis de carga e descarga da bateria, um representando características de pilotagem urbana (UDDS) (Figura 4.33) e o de direção urbana federal (FUDS) (Figura 4.34). Na sessão anterior, o perfil UDDS foi utilizado tanto nos dados de treinamento como nos dados de validação, nesta sessão os resultados foram apresentados simulando perfis diferentes do utilizado durante o treinamento, situação mais próxima à aplicação real em *Motorsports*.

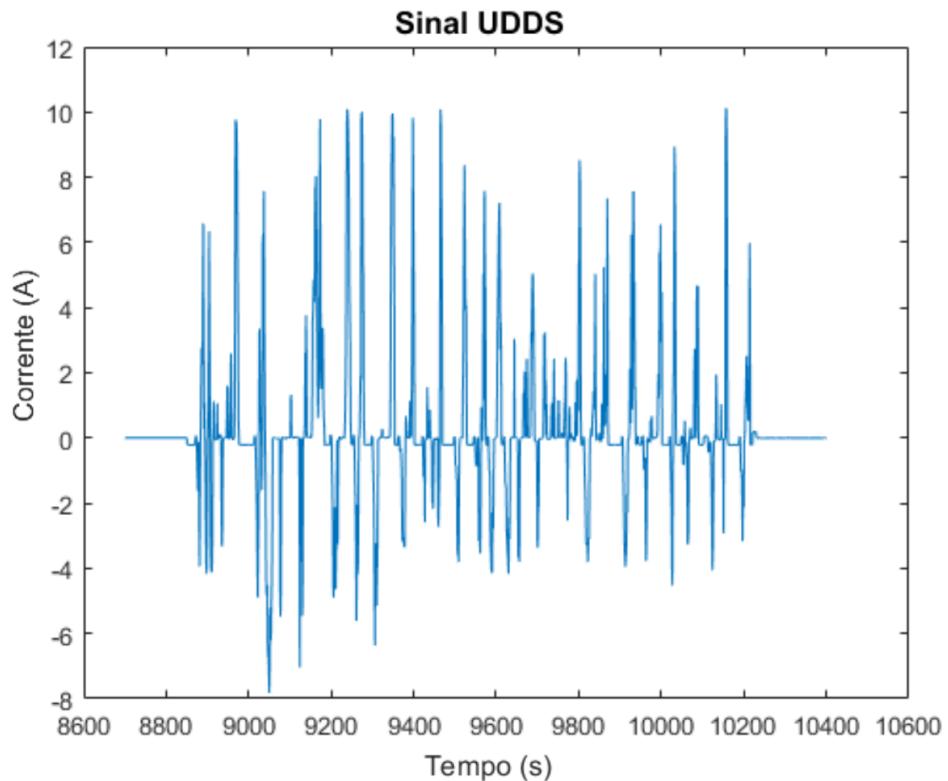


Figura 4.33 – Sinal UDDS

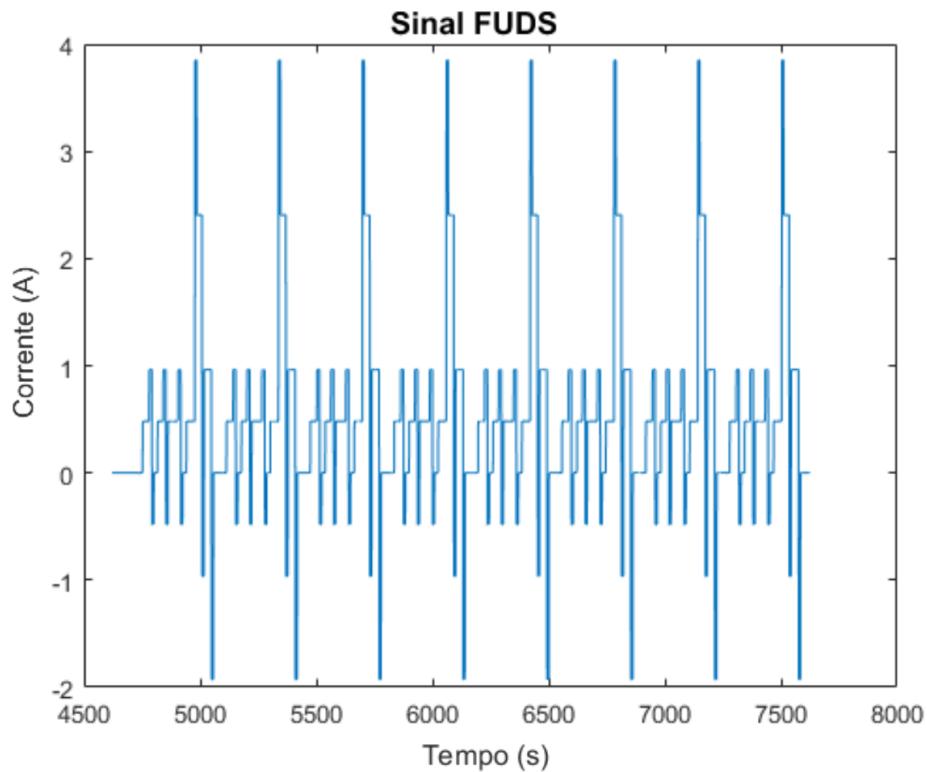


Figura 4.34 – Sinal FUDS

A RNA apresentou uma avaliação de acurácia= 100% e erro= 0.0052% com uma configuração de parâmetros listada na Tabela 4.2.

Tabela 4.2 – Configuração Rede Neural

Hiperparâmetro	Valor
Camadas ocultas	1
Neurônios por camada	100
Função de ativação	Hardtanh
Taxa de aprendizado	0.1
Agrupamentos de treinamento	128
Épocas de treinamento	100

Os resultados apresentados pela Figura 4.35 mostraram que a variação mais abrupta dos KPIs (Figuras 4.36 e 4.37) para este perfil de carga/descarga FUDS impactavam mais negativamente no valor estimado de SOC da RNA. O que é evidenciado quando plotamos a função de erro da simulação (Figura 4.38).

Para melhorar a resposta da RNA notou-se que seria necessário suavizar as variações do KPI, aplicando o cálculo da média móvel para cada ciclo FUDS em substituição ao método que considerava a média apenas do valor atual  $n$  e o valor passado mais próximo  $n - 1$ . Desta forma, foi possível minimizar a variação do KPI durante cada ciclo e ainda assim manter

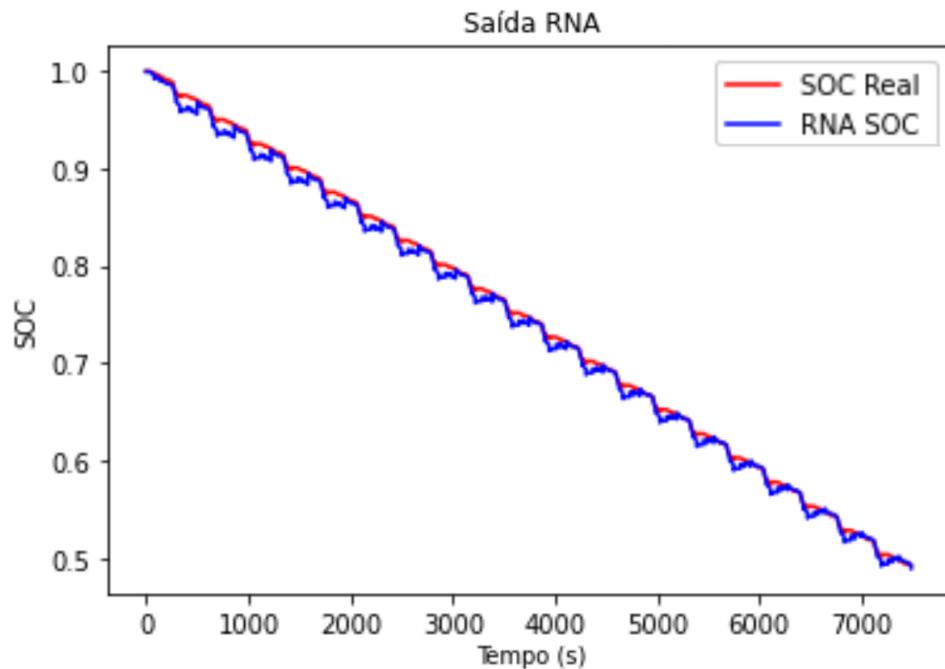


Figura 4.35 – Resultado Rede Neural Perfil FUDS - 35°C

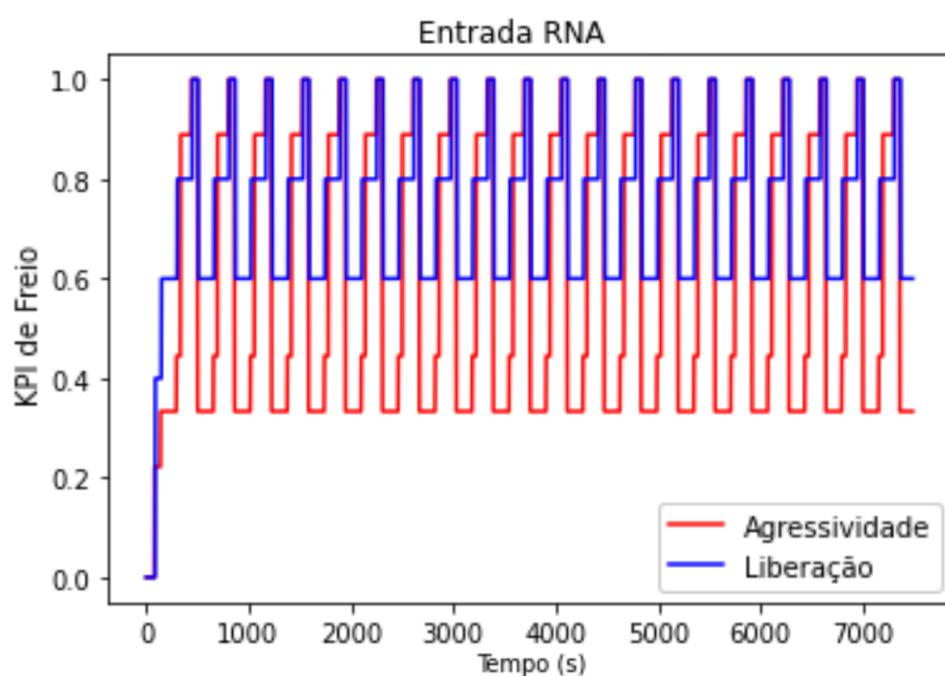


Figura 4.36 – Sinal FUDS - KPI de freio

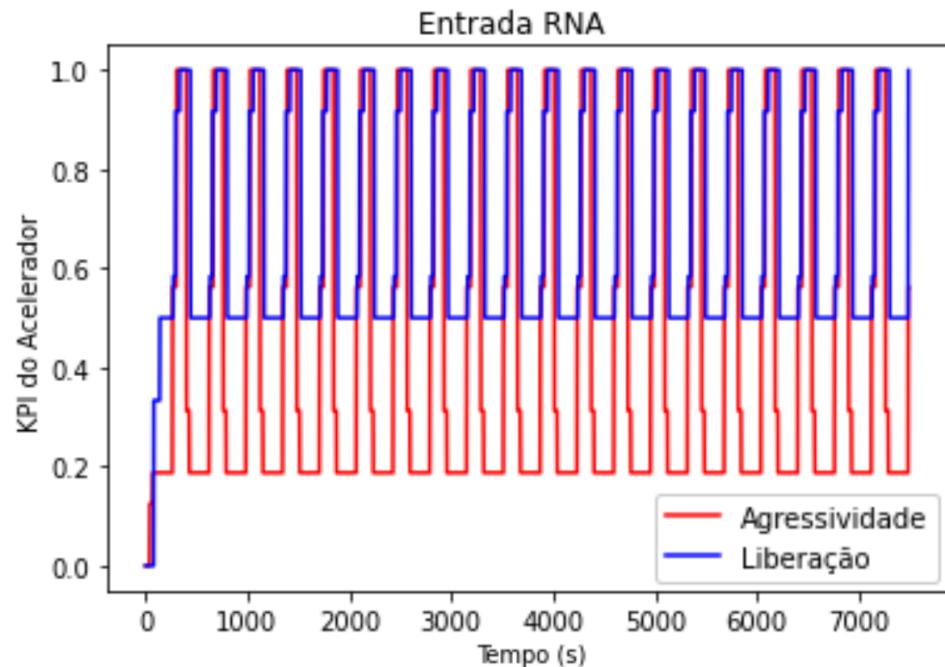


Figura 4.37 – Sinal FUDS - KPI do acelerador

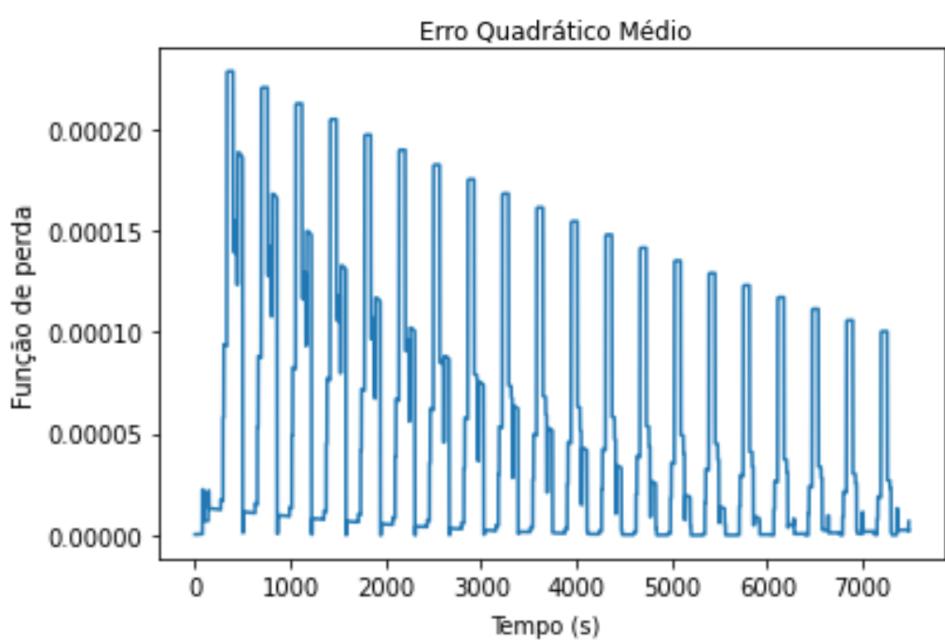


Figura 4.38 – Função de perda para o sinal FUDS à 35°C

as singularidades de valores médios entre pilotos com características diferentes de pilotagem, preservando o objetivo e a importância da sua utilização. Nas Figuras 4.39 e 4.40 é possível observar os novos sinais de KPI com a média móvel total aplicada. O novo resultado (Figura 4.41) mostrou que esta técnica melhorou a resposta da estimativa de SOC da RNA e reduziu o efeito da variação do KPI no erro do modelo (Figura 4.42).

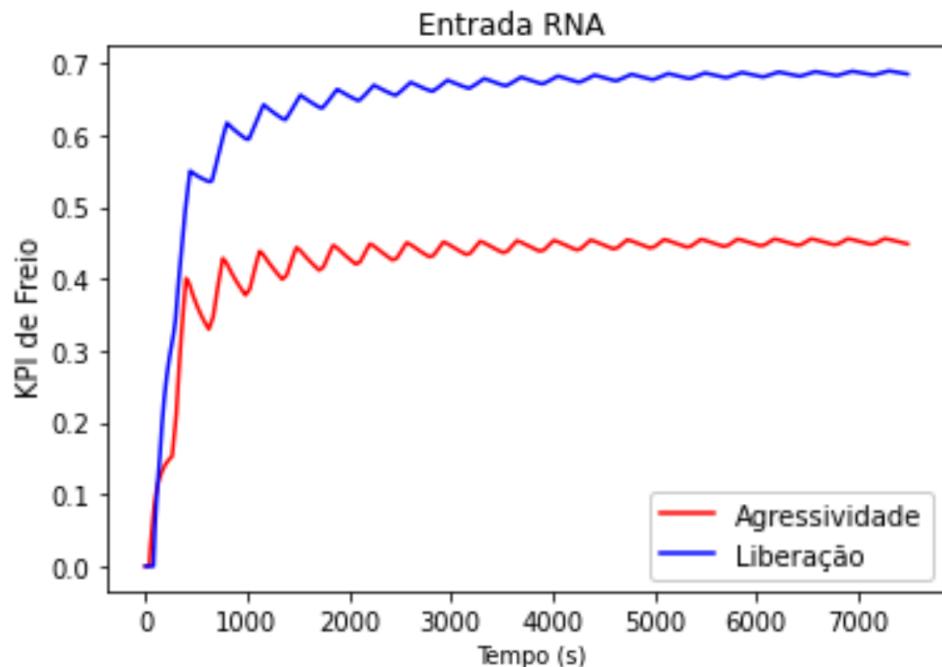


Figura 4.39 – Sinal FUDS - KPI de freio com média móvel

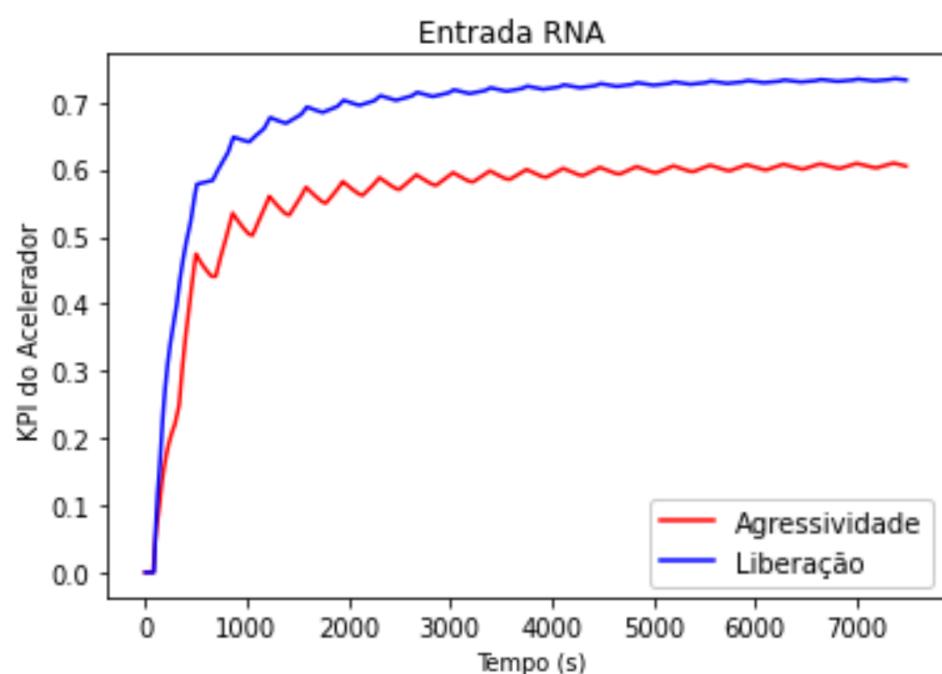


Figura 4.40 – Sinal FUDS - KPI do acelerador com média móvel

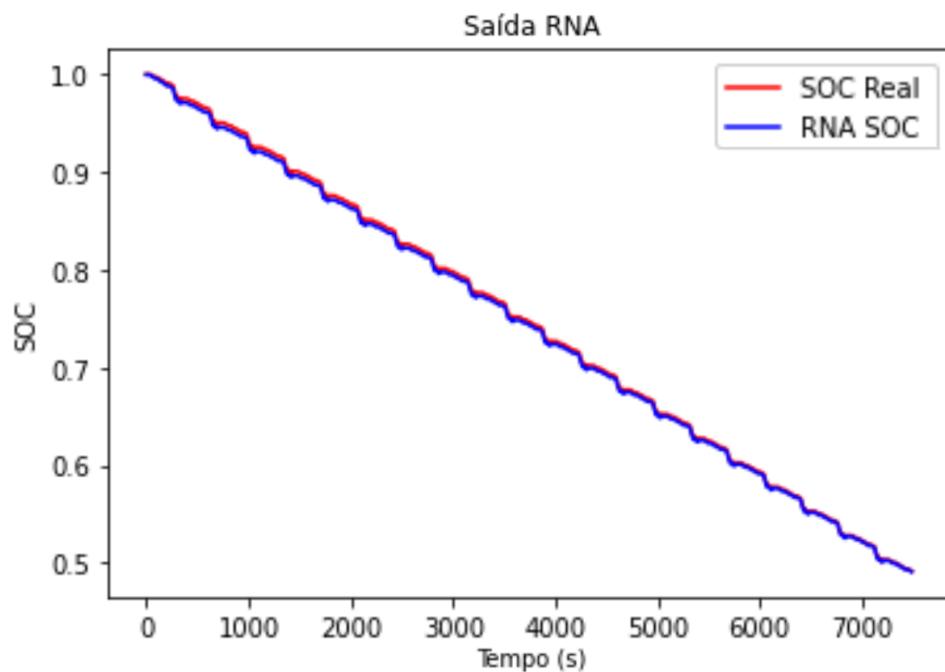


Figura 4.41 – Resultado da RNA com perfil FUDS e KPI com média móvel à 35°C

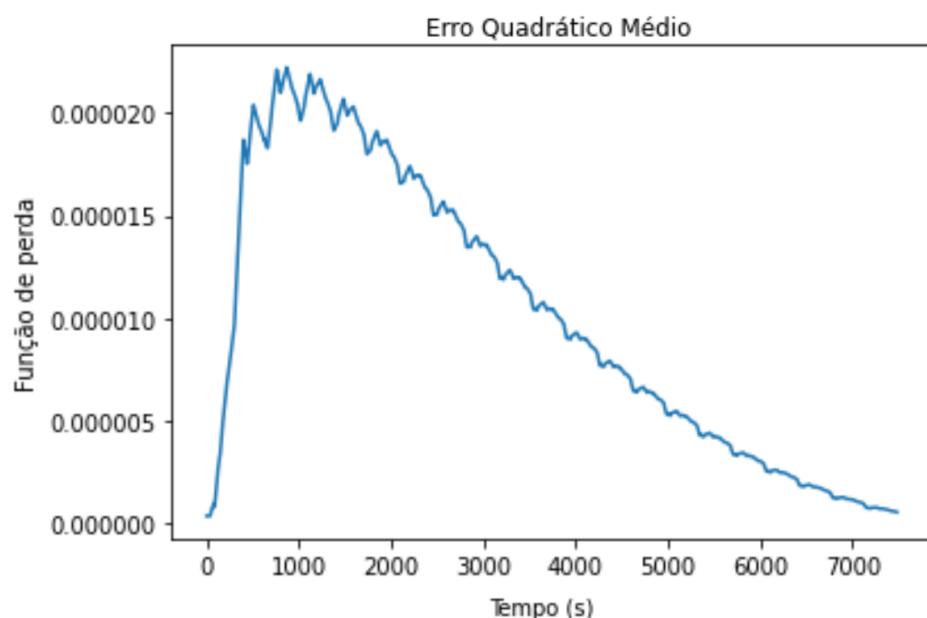


Figura 4.42 – Função de perda para o sinal FUDS e KPI com média móvel à 35°C

## 5 CONCLUSÃO

Podemos concluir que o modelo proposto, o Filtro de Kalman Ponto Sigma integrado a uma Rede Neural Artificial Profunda de um modelo parametrizado por Indicadores de Performance de um piloto, obteve resultados que demonstram que a linha de raciocínio adotado pode continuar a ser desenvolvida para aplicações específicas de *Motorsports*. Observamos que ao incluir os parâmetros intrínsecos a aplicação o algoritmo ganha maior robustez e precisão para a estimativa do Estado de Carga da Bateria. Atualmente temos modelos muito desenvolvidos na área eletroquímica, de modelos de circuitos equivalentes e também dos algoritmos baseados em dados como o Filtro de Kalman e Rede Neural, mas é raro encontrar estudos incluindo parâmetros específicos da aplicação do problema. Este desenvolvimento mostrou um bom potencial nos modelos simulados e indica que devemos continuar investindo em pesquisas neste sentido, não somente para este exemplo, mas para muitas aplicações onde modelos híbridos baseados em teoria e aplicação podem evoluir o desempenho do algoritmo. A utilização da expertise da aplicação real em conjunto com um modelo eficaz ao extrair os parâmetros da bateria e estimar o SOC poderá levar a excelência de performance do algoritmo para a aplicação embarcada. Os resultados demonstram que existe um vasto campo de pesquisa que pode aprimorar os métodos de estimar o SOC de uma bateria de Lítio e ajudar a desenvolver e implementar novas tecnologias.

## 6 TRABALHOS FUTUROS

Os métodos híbridos apresentaram o maior potencial de desenvolvimento de todos os métodos estudados durante este trabalho. A fusão de conceitos físicos e químicos através de equações diferenciais e uma grande quantidade de dados simulados fornecidos ao modelo pode trazer muitos benefícios de desempenho ao algoritmo. Os métodos teóricos baseado nas equações trazem maior capacidade de solucionar diferentes tipos de problemas em condições muito diferentes, o que é uma desvantagem dos métodos orientados a dados que costumam ter um bom desempenho apenas para dados com características semelhantes aos que foram utilizados durante o treinamento. Em contra partida, os métodos orientados a dados são mais eficientes para obter uma resposta do problema, sendo mais recomendados para aplicações embarcadas. Poder unificar esses métodos utilizando suas características positivas para melhorarem o desempenho final do modelo é uma área de pesquisa que acreditamos que possa ser o futuro das questões aplicadas a obtenção de parâmetros de saúde de uma célula de bateria. Por isso um modelo híbrido que contenha a densidade de informação expressa por equações diferenciais de um modelo eletroquímico, juntamente com uma grande quantidade de dados de treinamento disponíveis pelos métodos de simulação em laboratório é um potencial caminho de desenvolvimento futuro.

Outro ponto importante a ser considerado é a utilização de expertise da aplicação prática do problema no modelo proposto, assim como neste presente trabalho. A grande maioria dos estudos está focada, por exemplo, em aprimorar os algoritmos de estimativa de estados ou modelos equivalentes que representem melhor a dinâmica de uma bateria. Isso é correto conceitualmente para aprimorar o desempenho dos métodos propostos, porém ainda é necessário adicionar parâmetros intrínsecos ao problema de aplicação. Este tipo de variável relaciona condições que somente serão apresentadas durante a aplicação do método, mas que estão diretamente relacionadas com o resultado estimado do problema. Em *Motorsports* existem muitos fatores que podem ser adicionados a solução do problema de obtenção da estimativa de SOC da bateria. Além dos KPIs apresentados, um fator que não foi incluso no método e que está relacionado é o desgaste de pneu. Ele está diretamente relacionado com o desempenho do veículo durante a corrida, velocidades mínimas e capacidade de tração em curvas, o que está diretamente relacionado com a quantidade de acelerador que será solicitado pelo piloto ao sistema de trem

de força alimentado pelas baterias. Além de influenciar no cálculo este fator muda conforme o número de voltas completadas na corrida e pode se alterar novamente a qualquer momento se existir uma troca de pneus durante um *Pit Stop*. Este fator costuma ser de conhecimento da área de desenvolvimento de dinâmica veicular de um projeto e pode ser incorporado como variável de entrada, por exemplo, de uma RNA que irá modelar sua relação com a variação de SOC apresentada.

Quanto aos tipos de arquiteturas de uma RNA, as recorrentes (*Recurrent Neural Network*), mais especificamente uma memória de curto prazo longa (*Long Short Term Memory*) podem desempenhar melhores resultados na solução do modelo proposto. A característica de memória da RNA nesta arquitetura pode ser uma vantagem durante as diferentes etapas de uma corrida. Considerar o histórico recente dos dados para estimar o resultado pode ser positivo e ajudar o algoritmo a entender as mudanças de fatores como a intervenção de um carro de segurança ou um piloto que acaba de sair do trânsito, já que os momentos que sucedem estes costumam ter diferentes perfis de consumo e características de descarga de corrente que serão solicitados pelo piloto.

Como o desenvolvimento de novas tecnologias para obtenção de parâmetros de saúde para serem utilizados nos módulos de gerenciamento de baterias, os trabalhos futuros devem, em paralelo, desenvolver novas tecnologias com conhecimento aplicado específico à aplicações práticas de *Motorsports*, como a Fórmula E (Figura 6.1), para melhorar o desempenho dos algoritmos.

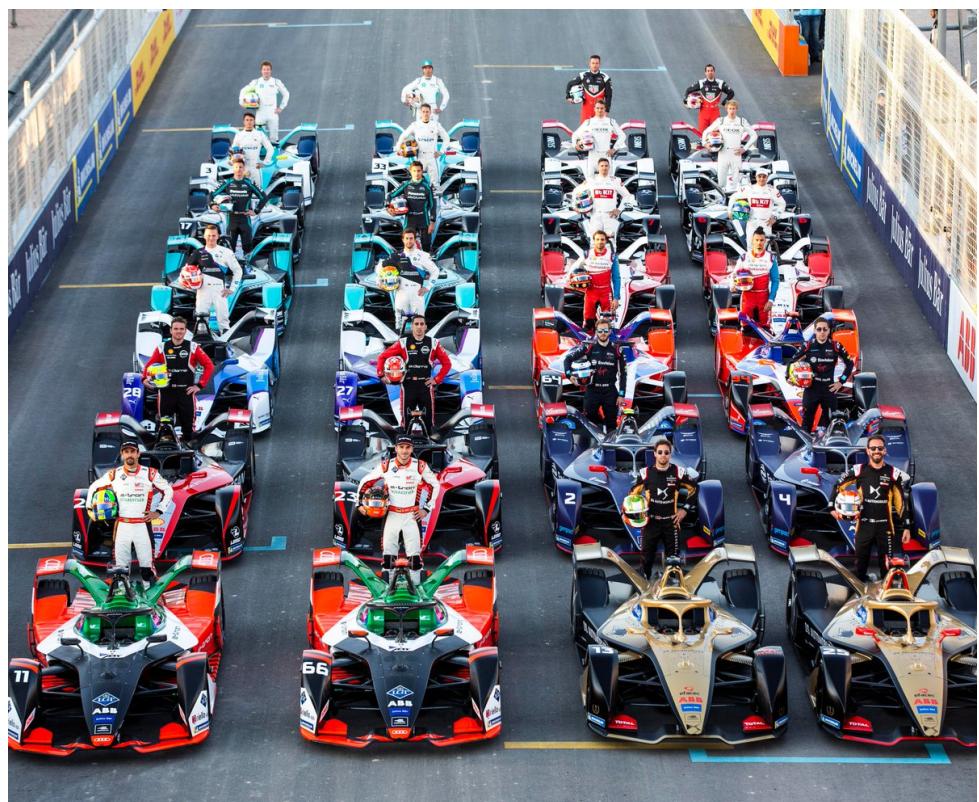


Figura 6.1 – Fórmula E temporada 2019 - 2020

## REFERÊNCIAS

- Chang, W.-Y. The state of charge estimating methods for battery: A review. **ISRN Applied Mathematics**, v. 2013, 2013.
- Chen, L.; Xu, R.; Rao, W.; Li, H.; Wang, Y.-P.; Jiang, T. Y. H.-B. Electrochemical model parameter identification of lithium-ion battery with temperature and current dependence. **Int. J. Electrochem. Sci**, v. 14, p. 4124–4143, 2019.
- Chi, Q. C.; Bole Brian ang Hogge, E.; Vazquez, S.; Daigle, M.; Celaya, J.; Weber, A. Battery charge depletion prediction on an electric aircraft. 2013.
- Eddahech, A.; Briat, O.; Bertrand, N.; Deletage, J.-Y.; Vinassa, J.-M. Behavior and state-of-health monitoring of li-ion batteries using impedance spectroscopy and recurrent neural networks. **International Journal of Electrical Power & Energy Systems**, v. 42, n. 1, p. 487–494, 2012.
- Fang, L.; Li, J.; Peng, B. Online estimation and error analysis of both soc and soh of lithium-ion battery based on dekf method. **Energy Procedia**, v. 158, p. 3008–3013, 2019.
- Farfán, J. F.; Palacios, K.; Ulloa, J.; Avilés, A. A hybrid neural network-based technique to improve the flow forecasting of physical and data-driven models: Methodology and case studies in andean watersheds. **Journal of Hydrology: Regional Studies**, v. 27, p. 100652, 2020.
- Guo, L.; Li, J.; Fu, Z. Lithium-ion battery soc estimation and hardware-in-the-loop simulation based on ekf. **Energy Procedia**, v. 158, p. 2599–2604, 2019.
- Hannan, M. A.; Lipu, M. S. H.; Hussain, A.; Saad, M. H.; Ayob, A. Neural network approach for estimating state of charge of lithium-ion battery using backtracking search algorithm. **Ieee Access**, v. 6, p. 10069–10079, 2018.
- He, W.; Pecht, M.; Flynn, D.; Dinmohammadi, F. A physics-based electrochemical model for lithium-ion battery state-of-charge estimation solved by an optimised projection-based method and moving-window filtering. **Energies**, v. 11, n. 8, p. 2120, 2018.
- He, W.; Williard, N.; Chen, C.; Pecht, M. State of charge estimation for li-ion batteries using neural network modeling and unscented kalman filter-based error cancellation. **International Journal of Electrical Power & Energy Systems**, v. 62, p. 783–791, 2014.
- Hosseinzadeh, E.; Marco, J.; Jennings, P. Electrochemical-thermal modelling and optimisation of lithium-ion battery design parameters using analysis of variance. **Energies**, v. 10, n. 9, p. 1278, 2017.
- How, D. N.; Hannan, M.; Lipu, M. H.; Ker, P. J. State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review. **IEEE Access**, v. 7, p. 136116–136136, 2019.
- Karpatne, A.; Watkins, W.; Read, J.; Kumar, V. Physics-guided neural networks (pgnn): An application in lake temperature modeling. **arXiv preprint arXiv:1710.11431**, 2017.

- Kharazmi, E.; Zhang, Z.; Karniadakis, G. Variational physics-informed neural networks for solving partial differential equations. **arXiv preprint arXiv:1912.00873**, 2019.
- Li, J.; Cheng, Y.; Ai, L.; Jia, M.; Du, S.; Yin, B.; Woo, S.; Zhang, H. 3d simulation on the internal distributed properties of lithium-ion battery with planar tabbed configuration. **Journal of Power Sources**, v. 293, p. 993–1005, 2015.
- Ning, B.; Xu, J.; Cao, B.; Wang, B.; Xu, G. A sliding mode observer soc estimation method based on parameter adaptive battery model. **Energy Procedia**, v. 88, p. 619–626, 2016.
- Nunes, T. M. S.; Oliveira Queiroz, F. F. de; Villanueva, J. M. M.; Macedo, E. C. T. **Modeling and estimation of the state of charge of lithium-ion battery based on artificial neural network**. [S.l.], 2018.
- Plett, G. L. **Battery management systems, Volume I: Battery modeling**. [S.l.]: Artech House, 2015.
- Raissi, M.; Perdikaris, P.; Karniadakis, G. E. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. **arXiv preprint arXiv:1711.10561**, 2017.
- Saxena, A.; Celaya, J. R.; Roychoudhury, I.; Saha, S.; Saha, B.; Goebel, K. Designing data-driven battery prognostic approaches for variable loading profiles: Some lessons learned. In: **European Conference of Prognostics and Health Management Society**. [S.l.: s.n.], 2012. p. 72–732.
- Singh, K. V.; Bansal, H. O.; Singh, D. Hardware-in-the-loop implementation of anfis based adaptive soc estimation of lithium-ion battery for hybrid vehicle applications. **Journal of Energy Storage**, v. 27, p. 101124, 2020.
- Stefanopoulou, A.; Kim, Y. System-level management of rechargeable lithium-ion batteries. In: **Rechargeable Lithium Batteries**. [S.l.]: Elsevier, 2015. p. 281–302.
- Tulsky, A.; Tsai, Y.; Gopaluni, R. B.; Braatz, R. D. State-of-charge estimation in lithium-ion batteries: A particle filter approach. **Journal of Power Sources**, v. 331, p. 208–223, 2016.
- Wang, Y.; Yang, D.; Zhang, X.; Chen, Z. Probability based remaining capacity estimation using data-driven and neural network model. **Journal of Power Sources**, v. 315, p. 199–208, 2016.
- Xing, Y.; He, W.; Pecht, M.; Tsui, K. L. State of charge estimation of lithium-ion batteries using the open-circuit voltage at various ambient temperatures. **Applied Energy**, v. 113, p. 106–115, 2014.
- Yang, D.; Wang, Y.; Pan, R.; Chen, R.; Chen, Z. A neural network based state-of-health estimation of lithium-ion battery in electric vehicles. **Energy Procedia**, v. 105, p. 2059–2064, 2017.
- Zahid, T.; Li, W. A comparative study based on the least square parameter identification method for state of charge estimation of a lifepo4 battery pack using three model-based algorithms for electric vehicles. **Energies**, v. 9, n. 9, p. 720, 2016.

Zhang, C.; Allafi, W.; Dinh, Q.; Ascencio, P.; Marco, J. Online estimation of battery equivalent circuit model parameters and state of charge using decoupled least squares technique. **Energy**, v. 142, p. 678–688, 2018.

Zhang, L.; Li, K.; Du, D.; Zhu, C.; Zheng, M. A sparse least squares support vector machine used for soc estimation of li-ion batteries. **IFAC-PapersOnLine**, v. 52, n. 11, p. 256–261, 2019.

Zhang, L.; Wang, L.; Lyu, C.; Li, J.; Zheng, J. Non-destructive analysis of degradation mechanisms in cycle-aged graphite/licoo<sub>2</sub> batteries. **Energies**, v. 7, n. 10, p. 6282–6305, 2014.

Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R. X. Deep learning and its applications to machine health monitoring. **Mechanical Systems and Signal Processing**, v. 115, p. 213–237, 2019.

## **Anexos**

## ANEXO A – ALGORITMOS EM MATLAB

Aqui vamos colocar os principais algoritmos utilizados no projeto e que referenciados neste trabalho feitos na linguagem Matlab. Todos eles possuem comentários explicando pontos importantes e caso exista o desejo você pode acessar o repositório do GitHub com todos estes arquivos e também os módulos adicionais que compõem tudo o que foi desenvolvido para chegar nos resultados deste trabalho, basta acessar o link a seguir: GitHub

### A.1 Algoritmos da Tensão de Circuito Aberto

#### A.1.1 runProcessOCV.m

```

1 % Algoritmo OCV
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %
7 %%%%%%%%%%%%%%
8 %
9 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
10 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
11 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
12 % It is provided "as is", without express or implied warranty, for
13 % educational and informational purposes only.
14 % This file is provided as a supplement to: Plett, Gregory L., "Battery
15 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
16 %
17 %%%%%%%%%%%%%%
18 %
19 %
20 %
21 
```

```

22 clear all
23 cellIDs = {'A123'}; % Identificação da célula A123
24 temps = {[ -25 -15 -5 5 15 25 35 45]}; % Temperaturas de cada teste
25 % tensão máxima e mínima de cada célula, usado para plotar os resultados
26 minV = [2.00];
27 maxV = [3.75];
28
29 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
30 % Antes de iniciar o algoritimo é necessário que se faça um pré
31 % processamento dos dados extraídos do teste em laboratório separando cada
32 % passo do processo com % os dados de tempo (s), passo, corrente (A),
33 % tensão (V), corrente de descarga (Ah) e corrente de carga (Ah).
34 %
35 % -----
36 % Carrega e processa os dados de teste
37 % -----
38 for theID = 1:length(cellIDs), % Loop varrendo todos os topos de célula (
39 % nessa aplicação só temos A123)
40 dirname = cellIDs{theID}; cellID = dirname;
41 ind = find(dirname == '_'); % lógica para se houver um "_", excluir
42 if ~isempty(ind), dirname = dirname(1:ind-1); end
43 OCVDir = sprintf('%s_OCV',dirname); % local onde vamos encontrar os
44 % arquivos
45 if ~exist(OCVDir,'dir'),
46 error(['Folder "%s" not found in current folder.\n' ...
47 'Please change folders so that "%s" is in the current '...
48 'folder and re-run runProcessOCV.'],OCVDir,OCVDir);
49 end
50
51 filetemps = temps{theID}(:); % arquivos de cada temperatura
52 numtemps = length(filetemps); % número de conjunto de dados
53 data = zeros([0 numtemps]); % Inicializa a variável de dados vazia
54
55 for k = 1:numtemps, % Loop varrendo todas as temperaturas (-25,
56 % -15, -5, 5, 15, 25, 35, 45)
57 if filetemps(k) < 0, % se a temperatura for negativa, então

```

```

52 filename = sprintf('%s/%s_OCV_N%02d.mat',... % procure por este arquivo (
53   nomenclatura para temperaturas negativas)
54 OCVDir,cellID,abs(filetemps(k)));
55 else % se a temperatura for positiva, então
56 filename = sprintf('%s/%s_OCV_P%02d.mat',... % procure por este arquivo (
57   nomenclatura para temperaturas positivas)
58 OCVDir,cellID,filetemps(k));
59 end
60 load(filename); % Leia o arquivo de dados OCV
61 data(k).temp = filetemps(k); % salvando a temperatura de teste
62 data(k).script1 = OCVData.script1; % salvando os 4 scripts
63 data(k).script2 = OCVData.script2;
64 data(k).script3 = OCVData.script3;
65 data(k).script4 = OCVData.script4;
66 end
67 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
68 % chamando a função "processOCV" para fazer o processamento dos dados
69 model = processOCV(data,cellID,minV(theID),maxV(theID),1);
70 save(sprintf('%smodel-ocv.mat',cellID),'model'); % salvando o arquivo
71 end

```

### A.1.2 processOCV.m

```

1 % Algoritmo OCV
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 % -----
15 %

```

```

16 % função = processOCV(data,cellID,minV,maxV,savePlots)
17 %
18 % Entradas:
19 %   data = dados de teste de célula
20 %   cellID = identificador do tipo de célula
21 %   minV = tensão mínima definida para relação OCV
22 %   maxV = tensão máxima definida para relação OCV
23 %   savePlots = 0 or 1 ... defina como "1" para salvar gráficos como
24 %   arquivos
25 % Output:
26 %   model = estrutura dos dados
27 %
28 % Nota técnica: O algoritmo "makeMATfiles.m" deve ser codado antes
29 %
30 % A seguir as etapas de cada um dos 4 scripts do teste OCV:
31 %   Script 1 (câmara térmica ajustada para temperatura de teste):
32 %       Step 1: Repouso a @100% SOC para aclimatar à temperatura de teste
33 %       Step 2: Descarga com taxa baixa (ca. C/30) até a tensão mínima
34 %       Step 3: Repouso
35 %   Script 2 (câmara térmica ajustada para 25°C):
36 %       Step 1: Repouso a @0% SOC para aclimatar à 25°C
37 %       Step 2: Descarga para a tensão mínima (ca. C/3)
38 %       Step 3: Repouso
39 %       Step 4: Tensão constante igual a Vmín
40 %       Steps 5-7: Aplicar um perfil de corrente oscilatório utilizado em
41 %                   desmagnetização
42 %       para minimizar o efeito da histerese
43 %       Step 8: Repouso
44 %       Step 9: Tensão constante em Vmin por 15 minutos
45 %       Step 10: Repouso
46 %   Script 3 (câmara térmica ajustada para temperatura de teste):
47 %       Step 1: Repouso a @0% SOC para aclimatar à temperatura de teste
48 %       Step 2: Carregar com taxa baixa (ca. C/30) até a tensão máxima
49 %       Step 3: Repouso
50 %   Script 4 (câmara térmica ajustada para 25°C):
51 %       Step 1: Repouso a @100% SOC para aclimatar à 25°C
52 %       Step 2: Carrega para a tensão máxima (ca. C/3)
53 %       Step 3: Repouso
54 %       Step 4: Tensão constante igual a Vmáx

```

```

53 % Steps 5-7: Aplicar um perfil de corrente oscilatório utilizado em
54 % desmagnetização
55 % para minimizar o efeito da histerese
56 % Step 8: Repouso
57 % Step 9: Tensão constante em Vmáx por 15 minutos
58 % Step 10: Repouso
59 %
60 function model=processOCV(data,cellID,minV,maxV,savePlots)
61 filetemps = [data.temp]; filetemps = filetemps(:);
62 numtemps = length(filetemps);
63
64 % Primeiro procura por testes à 25°C para calcular a eficiência de
65 % Coulomb e a capacidade nesta temperatura antes de continuar
66 % (necessário)
67 ind25 = find(filetemps == 25);
68 if isempty(ind25),
69 error('Must have a test at 25degC');
70 end
71 not25 = find(filetemps ~= 25);
72
73 %%%%%%
74 % Após o tratamento dos dados o algoritimo é iniciado inserindo a somatória
    % dos valores de corrente de descarga e carga nos vetores totDisAh e
    % totChgAh para
75 % calcular a eficiência de Coulomb (eta). Com o valor de eta o próximo
    % passo é calcular a capacidade total Q aplicando a eficiência (eta) na
    % corrente de carga % e fazendo a subtração da somatória da corrente de
    % descarga com a corrente de carga, como mostra a equação 3.5.
76 %
77 % -----
78 % Processa dados à 25°C para encontrar a relação entre OCV e eta25
79 % -----
80 SOC = 0:0.005:1; % define os pontos de saída para esta etapa de SOC
81 filedata = zeros([0 length(data)]);
82 eta = zeros(size(filetemps)); % eficiência de Coulomb
83 Q = zeros(size(filetemps)); % capacidade total aparente da bateria
84 k = ind25;
85

```

```

86 totDisAh = data(k).script1.disAh(end) + ... % calcula o total de Ah
87 data(k).script2.disAh(end) + ... % que foram descarregados
88 data(k).script3.disAh(end) + ...
89 data(k).script4.disAh(end);
90 totChgAh = data(k).script1.chgAh(end) + ... % calcula o total de Ah
91 data(k).script2.chgAh(end) + ... % que foram carregados
92 data(k).script3.chgAh(end) + ...
93 data(k).script4.chgAh(end);
94 eta25 = totDisAh/totChgAh; eta(k) = eta25; % calcula a eficiência de
   Coulomb à 25°C
95 data(k).script1.chgAh = data(k).script1.chgAh*eta25; % ajusta o valor de
96 data(k).script2.chgAh = data(k).script2.chgAh*eta25; % carga em Ah por
   eta25
97 data(k).script3.chgAh = data(k).script3.chgAh*eta25; % em todos os
   scripts
98 data(k).script4.chgAh = data(k).script4.chgAh*eta25;
99 %%%%%%%%%%%%%%
100
101 % capacidade da célula à 25°C (deve ser essencialmente a
102 % mesma em todas as temperaturas, mas estamos computando-as
103 % individualmente para verificar isso)
104 Q25 = data(k).script1.disAh(end) + data(k).script2.disAh(end) - ...
105 data(k).script1.chgAh(end) - data(k).script2.chgAh(end);
106 Q(k) = Q25;
107
108 %%%%%%%%%%%%%%
109 % O R0 é calculado obtendo a variação de tensão no início da carga e
   descarga. Este cálculo é feito subtraindo da tensão imediatamente antes
   do início da
110 % descarga a primeira tensão após a descarga, consequentemente o mesmo
   processo para a carga.
111
112 indD = find(data(k).script1.step == 2);           % etapa de descarga lenta
113 IR1Da = data(k).script1.voltage(indD(1)-1) - ... % queda de tensão i*R no
114 data(k).script1.voltage(indD(1));                 % inicio da descarga
115 IR2Da = data(k).script1.voltage(indD(end)+1) - ... % o mesmo que no final
116 data(k).script1.voltage(indD(end));               % da descarga
117 indC = find(data(k).script3.step == 2);           % etapa de carregamento lento
118 IR1Ca = data(k).script3.voltage(indC(1)) - ...    % queda de tensão i*R no

```

```

119 data(k).script3.voltage(indC(1)-1);           % inicio da carga
120 IR2Ca = data(k).script3.voltage(indC(end)) - ... % o mesmo que no final
121 data(k).script3.voltage(indC(end)+1);           % da carga
122 IR1D = min(IR1Da,2*IR2Ca); IR2D = min(IR2Da,2*IR1Ca);      % definindo as
123 IR1C = min(IR1Ca,2*IR2Da); IR2C = min(IR2Ca,2*IR1Da);      % bordas/limites
124
125 % Em seguida obtemos um valor de tensão considerando o efeito do R0,
126 % subtraindo o efeito durante a carga e somando durante a descarga, e
127 % calculamos o valor do % SOC.
128
129 blend = (0:length(indD)-1)/(length(indD)-1);
130 IRblend = IR1D + (IR2D-IR1D)*blend(:);
131 disV = data(k).script1.voltage(indD) + IRblend;
132 disZ = 1 - data(k).script1.disAh(indD)/Q25;      % calculando o SOC
133 disZ = disZ + (1 - disZ(1));                      % em cada ponto
134 filedata(k).disZ = disZ;
135 filedata(k).disV = data(k).script1.voltage(indD);
136
137 blend = (0:length(indC)-1)/(length(indC)-1);
138 IRblend = IR1C + (IR2C-IR1C)*blend(:);
139 chgV = data(k).script3.voltage(indC) - IRblend;
140 chgZ = data(k).script3.chgAh(indC)/Q25;      % calculando o SOC
141 chgZ = chgZ - chgZ(1);                      % em cada ponto
142 filedata(k).chgZ = chgZ;
143 filedata(k).chgV = data(k).script3.voltage(indC);
144 %%%%%%%%%%%%%%
145 % Calculando a diferença de tensão entre carga e descarga à @50% SOC
146 % força a curva compensada i*R para passar a meio caminho entre
147 % cada carga e descarga neste ponto
148 deltaV50 = interp1(chgZ,chgV,0.5) - interp1(disZ,disV,0.5);
149 ind = find(chgZ < 0.5);
150 vChg = chgV(ind) - chgZ(ind)*deltaV50;
151 zChg = chgZ(ind);
152 ind = find(disZ > 0.5);
153 vDis = flipud(disV(ind) + (1 - disZ(ind))*deltaV50);
154 zDis = flipud(disZ(ind));
155 filedata(k).rawocv = interp1([zChg; zDis],[vChg; vDis],SOC,...  

   'linear','extrap');

```

```

156
157 % "rawocv" agora tem nosso melhor palpite do verdadeiro OCV nesta
158 % temperatura
159 filedata(k).temp = data(k).temp;
160
161 % -----
162 % Processando as outras temperaturas para encontrar a relação entre
163 % OCV e eta (eficiência de Coulomb)
164 % Tudo o que se segue é igual a 25°C, exceto que precisamos
165 % compensar diferentes eficiências coulombicas eta em diferentes
166 % temperaturas
167 % -----
168
169 for k = not25',
170 data(k).script2.chgAh = data(k).script2.chgAh*eta25;
171 data(k).script4.chgAh = data(k).script4.chgAh*eta25;
172 eta(k) = (data(k).script1.disAh(end) + ...
173 data(k).script2.disAh(end) + ...
174 data(k).script3.disAh(end) + ...
175 data(k).script4.disAh(end) - ...
176 data(k).script2.chgAh(end) - ...
177 data(k).script4.chgAh(end))/ ...
178 (data(k).script1.chgAh(end) + ...
179 data(k).script3.chgAh(end));
180 data(k).script1.chgAh = eta(k)*data(k).script1.chgAh;
181 data(k).script3.chgAh = eta(k)*data(k).script3.chgAh;
182
183 Q(k) = data(k).script1.disAh(end) + data(k).script2.disAh(end) ...
184 - data(k).script1.chgAh(end) - data(k).script2.chgAh(end);
185 indD = find(data(k).script1.step == 2);
186 IR1D = data(k).script1.voltage(indD(1)-1) - ...
187 data(k).script1.voltage(indD(1));
188 IR2D = data(k).script1.voltage(indD(end)+1) - ...
189 data(k).script1.voltage(indD(end));
190 indC = find(data(k).script3.step == 2);
191 IR1C = data(k).script3.voltage(indC(1)) - ...
192 data(k).script3.voltage(indC(1)-1);
193 IR2C = data(k).script3.voltage(indC(end)) - ...
194 data(k).script3.voltage(indC(end)+1);

```

```

195 IR1D = min(IR1D,2*IR2C); IR2D = min(IR2D,2*IR1C);
196 IR1C = min(IR1C,2*IR2D); IR2C = min(IR2C,2*IR1D);
197
198 blend = (0:length(indD)-1)/(length(indD)-1);
199 IRblend = IR1D + (IR2D-IR1D)*blend(:);
200 disV = data(k).script1.voltage(indD) + IRblend;
201 disZ = 1 - data(k).script1.disAh(indD)/Q25;
202 disZ = disZ + (1 - disZ(1));
203 filedata(k).disZ = disZ;
204 filedata(k).disV = data(k).script1.voltage(indD);
205
206 blend = (0:length(indC)-1)/(length(indC)-1);
207 IRblend = IR1C + (IR2C-IR1C)*blend(:);
208 chgV = data(k).script3.voltage(indC) - IRblend;
209 chgZ = data(k).script3.chgAh(indC)/Q25;
210 chgZ = chgZ - chgZ(1);
211 filedata(k).chgZ = chgZ;
212 filedata(k).chgV = data(k).script3.voltage(indC);
213
214 deltaV50 = interp1(chgZ,chgV,0.5) - interp1(disZ,disV,0.5);
215 ind = find(chgZ < 0.5);
216 vChg = chgV(ind) - chgZ(ind)*deltaV50;
217 zChg = chgZ(ind);
218 ind = find(disZ > 0.5);
219 vDis = flipud(disV(ind) + (1 - disZ(ind))*deltaV50);
220 zDis = flipud(disZ(ind));
221 filedata(k).rawocv = interp1([zChg; zDis],[vChg; vDis],SOC,...
222 'linear','extrap');
223
224 filedata(k).temp = data(k).temp;
225 end
226
227 % -----
228 % Usando os dados SOC versus OCV disponíveis em cada temperatura
229 % para calcular uma relação OCV0 e OCVrel
230 % -----
231 % Primeiro, compile as voltagens e temperaturas em matrizes únicas
232 Vraw = []; temps = [];
233 for k = 1:numtemps,

```

```

234 if filedata(k).temp > 0,
235 Vraw = [Vraw; filedata(k).rawocv];
236 temps = [temps; filedata(k).temp];
237 end
238 end
239 numtempskept = size(Vraw,1);
240
241 % use os mínimos quadrados lineares para determinar o melhor valor para
242 % OCV à 0°C e em seguida a mudança de OCV por grau
243 OCV0 = zeros(size(SOC)); OCVrel = OCV0;
244 H = [ones([numtempskept,1]), temps];
245 for k = 1:length(SOC),
246 X = H\Vraw(:,k); % fit OCV(z,T) = 1*OCV0(z) + T*OCVrel(z)
247 OCV0(k) = X(1);
248 OCVrel(k) = X(2);
249 end
250 model.OCV0 = OCV0;
251 model.OCVrel = OCVrel;
252 model.SOC = SOC;
253
254 % -----
255 % Faça SOC0 e SOCrel
256 % Faça o mesmo tipo de análise para encontrar SOC como uma função de OCV
257 % -----
258 z = -0.1:0.01:1.1; % test soc vector
259 v = minV-0.01:0.01:maxV+0.01;
260 socs = [];
261 for T = filetemps',
262 v1 = OCVfromSOCTemp(z,T,model);
263 socs = [socs; interp1(v1,z,v)];
264 end
265
266 SOC0 = zeros(size(v)); SOCrel = SOC0;
267 H = [ones([numtemps,1]), filetemps];
268 for k = 1:length(v),
269 X = H\socs(:,k); % fit SOC(v,T) = 1*SOC0(v) + T*SOCrel(v)
270 SOC0(k) = X(1);
271 SOCrel(k) = X(2);
272 end

```

```

273 model.OCV = v;
274 model.SOC0 = SOC0;
275 model.SOCrel = SOCrel;
276
277 % -----
278 % Salvando outros dados na estrutura
279 % -----
280 model.OCVeta = eta;
281 model.OCVQ = Q;
282 model.name = cellID;
283
284 % -----
285 % Plotando alguns dados
286 % -----
287 for k = 1:numtemps,
288 figure;
289 plot(100*SOC,OCVfromSOCtemp(SOC,filedata(k).temp,model),...
290 100*SOC,filedata(k).rawocv); hold on
291 xlabel('SOC (%)'); ylabel('OCV (V)'); ylim([minV-0.1 maxV+0.1]);
292 title(sprintf('%s OCV relationship at temp = %d',...
293 cellID,filedata(k).temp)); xlim([0 100]);
294 err = filedata(k).rawocv - ...
295 OCVfromSOCtemp(SOC,filedata(k).temp,model);
296 rmserr = sqrt(mean(err.^2));
297 text(2,maxV-0.15,sprintf('RMS error = %4.1f (mV)',...
298 rmserr*1000),'fontsize',14);
299 plot(100*filedata(k).disZ,filedata(k).disV,'k--','linewidth',1);
300 plot(100*filedata(k).chgZ,filedata(k).chgV,'k--','linewidth',1);
301 legend('Model prediction','Approximate OCV from data',...
302 'Raw measured data','location','southeast');
303
304 if savePlots,
305 if ~exist('OCV FIGURES','dir'), mkdir('OCV FIGURES'); end
306 if filetemps(k) < 0,
307 filename = sprintf('OCV FIGURES/%s_N%02d.png',...
308 cellID,abs(filetemps(k)));
309 else
310 filename = sprintf('OCV FIGURES/%s_P%02d.png',...
311 cellID,filetemps(k));

```

```

312 end
313 print(filename,'-dpng')
314 end
315 end
316 end

```

## A.2 Algoritmos do Teste Dinâmico

### A.2.1 runProcessDynamic.m

```

1 % Algoritmo DYN
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 %%%%%%%%%%%%%%
15 % -----
16 % script runProcessDynamic
17 %
18 % RUNPROCESSDYNAMIC: lê os arquivos de dados correspondentes aos testes
19 % dinâmicos da célula, executa PROCESSDYNAMIC e então salva o resultado
20 % do modelo ESC. Ele se baseia no SETUPDYNDATA para fornecer a lista de
21 % dados a ser processados
22
23 close all
24 setupDynData; % obtém a lista de dados a serem processados
25 numpoles = 1; % número de pares R-C no modelo final
26
27 for indID = 1:length(cellIDs), % Loop varrendo todos os tipos de célula (
28   % nesta aplicação só temos A123)
29   cellID = cellIDs{indID};      % obtém a célula identificada

```

```

29
30 % Leia o arquivo OCV, previamente processado pelo runProcessOCV
31 modelFile = sprintf('../OCV/%smodel-ocv.mat',cellID);
32 if ~exist(modelFile,'file'),
33 error(['File "%s" not found.\n' ...
34 'Please change folders so that "%s" points to a valid model '...
35 'file and re-run runProcessDynamic.' ],modelFile,modelFile);
36 end
37 load(modelFile);

38
39 % Leia o arquivo de dados MAT
40 data = zeros([0 length(mags{indID} > 0)]); dataInd = 0;
41 for indTemps = 1:length(mags{indID}), % leia todas as temperaturas
42 theMag = mags{indID}(indTemps);      % máxima taxa-C no arquivo de dados *
43     10
44 if theMag < 0,                         % omita esses arquivos de dados
45 continue
46 else                                     % armazene estes dados em "data"
47 dataInd = dataInd + 1;
48 end
49 if temps(indTemps) < 0, % se a temperatura for negativa, então
50 DYNPrefix = sprintf('%s_DYN/%s_DYN_%02d_N%02d',... % nomenclatura para
51     temperaturas negativas
52 cellID,cellID,theMag,abs(temps(indTemps)));
53 else                                     % se a temperatura for positiva, então
54 DYNPrefix = sprintf('%s_DYN/%s_DYN_%02d_P%02d',... % nomenclatura para
55     temperaturas positivas
56 cellID,cellID,theMag,temps(indTemps));
57 end
58 inFile = sprintf('%s.mat',DYNPrefix);
59 if ~exist(inFile,'file'),
60 error(['File "%s" not found.\n' ...
61 'Please change folders so that "%s" points to a valid data '...
62 'file and re-run runProcessDynamic.' ],inFile,inFile);
63 end
64 fprintf('Loading %s\n',inFile); load(inFile);
65 data(dataInd).temp      = temps(indTemps); % armazena temperatura
66 data(dataInd).script1 = DYNDATA.script1; % armazena os dados de cada
67 data(dataInd).script2 = DYNDATA.script2; % um dos três scripts

```

```

65 data(dataInd).script3 = DYNDATA.script3;
66 end
67
68 model = processDynamic(data,model,numpoles,1); % faz o "heavy lifting"
69 modelFile = sprintf('%smodel.mat',cellID); % salva o modelo otimizado
70 save(modelFile,'model'); % neste arquivo
71
72 % Plota os resultados de tensão do modelo à 25°C, mais o erro RMS de
73 % estimação de tensão entre 5% e 95% de SOC da célula
74 figure(10+indID);
75 indTemps = find(temp == 25);
76 [vk,rck,hk,zk,sik,OCV] = simCell(data(indTemps).script1.current, ...
77 temps(indTemps),1,model,1,zeros(numpoles,1),0);
78 tk = (1:length(data(indTemps).script1.current))-1;
79 plot(tk,data(indTemps).script1.voltage,tk,vk);
80 verr = data(indTemps).script1.voltage - vk';
81 v1 = OCVfromSOCtemp(0.95,temp(indTemps),model);
82 v2 = OCVfromSOCtemp(0.05,temp(indTemps),model);
83 N1 = find(data(indTemps).script1.voltage<v1,1,'first');
84 N2 = find(data(indTemps).script1.voltage<v2,1,'first');
85 if isempty(N1), N1=1; end; if isempty(N2), N2=length(verr); end
86 rmserr=sqrt(mean(verr(N1:N2).^2));
87 fprintf('RMS error of simCell @ 25 degC = %0.2f (mv)\n',rmserr*1000);
88 end

```

## A.2.2 processDynamic.m

```

1 % Algoritmo DYN
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery

```

```

13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 %-----%
15 %
16 % função processDynamic
17 %
18 % Nota técnica: PROCESSDYNAMIC assume que os testes com a bateria A123
19 % já foram executados para gerar os dados de entrada.
20 % "makeMATfiles.m" converte o arquivo de dados do Excel para o formato "
21 % MAT"
22 % os arquivos MATtem os campos de time, step, current, voltage,
23 % chgAh, e disAh para cada run do script.
24 %
25 % Os resultados destes scripts são necessários para cada temperatura.
26 % A seguir as etapas de cada um dos 3 scripts do teste DYN:
27 % Script 1 (câmara térmica ajustada para temperatura de teste):
28 % Step 1: Repouso à @100% SOC para aclimatar à temperatura de teste
29 % Step 2: Descarga à @1C para atingir @90% SOC
30 % Step 3: Executa repetidamente os perfis dinâmicos (e possivelmente
31 % repousos intermediários) até o SOC estar por volta de @10%
32 % Script 2 (câmara térmica ajustada para 25°C):
33 % Step 1: Repouso à @10% SOC para aclimatar à 25°C
34 % Step 2: Descarregue até a tensão mínima (ca. C/3)
35 % Step 3: Repouso
36 % Step 4: Tensão constante em Vmín (ca. C/30)
37 % Steps 5-7: Aplicar um perfil de corrente oscilatório utilizado em
38 % desmagnetização
39 % para minimizar o efeito da histerese
40 % Step 8: Repouso
41 % Script 3 (câmara térmica ajustada para 25°C):
42 % Step 2: Carregue à @1C para a tensão máxima
43 % Step 3: Repouso
44 % Step 4: Tensão constante em Vmáx (ca. C/30)
45 % Steps 5-7: Aplicar um perfil de corrente oscilatório utilizado em
46 % desmagnetização
47 % para minimizar o efeito da histerese
48 % Step 8: Repouso
%
% Todas as outras etapas (se presentes) são ignoradas por PROCESSDYNAMIC
% os passos entre os dados devem ser uniformes, nós assumimos um período

```

```

        de
49 % 1 segundo de amostra neste código
50 %
51 % Entradas:
52 % - data: uma array, com uma entrada por temperatura a ser processada.
53 %         uma das entradas da array deve estar a 2°C. Os campos de dados
54 %         são: temp (temperatura de teste), script1,
55 %         script 2 e script 3, onde estes últimos incluem dados
56 %         coletados de cada script. Os subcampos desses scripts
57 %         são os vetores:
58 %         current, voltage, chgAh e disAh
59 % - model: A saída do processoOCV, compreendendo o modelo OCV
60 % - numpoles: Número de pares R-C neste modelo
61 % - doHyst: 0 se não desejamos um modelo de histerese, 1 se desejamos
62 %
63 % Saída:
64 % - model: Um modelo modificado, que agora contém os campos dinâmicos
65 %         preenchidos.

66
67 function model = processDynamic(data,model,numpoles,doHyst)
68 global bestcost
69
70 % usado por "fminbnd" mais tarde
71 if exist('optimset.m','file')
72 options=optimset('TolX',1e-8,'TolFun',1e-8,'MaxFunEval',100000, ...
73 'MaxIter',1e6,'Jacobian','Off');
74 end
75
76 % -----
77 % Step 1: Calcula a capacidade e a eficiência de Coulomb para cada teste
78 % -----
79 alltemps = [data(:).temp];
80 alletas = 0*alltemps;
81 allQs = 0*alltemps;
82
83 ind25 = find(alltemps == 25);
84 if isempty(ind25),
85 error('Must have a test at 25degC');
86 end

```

```

87 not25 = find(alltemps ~= 25);
88
89 for k = ind25,
90 totDisAh = data(k).script1.disAh(end) + ...
91 data(k).script2.disAh(end) + ...
92 data(k).script3.disAh(end);
93 totChgAh = data(k).script1.chgAh(end) + ...
94 data(k).script2.chgAh(end) + ...
95 data(k).script3.chgAh(end);
96 eta25 = totDisAh/totChgAh;
97 data(k).eta = eta25; alletas(k) = eta25;
98 data(k).script1.chgAh = data(k).script1.chgAh*eta25;
99 data(k).script2.chgAh = data(k).script2.chgAh*eta25;
100 data(k).script3.chgAh = data(k).script3.chgAh*eta25;
101
102 Q25 = data(k).script1.disAh(end) + data(k).script2.disAh(end) - ...
103 data(k).script1.chgAh(end) - data(k).script2.chgAh(end);
104 data(k).Q = Q25; allQs(k) = Q25;
105 end
106 eta25 = mean(alletas(ind25));
107
108 for k = not25,
109 data(k).script2.chgAh = data(k).script2.chgAh*eta25;
110 data(k).script3.chgAh = data(k).script3.chgAh*eta25;
111 eta = (data(k).script1.disAh(end) + data(k).script2.disAh(end)+...
112 data(k).script3.disAh(end) - data(k).script2.chgAh(end)-...
113 data(k).script3.chgAh(end))/data(k).script1.chgAh(end);
114 data(k).script1.chgAh = eta*data(k).script1.chgAh;
115 data(k).eta = eta; alletas(k) = eta;
116
117 Q = data(k).script1.disAh(end) + data(k).script2.disAh(end) - ...
118 data(k).script1.chgAh(end) - data(k).script2.chgAh(end);
119 data(k).Q = Q; allQs(k) = Q;
120 end
121
122 model.temps = unique(alltemps); numTemps = length(model.temps);
123 model.etaParam = NaN(1,numTemps);
124 model.QParam = NaN(1,numTemps);
125 for k = 1:numTemps,
```

```

126 model.etaParam(k) = mean(alletas(alltemps == model.temps(k)));
127 model.QParam(k) = mean(allQs(alltemps == model.temps(k)));
128 end
129
130 % -----
131 % Step 2: Calcula o OCV para cada "descarga" do teste
132 % -----
133 for k = 1:length(data),
134 etaParam = model.etaParam(k);
135 etaik = data(k).script1.current;
136 etaik(etaik<0)= etaParam*etaik(etaik<0);
137 data(k).Z = 1 - cumsum([0,etaik(1:end-1)])*1/(data(k).Q*3600);
138 data(k).OCV = OCVfromSOCTemp(data(k).Z(:),alltemps(k),model);
139 end
140
141 % -----
142 % Step 3: Otimização
143 % -----
144 model.GParam = NaN(1,numTemps); % Parâmetro de histerese "gamma"
145 model.M0Param = NaN(1,numTemps); % Parâmetro de histerese "M0"
146 model.MParam = NaN(1,numTemps); % Parâmetro de histerese "M"
147 model.R0Param = NaN(1,numTemps); % Parâmetro de resistência "R0"
148 model.RCParam = NaN(numTemps,numpoles); % Constante de tempo
149 model.RParam = NaN(numTemps,numpoles); % Parâmetro Rk
150
151 for theTemp = 1:numTemps,
152 fprintf('Processing temperature %d\n',model.temps(theTemp));
153 bestcost = Inf;
154 % theGammas = 1:5000;
155 % theFit = zeros(size(theGammas));
156 % for indGamma = 1:length(theGammas),
157 %     theFit(indGamma) = optfn(theGammas(indGamma),data,model,...
158 %                             model.temps(theTemp),doHyst);
159 % end
160 % figure(4); plot(theFit); stop
161 if doHyst,
162 if exist('fminbnd.m','file'),
163 model.GParam(theTemp) = abs(fminbnd(@(x) optfn(x,data, ...
164 model,model.temps(theTemp),...

```

```

165 doHyst),1,250,options));
166 else
167 model.GParam(theTemp) = abs(gss(@(x) optfn(x,data, ...
168 model,model.temps(theTemp),...
169 doHyst),1,250,1e-8));
170 end
171 else
172 model.GParam(theTemp) = 0;
173 optfn(theGParam,data,model,model.temps(theTemp),doHyst);
174 end
175 [~,model] = minfn(data,model,model.temps(theTemp),doHyst);
176 end
177 return
178
179 % -----
180 % Este "minfn" funciona para o modelo de célula de autocorreção
181 % aprimorado
182 % (ESC)
183 % -----
184 function cost=optfn(theGParam,data,model,theTemp,doHyst)
185 global bestcost
186
187 model.GParam(model.temps == theTemp) = abs(theGParam);
188 [cost,model] = minfn(data,model,theTemp,doHyst);
189 if cost<bestcost, % atualize o gráfico dos parâmetros do modelo para cada
190 % melhoria
191 bestcost = cost;
192 disp('Best ESC model values yet!');
193 figure(3); theXlim = [min(model.temps) max(model.temps)];
194 plot(model.temps,model.QParam);
195 title('Capacity');
196 xlabel('Temperatura (°C)'); ylabel('(Ah)');
197 xlim(theXlim);
198 figure(4); plot(model.temps,1000*model.R0Param);
199 title('Resistance');
200 xlabel('Temperatura (°C)'); ylabel('(m\Omega)');
201 xlim(theXlim);
202 figure(5); plot(model.temps,1000*model.M0Param);
203 title('Hyst Magnitude M0');

```

```

202 xlabel('Temperatura (°C)'); ylabel(' (mV)');
203 xlim(theXlim);
204 figure(6); plot(model.temps,1000*model.MParam);
205 title('Hyst Magnitude M');
206 xlabel('Temperatura (°C)'); ylabel(' (mV)');
207 xlim(theXlim);
208 figure(7); plot(model.temps,getParamESC('RCPParam',...
209 model.temps,model));
210 title('RC Time Constant');
211 xlabel('Temperatura (°C)'); ylabel(' (tau)');
212 xlim(theXlim);
213 figure(8); plot(model.temps,1000*getParamESC('RParam',...
214 model.temps,model));
215 title('R in RC');
216 xlabel('Temperatura (°C)'); ylabel(' (m\Omega)');
217 xlim(theXlim);
218 figure(9); plot(model.temps,abs(model.GParam));
219 title('Gamma');
220 xlabel('Temperatura (°C)'); ylabel('');
221 xlim(theXlim);
222 end
223 return
224
225 % -----
226 % Usando um valor assumido para gama (já armazenado no modelo), encontre
227 % os valores ótimos para os da célula restantes e calcule o erro RMS
228 % entre
229 % a tensão da célula real e a prevista
230 % Using an assumed value for gamma (already stored in the model), find
231 % optimum values for remaining cell parameters, and compute the RMS
232 % error between true and predicted cell voltage
233 % -----
234 function [cost,model]=minfn(data,model,theTemp,doHyst)
235 alltemps = [data(:).temp];
236 ind = find(alltemps == theTemp); numfiles = length(ind);
237
238 xplots = ceil(sqrt(numfiles));
239 yplots = ceil(numfiles/xplots);
240 rmserr = zeros(1,xplots*yplots);

```

```

240
241 G = abs(getParamESC('GParam',theTemp,model));
242 Q = abs(getParamESC('QParam',theTemp,model));
243 eta = abs(getParamESC('etaParam',theTemp,model));
244 RC = getParamESC('RCParam',theTemp,model);
245 numpoles = length(RC);

246
247 for thefile = 1:numfiles;
248 ik = data(ind(thefile)).script1.current(:);
249 vk = data(ind(thefile)).script1.voltage(:);
250 tk = (1:length(vk))-1;
251 etaik = ik; etaik(ik<0) = etaik(ik<0)*eta;

252 h=0*ik; sik = 0*ik;
253 fac=exp(-abs(G*etaik/(3600*Q)));
254 for k=2:length(ik),
255 h(k)=fac(k-1)*h(k-1)-(1-fac(k-1))*sign(ik(k-1));
256 sik(k) = sign(ik(k));
257 if abs(ik(k))<Q/100, sik(k) = sik(k-1); end
258 end

260
261 % Primeira etapa da modelagem: calcule o erro com o modelo
262 vest1 = data(ind(thefile)).OCV;
263 verr = vk - vest1;

264
265 % Segunda etapa da modelagem: Calcule a constante de tempo na matriz A
266 np = numpoles;
267 while 1,
268 A = SISOsubid(-diff(verr),diff(etaik),np);
269 eigA = eig(A);
270 eigA = eigA(eigA == conj(eigA)); % conferindo se real
271 eigA = eigA(eigA > 0 & eigA < 1); % conferindo se no intervalo
272 okpoles = length(eigA); np = np+1;
273 if okpoles >= numpoles, break; end
274 fprintf('Trying np = %d\n',np);
275 end
276 RCfact = sort(eigA); RCfact = RCfact(end-numpoles+1:end);
277 RC = -1./log(RCfact);
278 % Simulando os filtros R-C para encontrar as correntes R-C

```

```

279 if exist('dlsim.m','file') % na toolbox "control-system"
280 vrcRaw = dlsim(diag(RCfact),1-RCfact, ...
281 eye(numpoles),zeros(numpoles,1),etaik);
282 else % uma solução um pouco mais lenta se não houver a toolbox "control-
283 system"
284 vrcRaw = zeros(length(RCfact),length(etaik));
285 for vrcK = 1:length(etaik)-1,
286 vrcRaw(:,vrcK+1) = diag(RCfact)*vrcRaw(:,vrcK)+(1-RCfact)*etaik(vrcK);
287 end
288 vrcRaw = vrcRaw';
289 end
290
291 % Terceira etapa da modelagem: Parâmetros da histerese
292 if doHyst,
293 H = [h,sik,-etaik,-vrcRaw];
294 if exist('lsqnonneg.m','file'), % na toolbox "optimization"
295 W = lsqnonneg(H,verr); % W = H\verr;
296 else
297 W = nnls(H,verr); % W = H\verr;
298 end
299 M = W(1); M0 = W(2); R0 = W(3); Rfact = W(4:end)';
300 else
301 H = [-etaik,-vrcRaw];
302 W = H\verr;
303 M=0; M0=0; R0 = W(1); Rfact = W(2:end)';
304 end
305 ind = find(model.temps == data(ind(thefile)).temp,1);
306 model.R0Param(ind) = R0;
307 model.M0Param(ind) = M0;
308 model.MParam(ind) = M;
309 model.RCParam(ind,:) = RC';
310 model.RParam(ind,:) = Rfact';
311
312 vest2 = vest1 + M*h + M0*sik - R0*etaik - vrcRaw*Rfact';
313 verr = vk - vest2;
314
315 % Plotando as tensões
316 figure(1); subplot(yplots,xplots,thefile);
317 plot(tk(1:10:end)/60,vk(1:10:end),tk(1:10:end)/60, ...

```

```

317 vest1(1:10:end),tk(1:10:end)/60,vest2(1:10:end));
318 xlabel('Time (min)'); ylabel('Voltage (V)');
319 title(sprintf('Voltage and estimates at T=%d',...
320 data(ind(thefile)).temp));
321 legend('voltage','vest1 (OCV)','vest2 (DYN)', 'location','southwest');

322 % Plotando os erros de modelagem
323 figure(2); subplot(yplots,xplots,thefile);
324 thetitle=sprintf('Modeling error at T = %d',data(ind(thefile)).temp);
325 plot(tk(1:10:end)/60,verr(1:10:end)); title(thetitle);
326 xlabel('Time (min)'); ylabel('Error (V)');
327 ylim([-0.1 0.1]);
328 drawnow
329

330
331 % Calculando o erro RMS apenas em dados com SOC
332 % aproximadamente de 5% a 95%
333 v1 = OCVfromSOCTemp(0.95,data(ind(thefile)).temp,model);
334 v2 = OCVfromSOCTemp(0.05,data(ind(thefile)).temp,model);
335 N1 = find(vk<v1,1,'first'); N2 = find(vk<v2,1,'first');
336 if isempty(N1), N1=1; end; if isempty(N2), N2=length(verr); end
337 rmserr(thefile)=sqrt(mean(verr(N1:N2).^2));
338 end

339
340 cost=sum(rmserr);
341 fprintf('RMS error = %0.2f (mV)\n',cost*1000);
342 if isnan(cost), stop, end
343 return

344
345 %%%%%%%%
346 % A = SISOsubid(y,u,n);
347 % Identifica a matriz estado de espaço A dos dados de entrada e saída.
348 %     y: vetor de saídas medidas
349 %     u: vetor de entradas medidas
350 %     n: número de pólos na solução
351 %
352 %     A: matriz de transição de estado de espaço em tempo discreto
353 %
354 % A teoria desta aplicação foi retirada do:
355 %                 "Subspace Identification for Linear Systems"

```

```

356 % Theory - Implementation - Applications"
357 % Peter Van Overschee / Bart De Moor (VODM)
358 % Kluwer Academic Publishers, 1996
359 % Combined algorithm: Figure 4.8 page 131 (robust)
360 % Robust implementation: Figure 6.1 page 169"
361 %
362 % E o código adaptado do "subid.m" do:
363 % "Subspace Identification for
364 % Linear Systems" toolbox on MATLAB CENTRAL file
365 % exchange, originally by Peter Van Overschee, Dec. 1995"
366 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
367
368 function A = SISOsubid(y,u,n)
369 y = y(:); y = y'; ny = length(y); % transformando y em um vetor linha
370 u = u(:); u = u'; nu = length(u); % transformando u em um vetor linha
371 i = 2*n; % #linhas na matriz de Hankel. Usualmente: i=2*(ordem máxima)
372 twoi = 4*n;
373
374 if ny ~= nu, error('y and u must be same size'); end
375 if ((ny-twoi+1) < twoi); error('Not enough data points'); end
376
377 % Determine the number of columns in the Hankel matrices
378 % Determinando o número de colunas da matriz de Hankel
379 j = ny-twoi+1;
380
381 % Fazendo as matrizes de Hankel Y e U
382 Y=zeros(twoi,j); U=zeros(twoi,j);
383 for k=1:2*i
384 Y(k,:)=y(k:k+j-1); U(k,:)=u(k:k+j-1);
385 end
386 % Calculando o fator R
387 R = triu(qr([U;Y]'))'; % fator R
388 R = R(1:4*i,1:4*i);
389
390 % -----
391 % STEP 1: Calcular projeções oblíquas e ortogonais
392 % -----
393 Rf = R(3*i+1:4*i,:); % saídas futuras
394 Rp = [R(1:1*i,:);R(2*i+1:3*i,:)]; % entradas e saídas anteriores

```

```

395 Ru = R(1*i+1:2*i,1:twoi); % saídas futuras
396 % Saídas futuras perpendiculares
397 Rfp = [Rf(:,1:twoi) - (Rf(:,1:twoi)/Ru)*Ru,Rf(:,twoi+1:4*i)];
398 % Entradas e saídas perpendiculares anteriores
399 Rpp = [Rp(:,1:twoi) - (Rp(:,1:twoi)/Ru)*Ru,Rp(:,twoi+1:4*i)];
400
401 %%%%%%%%%%%%%%
402 % A projeção oblíqua é calculada como (6.1) em VODM, página 166.
403 % obl/Ufp = Yf/Ufp * pinv(Wp/Ufp) * (Wp/Ufp)
404 % A projeção extra em Ufp (Uf perpendicular) tende a dar um
405 % melhor condicionamento numérico (ver algo em VODM página 131)
406 % Esta verificação é necessária para evitar avisos de deficiência
407 % de classificação
408 %%%%%%%%%%%%%%
409
410 if (norm(Rpp(:,3*i-2:3*i),'fro')) < 1e-10
411 Ob = (Rfp*pinv(Rpp')'*Rp); % Projeção oblíqua
412 else
413 Ob = (Rfp/Rpp)*Rp;
414 end
415
416 % -----
417 % STEP 2: Calculando a projeção oblíqua ponderada e sua projeção
418 % SVD extra de Ob na Uf perpendicular
419 %
420 WOW = [Ob(:,1:twoi) - (Ob(:,1:twoi)/Ru)*Ru,Ob(:,twoi+1:4*i)];
421 [U,S,~] = svd(WOW);
422 ss = diag(S);
423
424 %
425 % STEP 3: Particionando U em U1 e U2 (o último não é usado)
426 %
427 U1 = U(:,1:n); % Determine U1
428
429 %
430 % STEP 4: Determine gam = Gamma(i) e gamm = Gamma(i-1)
431 %
432 gam = U1*diag(sqrt(ss(1:n)));
433 gamm = gam(1:(i-1),:);

```

```

434 gam_inv = pinv(gam); % Pseudo inverso de gam
435 gamm_inv = pinv(gamm); % Pseudo inverso de gamm
436
437 % -----
438 % STEP 5: Determine a matriz A (também C, que não é usada)
439 % -----
440 Rhs = [gam_inv*R(3*i+1:4*i,1:3*i), zeros(n,1); R(i+1:twoi,1:3*i+1)];
441 Lhs = [gamm_inv*R(3*i+1+1:4*i,1:3*i+1); R(3*i+1:3*i+1,1:3*i+1)];
442 sol = Lhs/Rhs; % Resolve o mínimo quadrado para [A;C]
443 A = sol(1:n,1:n); % Extrai A
444 return
445
446 function X = gss(f,a,b,tol)
447 % pesquisa para encontrar o mínimo de f em [a, b]
448 % baseada no código: https://en.wikipedia.org/wiki/Golden-section\_search
449 gr = (sqrt(5)+1)/2; % "golden ratio" usado na pesquisa
450
451 c = b - (b - a) / gr;
452 d = a + (b - a) / gr;
453 while abs(c - d) > tol
454 if f(c) < f(d),
455 b = d;
456 else
457 a = c;
458 end
459
460 % recalculamos c e d aqui para evitar perda de precisão o que
461 % pode levar a resultados incorretos ou loop infinito
462 c = b - (b - a) / gr;
463 d = a + (b - a) / gr;
464 end
465 X = (b+a)/2;
466 return
467
468 function [x,w,info]=nnls(C,d,opts)
469 %%%%%%%%%%%%%%
470 % Direitos autorais desta parte do código
471 % "nnls" Mínimos quadrados não negativos Cx=d x>=0 w=C' (d-Cx)<=0
472 % 2012-08-21 Matlab8 W.Whiten

```

```

473 % 2013-02-17 Line 52 added
474 % Copyright (C) 2012, W.Whiten (personal W.Whiten@uq.edu.au) BSD license
475 % (http://opensource.org/licenses/BSD-3-Clause)
476 %
477 % [x,w,info]=nnls(C,d,opts)
478 % C Coeficiente da matriz
479 % d Vetor Rhs
480 % "opts" Opções da estrutura: (opcional)
481 % .Accy 0 versão rápida, 1 refina o valor final (padrão),
482 % 2 usa passos precisos, mas é muito lenta em casos
483 % grandes, rápida em casos pequenos, resultado geralmente
484 % identico ao 1
485 % .Order Verdadeiro ou [], ou ordem para incluir inicialmente
486 % termos positivos se incluído fornecerá informações.
487 % Se x0 disponível use find(x0>0), mas é melhor salvar
488 % da execução anterior de nnls
489 % .Tol Valor do teste de tolerância, padrão é zero
490 % .Iter Número máximo de interações, não deve ser necessário.
491 %
492 % x Vetor de solução positiva x>=0
493 % w Vetor multiplicador de Lagrange w(x==0)<= aproximadamente zero
494 % Informação extra da estrutura:
495 % .iter Número de interações usadas
496 % .wsc0 Tamanho estimado de erros em w
497 % .wsc Máximo de valores de teste para w
498 % .Order Variáveis de pedido usadas, use para reiniciar "nnls"
499 % com "opts.Order"
500 %
501 % Existe com x>=0 e w<= zero ou ligeiramente acima de 0 devido ao
502 % arredondamento e para garantir a convergência
503 % Usando operações matriciais mais rápidas, refina a resposta
504 % como padrão (Accy 1).
505 % Accy 0 é mais robusto em casos singulares.
506 %
507 % Follows Lawson & Hanson, Solving Least Squares Problems, Ch 23.
508 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
509 %
510 [~,n]=size(C);
511 maxiter=4*n;

```

```

512
513 % valores iniciais
514 P=false(n,1);
515 x=zeros(n,1);
516 z=x;
517
518 w=C'*d;
519
520 % wsc_ são escalas para erros
521 wsc0=sqrt(sum(w.^2));
522 wsc=zeros(n,1);
523 tol=3*eps;
524 accy=1;
525 pn1=0;
526 pn2=0;
527 pn=zeros(1,n);
528
529 % veja se os valores das opções foram dados
530 ind=true;
531 if nargin>2
532 if(isfield(opts,'Tol'))
533 tol=opts.Tol;
534 wsc(:)=wsc0*tol;
535 end
536 if(isfield(opts,'Accy'))
537 accy=opts.Accy;
538 end
539 if(isfield(opts,'Iter'))
540 maxiter=opts.Iter;
541 end
542 end
543
544 % testa se usa a matriz normal para velocidade
545 if(accy<2)
546 A=C'*C;
547 b=C'*d;
548 %L=zeros(n,n);
549 LL=zeros(0,0);
550 lowtri=struct('LT',true);

```

```

551 uptri=struct('UT',true);
552 end
553
554 % testa se informações iniciais fornecidas
555 if(nargin>2)
556 if(isfield(opts,'Order') && ~islogical(opts.Order))
557 pn1=length(opts.Order);
558 pn(1:pn1)=opts.Order;
559 P(pn(1:pn1))=true;
560 ind=false;
561 end
562 if(~ind && accy<2)
563 %L(1:pn1,1:pn1)=chol(A(pn(1:pn1),pn(1:pn1)), 'lower');
564 UU(1:pn1,1:pn1)=chol(A(pn(1:pn1),pn(1:pn1)));
565 LL=UU';
566 end
567 pn2=pn1;
568 end
569
570 % loop até que todas as variáveis possitivas sejam adicionadas
571 iter=0;
572 while(true)
573 % Verifica se não tem mais termos a serem adicionados
574 if(ind && (all(P==true) || all(w(~P)<=wsc(~P))))
575 if(accy~=1)
576 break
577 end
578 accy=2;
579 ind=false;
580 end
581
582 % pula se primeira vez e ordem inicial foram dadas
583 if(ind)
584 % seleciona o melhor termo para adicionar
585 ind1=find(~P);
586 [~,ind2]=max(w(ind1)-wsc(ind1));
587 ind1=ind1(ind2);
588 P(ind1)=true;
589 pn2=pn1+1;

```

```

590 pn(pn2)=ind1;
591 end
592
593 % loop até que todos os termos negativos sejam removidos
594 while(true)
595
596 % verifica a divergência
597 iter=iter+1;
598 if(iter>=2*n)
599 if(iter>maxiter)
600 error(['nnls Failed to converge in ' num2str(iter) ...
601 ' iterations'])
602 elseif(mod(iter,n)==0)
603 wsc=(wsc+wsc0*tol)*2;
604 end
605 end
606
607 % resolver usando "suspected positive terms"
608 z(:)=0;
609 if(accy>=2)
610 z(P)=C(:,P)\d;
611 else
612 % adicionar uma linha ao fator triangular inferior
613 for i=pn1+1:pn2
614 i1=i-1;
615 %LL=L(1:i1,1:i1);
616 %LL=LL(1:i1,1:i1);
617 t=linsolve(LL,A(pn(1:i1),pn(i)),lowtri);
618 %t=LL\A(pn(1:i1),pn(i));
619 %L(i,1:i1)=t;
620 %LL(i,1:i1)=t;
621 AA=A(pn(i),pn(i));
622 tt=AA-t'*t;
623 if(tt<=AA*tol)
624 tt=1e300;
625 else
626 tt=sqrt(tt);
627 end
628 %L(i,i)=sqrt(tt);

```

```

629 %LL(i,i)=sqrt(tt);
630 LL(i,1:i)=[t',tt];
631 UU(1:i,i)=[t;tt];
632 end
633
634 % resolver usando o fator triangular inferior
635 %LL=L(1:pn2,1:pn2);
636 t=linsolve(LL,b(pn(1:pn2)),lowtri);
637 %t=LL\b(pn(1:pn2));
638 %UU=LL';
639 %z(pn(1:pn2))=linsolve(UU,t,uptri);
640 z(pn(1:pn2))=linsolve(UU,t,uptri);
641 %z(pn(1:pn2))=LL'\t;
642 % ou podemos usar isso para resolver sem atualizar os fatores
643 %z(pn(1:pn2))=A(pn(1:pn2),pn(1:pn2))\b(pn(1:pn2));
644 end
645 pn1=pn2;
646
647 % checando se os termos são positivos
648 if(all(z(P)>=0))
649 x=z;
650 if(accy<2)
651 w=b-A*x;
652 else
653 w=C'* (d-C*x);
654 end
655 wsc(P)=max(wsc(P),2*abs(w(P)));
656 ind=true;
657 break
658 end
659
660 % seleciona e remove o pior termo negativo
661 ind1=find(z<0);
662 [alpha,ind2]=min(x(ind1)./(x(ind1)-z(ind1)+realmin));
663 ind1=ind1(ind2);
664
665 % testa se removendo o último adicionado, aumenta "wsc" para evitar
666 % loop
667 if(x(ind1)==0 && ind)

```

```

668 w=C' * (d-C*z);
669 wsc(ind1)=(abs(w(ind1))+wsc(ind1))*2;
670 end
671 P(ind1)=false;
672 x=x-alpha*(x-z);
673 pn1=find(pn==ind1);
674 pn(pn1:end)=[pn(pn1+1:end),0];
675 pn1=pn1-1;
676 pn2=pn2-1;
677 if(accy<2)
678 LL=LL(1:pn1,1:pn1);
679 UU=UU(1:pn1,1:pn1);
680 end
681 ind=true;
682 end
683 end
684
685 % resultado de informação necessário
686 if(nargout>2)
687 info.iter=iter;
688 info.wsc0=wsc0*eps;
689 info.wsc=max(wsc);
690 if(nargin>2 && isfield(opts,'Order'))
691 info.Order=pn(1:pn1);
692 end
693 end
694
695 return

```

### A.2.3 setupDynData.m

```

1 % Algoritmo DYN
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons

```

```

9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 %-----%
15 % -----
16 % Opções de arquivos usados para otimizações em diferentes temperaturas
17 % Temp: -25 -15 -05 05 15 25 35 45
18 % -----
19 % A123: 10 10 30 45 50 50 50 50 (célula utilizada aqui)
20 % ATL: 2 5 15 20 30 40 50 50
21 % E1: 2 5 10 20 25 35 45 50
22 % E2: 2 5 5 15 25 35 45 50
23 % P14: 4 10 20 35 50 50 50 50
24 % SAM: 2 2 5 10 10 15 15 15
25 %
26 % Note que o indicador numérico indica a taxa-C máxima usada em cada
27 % teste
28 % (ex: "40" - 4.0C). Por causa da maior resistência em temperaturas mais
29 % baixas, nós devemos reduzir a corrente máxima para evitar exceder os
30 % limites de tensão de cada célula.
31 clear all
32
33 cellIDs = {'A123'};
34 temps = [-25 -15 -5 5 15 25 35 45];
35 mags = {[10 10 30 45 45 50 50 50]};
```

### A.3 Algoritmos do Filtro de Kalman Ponto Sigma

#### A.3.1 runSPKF.m

```

1 % Algoritmo SPKF
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %-----%
```

```

7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume II, Equivalent-Circuit Methods," Artech
14 % House,
15 % 2015.
16 % -----
17 % runSPKF: Executa um filtro de Kalman ponto-sigma para os dados dinâ-
18 % micos
19 %
20 % E2 salvos e um modelo de célula E2
21 %
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23 % Inicialmente carregamos o modelo ESC no algoritimo, definimos a
24 % temperatura de de trabalho como 25°C e armazenamos os valores de tempo,
25 % corrente, tensão e
26 % SOC obtidos do teste DYN em vetores. Também é definido os valores
27 % iniciais correspondentes ao Filtro como Sigma X0, Sigma V e Sigma W,
28 % correspondentes a incerteza do estado inicial, sensor de tensão na equa-
29 % ção de saída e sensor de corrente na equação de estado do Filtro.
30 %
31 load A123model.mat
32 %
33 % Carrega os dados de teste da célula para serem usados para este
34 % experimento em lote.
35 % Contém a variável "DYNDATA" na qual o campo "script1" é o de interesse.
36 % Este contém sub campos: time, current, voltage, soc
37 %
38 % load('A123_DYN_45_P05'); T = 05; % 05°C
39 % load('A123_DYN_45_P15'); T = 15; % 15°C
40 load('A123_DYN_50_P25'); T = 25; % 25°C
41 % load('A123_DYN_50_P30'); T = 25; %25°C
42 % load('A123_DYN_50_P35'); T = 35; % 35°C
43 % load('A123_DYN_50_P45'); T = 45; % 45°C
44 %
45 time      = DYNDATA.script1.time(:);    deltat = time(2)-time(1);

```

```

40 time      = time-time(1); % tempo inicial em 0
41 current   = DYNDData.script1.current(:); % descarga > 0; carga < 0.
42 voltage   = DYNDData.script1.voltage(:);
43 soc       = DYNDData.script1.soc(:);
44
45 % Reserve armazenamento para resultados computados, para plotagem
46 sochat = zeros(size(soc));
47 socbound = zeros(size(soc));
48
49 % Valores de covariância
50 SigmaX0 = diag([1e-6 1e-8 2e-4]); % incerteza do estado inicial
51 SigmaV = 2e-1; % Incerteza do sensor de tensão, equação de saída
52 SigmaW = 2e-1; % Incerteza do sensor de corrente, equação de estado
53 %%%%%%%%%%%%%%
54 %%%%%%%%%%%%%%
55 % A primeira função a ser chamada é a initSPKF que será utilizada apenas
56 % uma vez no início do algorítimo para criar as estruturas de dados que
57 % serão utilizadas
58 % para a implementação do SPKF.
59
60 % Cria a estrutura "spkfData" e inicializa as variáveis usando a primeira
61 % medição de tensão e primeira medição de temperatura
62 spkfData = initSPKF(voltage(1), T, SigmaX0, SigmaV, SigmaW, model);
63 %%%%%%%%%%%%%%
64 %%%%%%%%%%%%%%
65 % É dado início ao loop for que irá transcorrer sobre todos os dados do
66 % teste UDDS e computar os valores de saída do SPKF. Para calcular os
67 % valores de
68 % estado e saída é utilizada a função initSPKF que tem como entrada os
69 % valores medido de tensão vk, corrente ik, temperatura definida Tk,
70 % intervalo de tempo de amostra Delta t e a estrutura de dados definida
71 % anteriormente spkfData. A função retorna os valores zk que é o valor de
72 % SOC
73 % estimado para ponto, zkbnd que são as bordas de erro definidas pela
74 % simulação e o spkfData que é a estrutura dos dados que foram utilizados.
75
76 % Agora, entre no loop para o restante do tempo, onde vamos atualizar o

```

```

71 % SPKF
72 hwait = waitbar(0,'Computing...');
73 for k = 1:length(voltage),
74 vk = voltage(k); % tensão "medida"
75 ik = current(k); % corrente "medida"
76 Tk = T;           % temperatura "medida"
77
78 % Atualize o SOC (e outros estados)
79 [sochat(k),socbound(k),spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData);
80 % atualize periodicamente a barra de espera, mas não tão frequente
81 % (procedimento lento)
82 if mod(k,1000)==0,
83 waitbar(k/length(current),hwait);
84 end;
85 end
86 close(hwait);
87 %%%%%%%%%%%%%%
88
89 %%%%%%%%%%%%%%
90 % Agora plotamos os gráficos dos resultados da estivatima do SOC pelo
91 % SPKF
92
93 % Plotar estimativa do SOC
94 figure(1); clf; plot(time/60,100*soc,'k',time/60,100*sochat,'b'); hold on
95 plot([time/60; NaN; time/60],[100*(sochat+socbound); NaN; 100*(sochat-
96 -socbound)],'r');
97 %title('SOC estimation using SPKF'); xlabel('Time (min)'); ylabel('SOC
98 (%)');
99 title('Estimando o SOC com o SPKF'); xlabel('Tempo (min)'); ylabel('SOC
(%)');
100 %legend('Truth','Estimate','Bounds');
101 legend('Real','Estimado','Margem de 5%');
102 grid on; ylim([0 120])
103
104 % Exibir erro de estimativa RMS na janela de comando
105 fprintf('RMS SOC estimation error = %g%%\n',sqrt(mean((100*(soc-sochat))^.^2)));
106
107 % Plotar erro de estimativa e limites

```

```

105 figure(2); clf; plot(time/60,100*(soc-sochat),'b'); hold on
106 plot([time/60; NaN; time/60],[100*socbound; NaN; -100*socbound],'r');
107 %title('SOC estimation errors using SPKF');
108 title('Erro da estimativa de SOC utilizando o SPKF');
109 %xlabel('Time (min)'); ylabel('SOC error (%)'); ylim([-6 6]);
110 xlabel('Tempo (min)'); ylabel('Erro (%)'); ylim([-6 6]);
111 set(gca,'ytick',-6:2:6);
112 %legend('SPKF error','location','northwest');
113 legend('Erro','location','northwest');
114 grid on
115
116 % Mostrar limites de erro na janela de comando
117 ind = find(abs(soc-sochat)>socbound);
118 fprintf('Percent of time error outside bounds = %g%%\n',...
119 length(ind)/length(soc)*100);

```

### A.3.2 initSPKF.m

```

1 % Algoritmo SPKF
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 %
16 % função spkfData = initSPKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)
17 %
18 %     Inicializa uma estrutura "spkfData", usada pelo filtro de Kalman
19 %     ponto
20 %     sigma para armazenar seu próprio estado e dados associados
21 %

```

```

21 % Entradas:
22 % v0: Tensão inicial da célula
23 % T0: Temperatura inicial da célula
24 % SigmaX0: Matriz de covariância de incerteza de estado inicial
25 % SigmaV: Covariância do ruído de medição
26 % SigmaW: Covariância do ruído de processo
27 % model: modelo ESC da célula
28 %
29 % Saída:
30 % spkfData: Estrutura de dados usados pelo código SPKF
31
32 %%%%%%
33 % O primeiro bloco desta função define a corrente inicial do resistor de
    difusão, o estado inicial da histerese, de carga e armazena os valores
    na estimativa
34 % de estado xhat
35
36 function spkfData = initSPKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)
37 % Descrição do estado inicial
38 ir0 = 0;                      spkfData.irInd = 1;
39 hk0 = 0;                      spkfData.hkInd = 2;
40 SOC0 = SOCfromOCVtemp(v0,T0,model); spkfData.zkInd = 3;
41 spkfData.xhat = [ir0 hk0 SOC0]'; % estado inicial
42 %%%%%%
43
44 %%%%%%
45 % Armazena os valores de covariância do erro de estimativa do estado
    inicial do ruído do processo e do sensor e calcula o fator de Cholesky
    que é a raiz
46 % quadrada do ruído do processo e os componentes de ruído do sensor da
    matriz de covariância de estado aumentado
47
48 % Valores de covariância
49 spkfData.SigmaX = SigmaX0;
50 spkfData.SigmaV = SigmaV;
51 spkfData.SigmaW = SigmaW;
52 spkfData.Snoise = real(chol(diag([SigmaW; SigmaV]),'lower'));
53 spkfData.Qbump = 5;
54 %%%%%%

```

```

55
56 %%%%%%
57 % O segundo bloco define o número de elementos no vetor de estado Nx, de
58 % medição Ny, de entrada do sistema Nu, do ruído do processo Nw, do
59 % ruído do sensor Nv e calcula o número de elementos no vetor de estado
60 % aumentado Na.
61
62 % parâmetros específicos do SPKF
63 Nx = length(spkfData.xhat); spkfData.Nx = Nx; % comprimento do vetor de
64 % estado
65 Ny = 1; spkfData.Ny = Ny; % comprimento do vetor de medição
66 Nu = 1; spkfData.Nu = Nu; % comprimento do vetor de entrada
67 Nw = size(SigmaW,1); spkfData.Nw = Nw; % comprimento do vetor de ruído de
68 % processo
69 Nv = size(SigmaV,1); spkfData.Nv = Nv; % comprimento do vetor de ruído do
70 % sensor
71 Na = Nx+Nw+Nv; spkfData.Na = Na; % comprimento do vetor de estado
72 % aumentado
73
74 %%%%%%
75 % O terceiro bloco define os fatores de ajuste do SPKF
76
77 % valor prévio da corrente
78 spkfData.priorI = 0;
79 spkfData.signIk = 0;
80
81
82 % armazene a estrutura de dados do modelo também
83 spkfData.model = model;
84
85 % Todos estes dados gerados pela função initSPKF são armazenados para
86 % acesso durante a rotina de iteração da função iterSPKF

```

### A.3.3 iterSPKF.m

```

1 % Algoritmo SPKF
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo aplicado neste projeto está protegido por direitos autorais
5 % de Gregory L. Plett:
6 %%%%%%%%%%%%%%
7 % Copyright (c) 2015 by Gregory L. Plett of the University of Colorado
8 % Colorado Springs (UCCS). This work is licensed under a Creative Commons
9 % Attribution-NonCommercial-ShareAlike 4.0 Intl. License, v. 1.0.
10 % It is provided "as is", without express or implied warranty, for
11 % educational and informational purposes only.
12 % This file is provided as a supplement to: Plett, Gregory L., "Battery
13 % Management Systems, Volume I, Battery Modeling," Artech House, 2015.
14 %%%%%%%%%%%%%%
15 %
16 % função [zk,zkbnd,spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData)
17 %
18 % Executa uma interação do filtro de Kalman ponto-sigma usando a nova
19 % medição de dados
20 %
21 % Entradas:
22 % vk: tensão da célula medida (ruidosa) presente
23 % ik: corrente da célula medida (ruidosa) presente
24 % Tk: temperatura presente
25 % deltat: intervalo de amostragem
26 % spkfData: estrutura de dados inicializa pelo "initSPKF" e
27 %           atualizada pelo "iterSPKF".
28 %
29 % Saídas:
30 % zk: SOC estimado para esta amostra de tempo
31 % zkbnd: limites de estimativa "3-sigma"
32 % spkfData: estrutura de dados usados para armazenar variáveis
33
34 function [zk,zkbnd,spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData)
35 %%%%%%%%%%%%%%
36 % O primeiro bloco desta função extrai os dados do modelo ESC que serão
            utilizados para simular a célula e modificar a corrente de entrada

```

```

    aplicando o fator de
37 % Eficiência de Coulomb (eta).

38
39 model = spkfData.model;
40 % Carrega os parâmetros do modelo da célula
41 Q = getParamESC('QParam',Tk,model);
42 G = getParamESC('GParam',Tk,model);
43 M = getParamESC('MParam',Tk,model);
44 M0 = getParamESC('M0Param',Tk,model);
45 RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
46 R = getParamESC('RParam',Tk,model)';
47 R0 = getParamESC('R0Param',Tk,model);
48 eta = getParamESC('etaParam',Tk,model);
49 if ik<0, ik=ik*eta; end;
50 %%%%%%%%%%%%%%
51
52 %%%%%%%%%%%%%%
53 % Em seguida, carrega as constantes da estrutura de dados spkfData para
      utilização em variáveis mais acessíveis.

54
55 % Obtém os dados armazenados na estrutura "spkfData"
56 I = spkfData.priorI;
57 SigmaX = spkfData.SigmaX;
58 xhat = spkfData.xhat;
59 Nx = spkfData.Nx;
60 Nw = spkfData.Nw;
61 Nv = spkfData.Nv;
62 Na = spkfData.Na;
63 Snoise = spkfData.Snoise;
64 Wc = spkfData.Wc;
65 irInd = spkfData.irInd;
66 hkInd = spkfData.hkInd;
67 zkInd = spkfData.zkInd;
68 if abs(ik)>Q/100, spkfData.signIk = sign(ik); end;
69 signIk = spkfData.signIk;
70 %%%%%%%%%%%%%%
71
72 %%%%%%%%%%%%%%
73 % Agora o algorítimo começa a calcular os Steps apresentados na Seção de

```

```

Metodologia que definem o SPKF. Primeiro calculamos a estimativa de
estado aumentado

74 % no tempo anterior e a matriz de incerteza da raiz quadrada
    correspondente.

75

76 % Step 1a: Atualiza o tempo de estimativa de estado
77 %           - Cria os pontos SigmaX aumentados "xhatminus"
78 %           - Extrai pontos SigmaX do estado "xhatminus"
79 %           - Calcula média ponderada "xhatminus (k)"

80

81 % Step 1a-1: Cria SigmaX e xhat aumentados
82 [sigmaXA,p] = chol(SigmaX,'lower');
83 if p>0,
84 fprintf('Cholesky error. Recovering...\n');
85 theAbsDiag = abs(diag(SigmaX));
86 sigmaXA = diag(max(SQRT(theAbsDiag),SQRT(spkfData.SigmaW)));
87 end
88 sigmaXA=[real(sigmaXA) zeros([Nx Nw+Nv]); zeros([Nw+Nv Nx]) Snoise];
89 xhata = [xhat; zeros([Nw+Nv 1])];
90 % NOTA: sigmaXA é triangular inferior
91 %%%%%%%%%%%%%%
92
93 % O algorítimo segue calculando a matriz Xa que contém os pontos Sigma
    aumentado, implementa a equação de estado em Xx, a estimativa da previsã
    o
94
95 % de estado no momento em xhat e a matriz de covariância de erro SigmaX.

96

97 % Step 1a-2: Calcula pontos SigmaX (indexação estranha de xhata para
98 % evitar a chamada "repmat", que é muito ineficiente no MATLAB)
99 Xa = xhata(:,ones([1 2*Na+1])) + ...
100 spkfData.h*[zeros([Na 1]), sigmaXA, -sigmaXA];

101

102 % Step 1a-3: Atualiza o tempo da última interação até agora
103 %           stateEqn(xold,current,xnoise)
104 Xx = stateEqn(Xa(1:Nx,:), I, Xa(Nx+1:Nx+Nw,:));
105 xhat = Xx*spkfData.Wm;
106 xhat(hkInd) = min(1,max(-1,xhat(hkInd)));
107 xhat(zkInd) = min(1.05,max(-0.05,xhat(zkInd)));

```

```

108
109 % Step 1b: Atualização do tempo de covariância de erro
110 %           - Calcula a covariância ponderada "sigmaminus(k)"
111 %           (indexação estranha de xhat para evitar a chamada "repmat")
112 Xs = Xx - xhat(:,ones([1 2*Na+1]));
113 SigmaX = Xs*diag(Wc)*Xs';
114 %%%%%%%%%%%%%%
115
116 % O Step 1 é finalizado calculando a previsão da saída como yhat
117
118
119 % Step 1c: Estimativa de saída
120 %           - Calcula a estimativa de saída ponderada "yhat(k)"
121 I = ik; yk = vk;
122 Y = outputEqn(Xx,I+Xa(Nx+1:Nx+Nw,:),Xa(Nx+Nw+1:end,:),Tk,model);
123 yhat = Y*spkfData.Wm;
124 %%%%%%%%%%%%%%
125
126 % Seguimos com o Step 2 que calcula a matriz de ganho L através da covari
127 % ância cruzada SigmaXY entre o erro de previsão de estado e a inovação e
128 % o somatório
129
130 % ponderado para a covariância da inovação SigmaY.
131
132 % Step 2a: Matriz de ganho estimado
133 Ys = Y - yhat(:,ones([1 2*Na+1]));
134 SigmaXY = Xs*diag(Wc)*Ys';
135 SigmaY = Ys*diag(Wc)*Ys';
136 L = SigmaXY/SigmaY;
137
138 % Na sequencia calculamos a inovação ou medição residual r, verificamos
139 % se não existe muita discrepância nos valores para que possa detectar um
140 % erro de
141 % medição do sensor, caso esteja dentro do limite é atualizada a previsão
142 % de estado xhat e definido os limites superior e inferior da estimativa
143 % desejada
144 % e atualizamos a matriz de covariância de erro de medição.
145
146 % Step 2b: Atualização da medição da estimativa do estado

```

```

141 r = yk - yhat; % residual. Use para verificar erros do sensor...
142 if r^2 > 100*SigmaY, L(:,1)=0.0; end
143 xhat = xhat + L*r;
144 xhat(zkInd)=min(1.05,max(-0.05,xhat(zkInd)));
145
146 % Step 2c: Atualização de medição de covariância de erro
147 SigmaX = SigmaX - L*SigmaY*L';
148 [~,S,V] = svd(SigmaX);
149 HH = V*S*V';
150 SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % ajuda a manter robustez
151 %%%%%%%%%%%%%%
152
153 % Código Q-bump
154 if r^2>4*SigmaY % estimativa ruim de tensão por (2-sigmaY), "bump Q"
155 fprintf('Bumping sigmax\n');
156 SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*spkfData.Qbump;
157 end
158
159 %%%%%%%%%%%%%%
160 % Por fim armazenamos as informações necessárias para a próxima iteração
161 % e retornamos o estimador de estado de carga zk e seus limites zkbnd.
162 % Salva os dados na estrutura "spkfData" spara a próxima vez...
163 spkfData.priorI = ik;
164 spkfData.SigmaX = SigmaX;
165 spkfData.xhat = xhat;
166 zk = xhat(zkInd);
167 zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));
168
169 %%%%%%%%%%%%%%
170 % A função stateEqn que retorna os valores a serem armazenados em Xx
171 % calcula os novos valores dos estados de corrente no resistor de difusão
172 % ir, a
173 % matriz de histerese hk e o SOC zk.
174
175 % Calcula novos estados para todos vetores de estados antigos em "xold"
176 function xnew = stateEqn(xold,current,xnoise)
177 current = current + xnoise; % ruído adicionado a corrente
178 xnew = 0*xold;

```

```

177 xnew(irInd,:) = RC*xold(irInd,:)+ (1-RC)*current;
178 Ah = exp(-abs(current*G*deltat/(3600*Q))); % fator de histerese
179 xnew(hkInd,:) = Ah.*xold(hkInd,:)+(Ah-1).*sign(current);
180 xnew(zkInd,:) = xold(zkInd,:)- current*deltat/(3600*Q);
181 end
182 %%%%%%%%%%%%%%
183
184 % A função outputEqn que retorna o valor a ser armazenado em Y tem como
185 % parâmetros de entrada xhat que é uma matriz que contém em cada coluna os
186 % ponto
187 % sigma de estado previsto, corrente medida current, ruídos do sensor
188 % ynoise, temperatura escolhida do modelo T e os parâmetros do modelo ESC
189 % model.
190 % A previsão de tensão yhat é então calculada com base nos pontos sigma
191 % de entrada do modelo.
192
193 % Calcula a tensão de saída da célula para todos os vetores de estado em
194 % "xhat"
195 function yhat = outputEqn(xhat,current,ynoise,T,model)
196 yhat = OCVfromSOCtemp(xhat(zkInd,:),T,model);
197 yhat = yhat + M*xhat(hkInd,:)+M0*signIk;
198 yhat = yhat - R*xhat(irInd,:)-R0*current+ynoise(1,:);
199 end
200 %%%%%%%%%%%%%%
201
202 % Raiz quadrada "segura"
203 function X = SQRT(x)
204 X = sqrt(max(0,x));
205 end
206 end

```

## A.4 Algoritmos dos Indicadores de Performance do Piloto

### A.4.1 KPIDYN.m

```

1 % Algoritmo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.

```

```

4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 % Este algoritmo chama todas as funções definidas para separar os sinhas
7 % de
8 % brake e throttle do sinal de corrente UUDS de teste dinâmico da bateria
9 % em laboratório e posteriormente as funções que retornar os KPIs de
10 % agressividade e liberação do freio e também do acelerador
11
12 % Definição da função
13
14 % Carregar aquivo A123 DYN com a temperatura selecionada
15 % load('A123_DYN_50_P25.mat', 'DYNDdata') % 25°C
16 load('A123_DYN_50_P35.mat', 'DYNDdata') % 35°C
17 % load('A123_DYN_50_P45.mat', 'DYNDdata') % 45°C
18 % load('A123_DYN_45_P15.mat', 'DYNDdata') % 15°C
19 % load('A123_DYN_45_P05.mat', 'DYNDdata') % 05°C
20
21 teste = DYNDdata.script1;
22
23 % Condições iniciais
24 T = 25; % Mude para qualquer temperatura que for do seu interesse
25 k = 0;
26
27 %Variáveis
28 time = teste.time; % tempo
29 current = teste.current; % corrente
30
31 % Inicializa as variáveis com o tamanho real do sinal UDDS
32 interations = length(time); % variável para armazenar o número de
   amostras
33 throttle = zeros(1,interations); % variável para armazenar o sinal
   throttle
34 brake = zeros(1,interations); % variável para armazenar o sinal brake
35
36 % Funções para calcular os KPIs
37 % Separando os sinais de Throttle e Brake
38 [throttle, brake] = signals(current,throttle,brake,interations,k);
39

```

```

40 % Funções para KPI de freio
41 % Passos para gerar os KPIs agressividade de freio e liberação de freio
42 [brake_speed] = bspeed(brake);
43 [brake_aggression] = baggression(brake,brake_speed,interations,k);
44 [brake_release] = brelease(brake,brake_speed,interations,k);
45 [brake_aggression_kpi_mean, brake_aggression_kpi] = bmeanaggression(
    brake_aggression,interations,k);
46 [brake_release_kpi, brake_release_kpi_mean] = bmeanrelease(brake_release,
    interations,k);

47
48 % Funções para KPI de acelerador
49 % Passos para gerar os KPIs agressividade do acelerador e liberação do
50 % acelerador
51 [throttle_speed] = tspeed(throttle);
52 [throttle_aggression] = taggression(throttle,throttle_speed,interations,k
    );
53 [throttle_release] = trelease(throttle,throttle_speed,interations,k);
54 [throttle_aggression_kpi_mean, throttle_aggression_kpi] = tmeanaggression(
    throttle_aggression,interations,k);
55 [throttle_release_kpi, throttle_release_kpi_mean] = tmeanrelease(
    throttle_release,interations,k);

56
57 % Plotando os gráficos de corrente de descarga e recarga da bateria
58 figure(01); % Sinal de corrente UDDS
59 subplot(1,2,1)
60 plot(time, current)
61 %title('Current Signal UDDS Dynamic ')
62 title('Sinal de Corrente UDDS Dinâmico')
63 xlabel('Tempo (s)')
64 %xlabel('Time')
65 ylabel('Sinal de Corrente UDDS (A)')
66 %ylabel('Current UDDS Signal')
67 legend({'Sinal UDDS'},'Location','southwest')

68
69 subplot(1,2,2) % Sinal de freio e acelerador extraídos do sinal UDDS
70 plot(time, throttle)
71 %title('Throttle and Brake')
72 title('Acelerador e Freio')
73 %xlabel('Time')

```

```

74 xlabel('Tempo (s)')
75 ylabel('Current')
76 ylabel('Corrente (A)')
77 hold on
78 plot(time, brake)
79 %legend({'Throttle','Brake'},'Location','southwest')
80 legend({'Acelerador','Freio'},'Location','southwest')
81 hold off
82
83 % Plotando os gráficos de freio
84 figure(02);
85
86 subplot(2,2,1) % Derivada do sinal de freio, "velocidade do freio"
87 plot(time, brake)
88 %title('Brake and Brake Speed')
89 title('Freio e Velocidade do Freio')
90 %xlabel('Time')
91 xlabel('Tempo (s)')
92 ylabel('Current / Current Derivate')
93 ylabel('Corrente (A) / Derivada da corrente')
94 hold on
95 plot(time, brake_speed)
96 %legend({'Brake','Brake Speed'},'Location','southwest')
97 legend({'Freio','Velocidade do Freio'},'Location','southwest')
98 hold off
99
100 subplot(2,2,2) % Agressividade de freio
101 plot(time, brake)
102 %title('Brake and Brake Aggression')
103 title('Freio e Agressividade do Freio')
104 %xlabel('Time')
105 xlabel('Tempo (s)')
106 ylabel('Current / Current Derivate')
107 ylabel('Corrente (A) / Derivada da corrente')
108 hold on
109 plot(time, brake_aggression)
110 %legend({'Brake','Brake Aggression'},'Location','southwest')
111 legend({'Freio','Agressividade do freio'},'Location','southwest')
112 hold off

```

```

113
114 subplot(2,2,3) % Liberação de freio
115 plot(time, brake)
116 %title('Brake and Brake Release')
117 title('Freio e Liberação do Freio')
118 %xlabel('Time')
119 xlabel('Tempo (s)')
120 %ylabel('Current / Current Derivate')
121 ylabel('Corrente (A) / Derivada da corrente')
122 hold on
123 plot(time, brake_release)
124 %legend({'Brake','Brake Release'},'Location','southwest')
125 legend({'Freio','Liberação do Freio'},'Location','southwest')
126 hold off
127
128 subplot(2,2,4) % KPIs de agressividade e liberação de freio
129 plot(time, brake_aggression_kpi_mean)
130 %title('KPI - Brake Aggression (Mean) and Brake Release (Mean)')
131 title('KPI - Agressividade do Freio e Liberação do Freio')
132 %xlabel('Time')
133 xlabel('Tempo (s)')
134 %ylabel('KPI')
135 ylabel('KPI')
136 hold on
137 plot(time, brake_release_kpi_mean)
138 %legend({'Brake Aggression KPI','Brake Release KPI'},'Location','
139     southwest')
140 legend({'KPI - Agressividade do Freio','KPI - Liberação do Freio'},'
141     'Location','southwest')
142 hold off
143
144 % Plotando os gráficos do acelerador
145 figure(03);
146
147 subplot(2,2,1) % Derivada do sinal do acelerador, "velocidade do
148     acelerador"
149 plot(time, throttle)
150 %title('Throttle and Throttle Speed')
151 title('Acelerador e Velocidade do Acelerador')

```

```

149 xlabel('Time')
150 xlabel('Tempo (s)')
151 ylabel('Current / Current Derivate')
152 ylabel('Corrente (A) / Derivada da corrente')
153 hold on
154 plot(time, throttle_speed)
155 %legend({'Throttle','Throttle Speed'},'Location','southwest')
156 legend({'Acelerador','Velocidade do Acelerador'},'Location','southwest')
157 hold off
158
159 subplot(2,2,2) % Agressividade do acelerador
160 plot(time, throttle)
161 %title('Throttle and Throttle Aggression')
162 title('Acelerador e Agressividade do Acelerador')
163 xlabel('Time')
164 xlabel('Tempo (s)')
165 ylabel('Current / Current Derivate')
166 ylabel('Corrente (A) / Derivada da corrente')
167 hold on
168 plot(time, throttle_aggression)
169 %legend({'Throttle','Throttle Aggression'},'Location','southwest')
170 legend({'Acelerador','Agressividade do Acelerador'},'Location','southwest')
171 hold off
172
173 subplot(2,2,3) % Liberação do acelerador
174 plot(time, throttle)
175 %title('Throttle and Throttle Release')
176 title('Acelerador e Liberação do Acelerador')
177 xlabel('Time')
178 xlabel('Tempo (s)')
179 ylabel('Current / Current Derivate')
180 ylabel('Corrente (A) / Derivada da corrente')
181 hold on
182 plot(time, throttle_release)
183 %legend({'Throttle','Throttle Release'},'Location','southwest')
184 legend({'Acelerador','Liberação do Acelerador'},'Location','southwest')
185 hold off
186

```

```

187 subplot(2,2,4) % KPIs de agressividade e liberação do acelerador
188 plot(time, throttle_aggression_kpi_mean)
189 %title('KPI - Throttle Aggression (Mean) and Throttle Release (Mean)')
190 title('KPI - Agressividade do Acelerador e Liberação do Acelerador')
191 %xlabel('Time')
192 xlabel('Tempo (s)')
193 %ylabel('KPI')
194 ylabel('KPI')
195 hold on
196 plot(time, throttle_release_kpi_mean)
197 %legend({'Throttle Aggression KPI','Throttle Release KPI'},'Location',...
198 %        'southwest')
199 legend({'KPI - Agressividade do Acelerador','KPI - Liberação do
Acelerador'},'Location','southwest')
200 hold off

```

#### A.4.2 signals.m

```

1 % Algorítimo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna a o sinal separado de throttle e brake,
8 % correspondentes ao sinal positivo de descarga da bateria e negativo de
9 % recarga da bateria, respectivamente, do sinal original do perfil UDDS
10 % de teste dinâmico da bateria em laboratório
11 %
12 % Entrada:
13 %     current = sinal de corrente UDDS de teste dinâmico da bateria em
14 %               laboratório
15 %     throttle = vetor linha nulo com comprimento igual ao número de
16 %               amostras do sinal UDDS
17 %     brake = vetor linha nulo com o comprimento igual ao número de
18 %               amostras do sinal UDDS
19 %     interations = número de amostra de dados do sinal
20 %     k = número da amostra
21 %
22 % Saída:

```

```

21 %      throttle = sinal de corrente de descarga da bateria
22 %      brake = sinal de corrente de recarga da bateria
23
24 % Definição da função
25 function [throttle, brake] = signals(current,throttle,brake,interations,k
26 )
27
28 for k = 1:interations,           % loop para percorrer todas as amostras
29 if current(k) > 0,             % sinal de corrente for positivo, descarga
30   throttle(k) = current(k);    % armazena o valor na variável throttle
31 else                           % sinal de corrente for negativo, recarga
32   throttle(k) = 0;            % armazena zero na variável throttle
33 brake(k) = (-1)*current(k);  % armazena o valor na variável brake e
34   inverte o sinal
35 end
36 end

```

#### A.4.3 bspeed.m

```

1 % Algoritmo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna a derivada da função brake, sinal de corrente de
8 % recarga da bateria, do sinal original de teste dinâmico
9 %
10 % Entrada:
11 %      brake = sinal de corrente de recarga da bateria
12 % Saída:
13 %      brake_speed = derivada do sinal brake
14
15 % Definição da função
16 function [brake_speed] = bspeed(brake)
17
18 brake_speed = diff(brake);        % calcula a derivada da função brake
19 % brake_speed(36880)=0; %Training data : A123_DYN_50_P25

```

```

20 brake_speed(36814)=0; %Test data : A123_DYN_50_P35
21 end

```

#### A.4.4 brelease.m

```

1  % Algorítimo KPI
2  %
3  % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4  % O algoritmo é de autoria do aluno Fábio Mori
5  %-----%
6  %
7  % Função que retorna o sinal de liberação de freio filtrado dentro dos
8  % parâmetros definidos
9  %
10 % Entradas:
11 %     brake = sinal de corrente de recarga da bateria
12 %     brake_speed = derivada do sinal de corrente de recarga da
13 %         bateria
14 %     interations = número de amostra de dados do sinal
15 %     k = número da amostra
16 % Saída:
17 %     brake_release = sinal filtrado de liberação do freio
18 %
19 % Definição da função
20 function [brake_release] = brelease(brake,brake_speed,interations,k)
21
22 for k = 1:interations, % loop para filtrar o sinal dentro das amostras
23 if (brake(k) > 3.5) && (brake_speed(k) < -2.5) && (brake_speed(k) > -3.0)
24
25     brake_release(k) = (-1)*brake_speed(k); % inverte e armazena o sinal na
26     % variável de saída
27 else % se fora dos limites do filtro
28     brake_release(k) = 0; % armazena valor nulo na variável de saída
29 end
30 end

```

#### A.4.5 bmeanrelease.m

```

1  % Algorítimo KPI

```

```

2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 % -----
6 %
7 % Função que retorna o KPI de liberação de freio e sua média após a
8 % filtragem da função "orelease.m"
9 %
10 % Entradas:
11 %     brake_release = função que filtra o sinal de liberação do
12 %     freio em um intervalo pre definido
13 %     interations = número de amostra de dados do sinal
14 %     k = número da amostra
15 % Saídas:
16 %     brake_release_kpi = KPI de liberação do freio
17 %     brake_release_kpi_mean = média móvel do KPI de liberação do
18 %     freio implementada como melhoria do sinal de entrada da RNA do
19 %     projeto final, desta forma o sinal sofre menos variações ao
20 %     longo do tempo.
21 %
22 % Definição da função
23 function [brake_release_kpi,brake_release_kpi_mean] = bmeanrelease(
24     brake_release,interations,k)
25 j = 0;
26 l = 1;
27 for k = 1:interations, % loop para percorrer todas as amostras
28
29 if brake_release(k) > 0 % se existir sinal brake release
30 j = j + 1;
31 brake_release_kpi(j) = (brake_release(k)); % armazena valor no KPI
32 if j == 1 % se primeiro valor
33 brake_release_kpi_mean(l) = (brake_release_kpi(j)/2);
34 l = l + 1;
35 else % para os demais valores, calcula a média ponderada
36 brake_release_kpi_mean(l) = ((brake_release_kpi(j-1))+((brake_release_kpi
37 (j))/2))/2;
38 l = l + 1;
39 end
40 else % se não existir sinal brake release

```

```

38 if j == 0 % se for primeira amostra
39 brake_release_kpi_mean(l) = 0;
40 l = l + 1;
41 else % senão mantém valor da amostra anterior
42 brake_release_kpi_mean(l) = brake_release_kpi_mean(l-1);
43 l = l + 1;
44 end
45 end
46 end

```

#### A.4.6 baggession.m

```

1 % Algorítimo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna o sinal de agressividade de freio filtrado dentro
8 % dos
9 %
10 % Entradas:
11 %     brake = sinal de corrente de recarga da bateria
12 %     brake_speed = derivada do sinal de corrente de recarga da
13 %     bateria
14 %     interations = número de amostra de dados do sinal
15 %     k = número da amostra
16 % Saída:
17 %     brake_aggression = sinal filtrado de agresisvidade do freio
18 %
19 % Definição da função
20 function [brake_aggression] = baggession(brake,brake_speed,interations,k
21 )
22 for k = 1:interations, % loop para filtrar o sinal dentro das amostras
23 if (brake(k) > 2) && (brake_speed(k) > 2) && (brake_speed(k) < 3.0),
24 brake_aggression(k) = brake_speed(k); % armazena o sinal na variável de
25 saída

```

```

25 else % se fora dos limites do filtro
26 brake_aggression(k) = 0; % armazena valor nulo na variável de saída
27 end
28 end

```

#### A.4.7 bmeanaggresssion.m

```

1 % Algorítimo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna o KPI de agressividade de freio e sua média após a
8 % filtragem da função "baggression.m"
9 %
10 % Entradas:
11 %     brake_aggression = função que filtra o sinal de agressividade do
12 %     freio em um intervalo pre definido
13 %     interations = número de amostra de dados do sinal
14 %     k = número da amostra
15 % Saídas:
16 %     brake_aggression_kpi = KPI de agressividade do freio
17 %     brake_aggression_kpi_mean = média móvel do KPI de agressividade
18 %         do freio implementada como melhoria do sinal de entrada da RNA do
19 %         projeto final, desta forma o sinal sofre menos variações ao
20 %         longo do tempo.
21
22 % Definição da função
23 function [brake_aggression_kpi_mean, brake_aggression_kpi] =
24     bmeanaggression(brake_aggression,interations,k)
25     j = 0;
26     l = 1;
27     for k = 1:interations, % loop para percorrer todas as amostras
28
29     if brake_aggression(k) > 0 % se existir sinal brake aggression
30     j = j + 1;
31     brake_aggression_kpi(j) = (brake_aggression(k)); % armazena valor no KPI
32     if j == 1 % se primeiro valor

```

```

29 brake_aggression_kpi_mean(l) = (brake_aggression_kpi(j)/2);
30 l = l + 1;
31 else % para os demais valores, calcula a média
32     ponderada
33     brake_aggression_kpi_mean(l) = ((brake_aggression_kpi(j-1))+(
34         brake_aggression_kpi(j))/2))/2;
35     l = l + 1;
36 end
37 else % se não existir sinal brake aggression
38 if j == 0 % se for primeira amostra
39     brake_aggression_kpi_mean(l) = 0;
40     l = l + 1;
41 else % senão mantém valor da amostra anterior
42     brake_aggression_kpi_mean(l) = brake_aggression_kpi_mean(l-1);
43     l = l + 1;
44 end

```

#### A.4.8 tspeed.m

```

1 % Algoritmo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna a derivada da função throttle, sinal de corrente de
8 % descarga da bateria do sinal original de teste dinâmico
9 %
10 % Entrada:
11 %      throttle = sinal de corrente de descarga da bateria
12 % Saída:
13 %      throttle_speed = derivada do sinal throttle
14 %
15 % Definição da função
16 function [throttle_speed] = tspeed(throttle)
17

```

```

18 throttle_speed = diff(throttle);      % calcula a derivada da função
19
20 % throttle_speed(36880)=0; %Training data : A123_DYN_50_P25
21 throttle_speed(36814)=0; %Test data : A123_DYN_50_P35
22
23 end

```

#### A.4.9 trelease.m

```

1    % Algoritmo KPI
2
3    % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4    % O algoritmo é de autoria do aluno Fábio Mori
5    %-----%
6
7    % Função que retorna o sinal de liberação do acelerador filtrado dentro
8    % dos parâmetros definidos
9
10   %
11   % Entradas:
12   %
13   %     throttle = sinal de corrente de descarga da bateria
14   %     throttle_speed = derivada do sinal de corrente de descarga da
15   %     bateria
16   %     interations = número de amostra de dados do sinal
17   %     k = número da amostra
18   %
19   % Saída:
20   %
21   %     throttle_release = sinal filtrado de liberação do acelerador
22
23   %
24   % Definição da função
25   function [throttle_release] = trelease(throttle,throttle_speed,
26   interations,k)
27
28
29   for k = 1:interations, % loop para filtrar o sinal dentro das amostras
30   if (throttle(k) > 4.5) && (throttle_speed(k) < -4.5) && (throttle_speed(k)
31   )> -5.0),
32   throttle_release(k) = (-1)*throttle_speed(k); % inverte e armazena o
33   % sinal na variável de saída
34   else % se fora dos limites do filtro
35   throttle_release(k) = 0; % armazena valor nulo na variável de saída
36   end
37 end

```

#### A.4.10 tmeanrelease.m

```

1 % Algoritmo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna o KPI de liberação do acelerador e sua média após a
8 % filtragem da função "trelease.m"
9 %
10 % Entradas:
11 %     throttle_release = função que filtra o sinal de liberação do
12 %     acelerador em um intervalo pre definido
13 %     interations = número de amostra de dados do sinal
14 %     k = número da amostra
15 % Saídas:
16 %     throttle_release_kpi = KPI de liberação do acelerador
17 %     throttle_release_kpi_mean = média móvel do KPI de liberação do
18 %     acelerador implementada como melhoria do sinal de entrada da RNA
19 %     do projeto final, desta forma o sinal sofre menos variações ao
20 %     longo do tempo.
21
22 % Definição da função
23 function [throttle_release_kpi,throttle_release_kpi_mean] = tmeanrelease(
24     throttle_release,interations,k)
25 j = 0;
26 l = 1;
27 for k = 1:interations,           % loop para percorrer todas as amostras
28
29 if throttle_release(k) > 0      % se existir sinal throttle release
30 j = j + 1;
31 throttle_release_kpi(j) = (throttle_release(k)); % armazena valor no KPI
32 if j == 1                      % se primeiro valor
33     throttle_release_kpi_mean(l) = (throttle_release_kpi(j)/2);
34     l = l + 1;
35 else                            % para os demais valores, calcula a média ponderada
36     throttle_release_kpi_mean(l) = ((throttle_release_kpi(j-1))+(
37         throttle_release_kpi(j))/2))/2;

```

```

36 l = l + 1;
37 end
38 else % se não existir sinal throttle release
39 if j == 0 % se for primeira amostra
40 throttle_release_kpi_mean(l) = 0;
41 l = l + 1;
42 else % senão mantém valor da amostra anterior
43 throttle_release_kpi_mean(l) = throttle_release_kpi_mean(l-1);
44 l = l + 1;
45 end
46 end
47 end

```

#### A.4.11 taggression.m

```

1 % Algorítimo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna o sinal de agressividade do acelerador filtrado
     dentro dos parâmetros definidos
8 %
9 % Entradas:
10 %      throttle = sinal de corrente de descarga da bateria
11 %      throttle_speed = derivada do sinal de corrente de descarga da
12 %      bateria
13 %      interations = número de amostra de dados do sinal
14 %      k = número da amostra
15 % Saída:
16 %      throttle_agression = sinal filtrado de agresisvidade do
     acelerador
17
18 % Definição da função
19 function [throttle_aggression] = taggression(throttle,throttle_speed,
     interations,k)
20
21 for k = 1:interations, % loop para filtrar o sinal dentro das amostras

```

```

22 if (throttle(k) > 3) && (throttle_speed(k) > 3) && (throttle_speed(k) <
23   4.5),
24   throttle_aggression(k) = throttle_speed(k); % armazena o sinal na variá-
25   vel de saída
26 else % se fora dos limites do filtro
27   throttle_aggression(k) = 0; % armazena valor nulo na variável de saída
28 end
29 end

```

#### A.4.12 tmeanaggression.m

```

1  % Algorítimo KPI
2 %
3 % Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
4 % O algoritmo é de autoria do aluno Fábio Mori
5 %-----%
6 %
7 % Função que retorna o KPI de agressividade do acelerador e sua média
8 % após a filtragem da função "taggression.m"
9 %
10 % Entradas:
11 %      throttle_aggression = função que filtra o sinal de agressividade
12 %      do acelerador em um intervalo pre definido
13 %      interations = número de amostra de dados do sinal
14 %      k = número da amostra
15 % Saídas:
16 %      throttle_aggression_kpi = KPI de agressividade do acelerador
17 %      throttle_aggression_kpi_mean = média móvel do KPI de
18 %      agressividade
19 %      do acelerador implementada como melhoria do sinal de entrada da
20 %      RNA do projeto final, desta forma o sinal sofre menos variações
21 %      ao longo do tempo.
22 %
23 % Definição da função
24 function [throttle_aggression_kpi_mean, throttle_aggression_kpi] =
25   tmeanaggression(throttle_aggression,interations,k)
26 j = 0;
27 l = 1;
28 for k = 1:interations, % loop para percorrer todas as amostras

```

```
27
28 if throttle_aggression(k) > 0 % se existir sinal throttle aggression
29 j = j + 1;
30 throttle_aggression_kpi(j) = (throttle_aggression(k)); % armazena valor
   no KPI
31 if j == 1                  % se primeiro valor
32 throttle_aggression_kpi_mean(l) = (throttle_aggression_kpi(j))/2;
33 l = l + 1;
34 else                      % para os demais valores, calcula a média
   ponderada
35 throttle_aggression_kpi_mean(l) = ((throttle_aggression_kpi(j-1)) + (
   throttle_aggression_kpi(j))/2))/2;
36 l = l + 1;
37 end
38 else                      % se não existir sinal throttle aggression
39 if j == 0                  % se for primeira amostra
40 throttle_aggression_kpi_mean(l) = 0;
41 l = l + 1;
42 else                      % senão, mantém valor da amostra anterior
43 throttle_aggression_kpi_mean(l) = throttle_aggression_kpi_mean(l-1);
44 l = l + 1;
45 end
46 end
47 end
```

## ANEXO B – ALGORITMOS EM PYTHON

Aqui vamos colocar os principais algoritmos utilizados no projeto e que referenciados neste trabalho feitos na linguagem Python. Todos eles possuem comentários explicando pontos importantes e caso exista o desejo você pode acessar o repositório do GitHub com todos estes arquivos e também os módulos adicionais que compõem tudo o que foi desenvolvido para chegar nos resultados deste trabalho, basta acessar o link a seguir: GitHub.

### B.1 Algoritmos da Rede Neural de aprendizagem profunda

#### B.1.1 SOC.py

```

1  # Algoritmo RNA
2  # Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
3  # Este algoritmo é de autoria do aluno Fábio Mori
4  ######
5  # Este algoritmo teve como base de desenvolvimento teve como base o
6  # conhecimento adquirido durante a realização das matérias:
7  # MITx - 6.00.1x
8  # Introduction to Computer Science and Programming Using Python
9  # MITx - 6.86x
10 # Machine Learning with Python-From Linear Models to Deep Learning
11 # IBM - DL0120EN
12 # Deep Learning with Tensorflow
13 # Todas estas matérias foram feitas pela plataforma edX
14 #####
15 # Rede Neural Artificial para estimar o SOC
16 # Baseado nos dados simulados em laboratório da célula A123
17 # Este código implementa a rede neural artificial profunda
18 # Dados de entrada: SOC estimado pelo SPKF e KPI dos dados UDDS
19 # Dado de saída: SOC estimado pela RNA
20
21 # O algoritmo inicia com a importação das bibliotecas e carregando os
22 # dados de teste e treinamento
23
24 import _pickle as cPickle, gzip
25 import numpy as np

```

```
25 from tqdm import tqdm
26 import torch
27 import torch.autograd as autograd
28 import torch.nn.functional as F
29 import torch.nn as nn
30 import matplotlib.pyplot as plt
31 import sys
32 sys.path.append("..")
33 import utils
34 from utils import *
35 from train import batchify_data, run_epoch, train_model, spkf_parameters
36
37 def main():
38     # Carregar os dados de treinamento e teste
39     # X: SOC SPKF, KPI BA, KPI BR, KPI TA, KPI TR
40     # Y: SOC REAL
41     num_classes = 10
42     X_train = np.loadtxt('../Datasets/x45.txt')
43     y_train = np.loadtxt('../Datasets/y45.txt')
44     X_test = np.loadtxt('../Datasets/x45.txt')
45     y_test = np.loadtxt('../Datasets/y45.txt')
46
47 ##########
48 # Vetores do eixo x para plotar os gráficos
49 time_train = len(y_train) + 1
50 train = np.arange(1, time_train, 1)
51 time_test = len(y_test) + 1
52 test = np.arange(1, time_test, 1)
53
54
55 # Plotando os dados y de treinamento e teste de entrada da Rede Neural
56 plt.plot(train, y_train, 'r', label='treinamento')
57 plt.plot(test, y_test, 'b', label='teste')
58 plt.title('Entrada RNA')
59 plt.ylabel('SOC')
60 plt.xlabel('Tempo (s)')
61 plt.legend(framealpha=1, frameon=True);
62 plt.show()
```

```
64 # Plotando os dados X de treinamento de entrada da Rede Neural
65 plt.plot(train, X_train[:,0], 'b', label='treinamento')
66 plt.plot(train, X_train, 'b', label='treinamento')
67 plt.title('Entrada RNA')
68 plt.ylabel('SOC')
69 plt.xlabel('Tempo (s)')
70 plt.legend(framealpha=1, frameon=True);
71 plt.show()
72
73 plt.plot(train, X_train[:,1], 'r', label='Agressividade')
74 plt.plot(train, X_train[:,2], 'b', label='Liberação')
75 plt.title('Entrada RNA')
76 plt.ylabel('KPI de Freio')
77 plt.xlabel('Tempo (s)')
78 plt.legend(framealpha=1, frameon=True);
79 plt.show()
80
81 plt.plot(train, X_train[:,3], 'r', label='Agressividade')
82 plt.plot(train, X_train[:,4], 'b', label='Liberação')
83 plt.title('Entrada RNA')
84 plt.ylabel('KPI do Acelerador')
85 plt.xlabel('Tempo (s)')
86 plt.legend(framealpha=1, frameon=True);
87 plt.show()
88
89 # Plotando os dados X de teste de entrada da Rede Neural
90 plt.plot(test, X_test[:,0], 'b', label='teste')
91 plt.title('Entrada RNA')
92 plt.ylabel('SOC')
93 plt.xlabel('Tempo (s)')
94 plt.legend(framealpha=1, frameon=True);
95 plt.show()
96
97 plt.plot(test, X_test[:,1], 'r', label='Agressividade')
98 plt.plot(test, X_test[:,2], 'b', label='Liberação')
99 plt.title('Entrada RNA')
100 plt.ylabel('KPI de Freio')
101 plt.xlabel('Tempo (s)')
102 plt.legend(framealpha=1, frameon=True);
```

```
103 plt.show()

104

105 plt.plot(test, X_test[:,3], 'r', label='Agressividade')
106 plt.plot(test, X_test[:,4], 'b', label='Liberação')
107 plt.title('Entrada RNA')
108 plt.ylabel('KPI do Acelerador')
109 plt.xlabel('Tempo (s)')
110 plt.legend(framealpha=1, frameon=True);
111 plt.show()

112 ######
113 # Em seguida é aplicado uma relação matemática para separar os dados de
114 # treinamento em validação 10% e treinamento 90%
115
116 # Dividir os dados de treinamento em validação (10%) e treinamento (90%)
117 dev_split_index = int(9 * len(X_train) / 10)
118 X_val = X_train[dev_split_index:]
119 y_val = y_train[dev_split_index:]
120 X_train = X_train[:dev_split_index]
121 y_train = y_train[:dev_split_index]

122 # Randomizar os dados de treinamento para inserir na rede
123 #     permutation = np.array([i for i in range(len(X_train))])
124 #     np.random.shuffle(permutation)
125 #     X_train = [X_train[i] for i in permutation]
126 #     y_train = [y_train[i] for i in permutation]

127
128 # Aplicamos a técnica de separação dos dados em batches para melhorar
129 # desempenho do algoritmo e posteriormente na sessão de Resultados será
130 # discutido seu
131 # valor ótimo

132 # Dividindo os dados em batches para o treinamento
133 batch_size_train = 128
134 batch_size_test = 1
135 train_batches = batchify_data(X_train, y_train, batch_size_train)
136 val_batches = batchify_data(X_val, y_val, batch_size_train)
137 test_batches = batchify_data(X_test, y_test, batch_size_test)
138
```

```

139 #####
140 # O modelo da Rede Neural Artificial Profunda foi definido como
141     # sequencial e utilizou a função de ativação Hardtanh, variação da
142     # tangente hiperbólica, que
143     # também será discutida durante a sessão de Resultados.
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170

```

#####

# O modelo da Rede Neural Artificial Profunda foi definido como

sequencial e utilizou a função de ativação Hardtanh, variação da

tangente hiperbólica, que

# também será discutida durante a sessão de Resultados.

#####

# Definindo modelo sequencial da Rede Neural Artificial Profunda

```

model = nn.Sequential(
    nn.Linear(5, 100),
    nn.Hardtanh(),
    nn.Linear(100, 100),      # camada oculta 1
    nn.Hardtanh(),
    nn.Linear(100, 1),
)

```

#####

lr=0.1

momentum=0

#####

# As funções trainmodel e runepoch foram criadas para executar as funções

durante o treinamento e teste da RNA.

#####

```

train_model(train_batches, val_batches, model, lr=lr, momentum=momentum)

# Validando o modelo com os dados de teste UDDS A123 à 25°C
# Calculando loss, error, accuracy e SOC estimado do modelo
loss, accuracy, error, sochat = run_epoch(test_batches, model.eval(),
                                           None)

print ("Loss on test set:" + str(loss) + " Accuracy on test set: " + str
       (accuracy))

loss_spkf, accuracy_spkf, error_spkf = spkf_parameters(X_test[:,0],
                                                       y_test)

```

#####

```

print ("Loss on SPKF set:" + str(loss_spkf) + " Accuracy on SPKF set: "
       + str(accuracy_spkf))

sochat = np.concatenate( sochat, axis=0 )

```

#####

# Plotando os resultados da Rede Neural Artificial Profunda

```

171 plt.title('Erro Quadrático Médio')
172 plt.plot(range(len(error)), error)
173 plt.ylabel('Função de perda')
174 plt.xlabel('Tempo (s)')
175 plt.show()
176
177 plt.plot(range(len(error)), sochat)
178 plt.ylabel('SOC')
179 plt.xlabel('Tempo (s)')
180 plt.show()
181
182 #plt.plot(range(len(test)), y_test, 'r', label='Real SOC')
183 plt.plot(range(len(y_test)), y_test, 'r', label='SOC Real')
184 #plt.plot(range(len(test)), sochat, 'b', label='ANN SOC Estimator')
185 plt.plot(range(len(sochat)), sochat, 'b', label='RNA SOC')
186 #plt.plot(range(len(test)), X_test[:,0], 'g', label='SPKF SOC Estimator')
187 plt.plot(range(len(y_test)), X_test[:,0], 'g', label='SPKF SOC')
188
189 plt.title('Saída RNA')
190 plt.ylabel('SOC')
191 plt.xlabel('Tempo (s)')
192 plt.legend(framealpha=1, frameon=True);
193 plt.show()
194
195 plt.plot(range(len(error)), sochat)
196 plt.ylabel('SOC')
197 plt.xlabel('Tempo (s)')
198 plt.show()
199
200 #####
201 #####
202 #####
203 if __name__ == '__main__':
204     # Especifique o "seed" para o comportamento determinístico e, em seguida,
205     # embaralhe.
206     np.random.seed(12321)    # para reproduzibilidade
207     torch.manual_seed(12321)  # para reproduzibilidade
208     main()

```

### B.1.2 train.py

```

1 # Algoritmo RNA
2 # Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
3 # O algoritmo foi adaptado pelo aluno Fábio Mori
4 ######
5 # Este algoritmo teve como base de desenvolvimento teve como base o
6 # conhecimento adquirido durante a realização da matéria:
7 # MITx - 6.86x
8 # Machine Learning with Python-From Linear Models to Deep Learning
9 # Essa matéria foi feita pela plataforma edX
10 #####
11 # Rede Neural Artificial para estimar o SOC
12 # Baseado nos dados simulados em laboratório da célula A123
13 # Este código contém funções auxiliares utilizadas pelo "SOC.py"
14 # Dados de entrada: SOC estimado pelo SPKF e KPI dos dados UDDS
15 # Dado de saída: SOC estimado pela RNA
16
17 from tqdm import tqdm
18 import numpy as np
19 import torch
20 import torch.nn.functional as F
21 import torch.nn as nn
22 import matplotlib.pyplot as plt
23
24 # Funções auxiliares
25 def batchify_data(x_data, y_data, batch_size):
26     """Takes a set of data points and labels and groups them into batches."""
27     # Separar os dados em lotes e usar apenas eles
28     N = int(len(x_data) / batch_size) * batch_size
29     batches = []
30     for i in range(0, N, batch_size):
31         batches.append({
32             'x': torch.tensor(x_data[i:i+batch_size], dtype=torch.float32),
33             'y': torch.tensor(y_data[i:i+batch_size], dtype=torch.float32)
34         })
35     return batches
36
37 def outing_numpy(predictions):

```

```
38
39     i = 0
40     lenght = len(predictions)
41     outing = np.zeros(lenght)
42
43     for i in range(lenght):
44         outing[i] = predictions[i]
45         i=i+1
46
47     return outing
48
49 def compute_SOC_accuracy(predictions, y):
50     # Calcular a accuracy do SOC estimado em relação ao SOC real
51     i = 0
52     lenght = len(predictions)
53     accepty = np.zeros(lenght)
54
55     for i in range(lenght):
56         bheight = y*1.05
57         blower = y*0.95
58
59         if predictions[i] > bheight[i]:
60             accepty[i] = 0
61             i=i+1
62         elif predictions[i] < blower[i]:
63             accepty[i] = 0
64             i=i+1
65         else:
66             accepty[i] = 1
67             i = i+1
68     return np.mean(accepty)
69
70     # O otimizador escolhido foi o Stochastic Gradient Descent, SGD e os parâmetros de loss, accuracy e error são calculados a cada época durante o treinamento
71     # baseado no valor estimado de SOC pela RNA.
72
73     def train_model(train_data, val_data, model, lr=0.01, momentum=0.9,
74                     nesterov=False, n_epochs=30):
```

```

74 " Treinar o modelo por N épocas com os dados e os hiper parâmetros "
75 # Utilizando o otimizador Stochastic Gradient Descent
76 optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=momentum,
    nesterov=nesterov)
77
78 for epoch in range(1, 101):
79     print("-----\nEpoch {}:\n{}".format(epoch))
80     # Treinamento
81     loss, acc, error, sochat = run_epoch(train_data, model.train(), optimizer)
82     print('Train loss: {:.6f} | Train accuracy: {:.6f}'.format(loss, acc))
83
84     # Validação
85     val_loss, val_acc, error, sochat = run_epoch(val_data, model.eval(),
86         optimizer)
86     print('Val loss: {:.6f} | Val accuracy: {:.6f}'.format(val_loss,
87         val_acc))
88
89     # Salvando o modelo
90     torch.save(model, 'mnist_model_fully_connected.pt')
91     return val_acc
92
93     # A função runepoch é responsável por inserir os dados no modelo
94     # sequencial, obter a resposta do SOC estimado e retornar os valores de
95     # error e accuracy médios.
96
97 def run_epoch(data, model, optimizer):
98
99     " Treinando o modelo para cada interação com os dados de treinamento e ..
100     "
101     " .. retornando os valores de loss e accuracy"
102     # Criando os vetores a serem armazenados as respostas
103     losses = []
104     batch_accuracies = []
105     outing = []
106     outing_soc = []
107     soc_estimating = []
108
109     # Se o modelo esta em treinamento, utilizar o otimizador

```

```
106 is_training = model.training
107
108 # Iteração através dos batches
109 for batch in tqdm(data):
110     # Criando o tensor com X e Y
111     x, y = batch['x'], batch['y']
112
113     # Inserindo dados no modelo e obtendo a resposta
114     out = (model(x))
115     outing.append(out)
116     outing_soc.append(outing_numpy(out))
117
118     # Calculando e armazenando o valor do accuracy
119     batch_accuracies.append(compute_SOC_accuracy(out, y))
120
121     # Calculando o loss
122     loss = F.mse_loss(out, y)
123     losses.append(loss.data.item())
124
125     # Se está em treinamento, faça a atualização
126     if is_training:
127         optimizer.zero_grad()
128         loss.backward()
129         optimizer.step()
130
131     # Calcular a média das épocas
132     avg_loss = np.mean(losses)
133     avg_accuracy = np.mean(batch_accuracies)
134
135 return avg_loss, avg_accuracy, losses, outing_soc
136
137 def spkf_parameters(data, y):
138
139     losses_spkf = []
140     batch_accuracies_spkf = []
141
142     soc_spkf = torch.from_numpy(data)
143     soc_real = torch.from_numpy(y)
```

```

145 # Calculando e armazenando o valor do accuracy do SPKF
146 batch_accuracies_spkf.append(compute_SOC_accuracy(soc_spkf,soc_real))
147
148 # Calculando o loss
149 loss_spkf = F.mse_loss(soc_spkf, soc_real)
150 losses_spkf.append(loss_spkf.data.item())
151
152 # Calcular a média das épocas
153 avg_loss_spkf = np.mean(losses_spkf)
154 avg_accuracy_spkf = np.mean(batch_accuracies_spkf)
155
156 return avg_loss_spkf, avg_accuracy_spkf, losses_spkf

```

### B.1.3 utils.py

```

1 # Algoritmo RNA
2 # Este arquivo foi utilizado no projeto de mestrado do aluno Fábio Mori.
3 ######
4 # Este algoritmo teve como base de desenvolvimento teve como base o
5 # conhecimento adquirido durante a realização da matéria:
6 # MITx - 6.86x
7 # Machine Learning with Python-From Linear Models to Deep Learning
8 # Essa matéria foi feita pela plataforma edX
9 #####
10 # A base do algoritmo util
11 # Rede Neural Artificial para estimar o SOC
12 # Baseado nos dados simulados em laboratório da célula A123
13 # Este código contém funções auxiliares utilizadas pelo "SOC.py"
14 # Dados de entrada: SOC estimado pelo SPKF e KPI dos dados UDDS
15 # Dado de saída: SOC estimado pela RNA
16
17 import pickle, gzip, numpy as np
18 import matplotlib.pyplot as plt
19 import matplotlib.cm as cm
20 import math
21
22 def plot_images(X):
23     if X.ndim == 1:
24         X = np.array([X])

```

```

25 num_images = X.shape[0]
26 num_rows = math.floor(math.sqrt(num_images))
27 num_cols = math.ceil(num_images/num_rows)
28 for i in range(num_images):
29     reshaped_image = X[i,:].reshape(28,28)
30     plt.subplot(num_rows, num_cols, i+1)
31     plt.imshow(reshaped_image, cmap = cm.Greys_r)
32     plt.axis('off')
33     plt.show()
34
35 def pick_examples_of(X, Y, labels, total_count):
36     bool_arr = None
37     for label in labels:
38         bool_arr_for_label = (Y == label)
39         if bool_arr is None:
40             bool_arr = bool_arr_for_label
41         else:
42             bool_arr |= bool_arr_for_label
43     filtered_x = X[bool_arr]
44     filtered_y = Y[bool_arr]
45     return (filtered_x[:total_count], filtered_y[:total_count])
46
47 def extract_training_and_test_examples_with_labels(train_x, train_y,
48     test_x, test_y, labels, training_count, test_count):
49     filtered_train_x, filtered_train_y = pick_examples_of(train_x, train_y,
50         labels, training_count)
51     filtered_test_x, filtered_test_y = pick_examples_of(test_x, test_y,
52         labels, test_count)
53     return (filtered_train_x, filtered_train_y, filtered_test_x,
54         filtered_test_y)
55
56 def write_pickle_data(data, file_name):
57     f = gzip.open(file_name, 'wb')
58     pickle.dump(data, f)
59     f.close()
60
61 def read_pickle_data(file_name):
62     f = gzip.open(file_name, 'rb')
63     data = pickle.load(f, encoding='latin1')

```

```
60     f.close()
61
62
63     def load_train_and_test_pickle(file_name):
64         train_x, train_y, test_x, test_y = read_pickle_data(file_name)
65
66         return train_x, train_y, test_x, test_y
67
68     # returns the feature set in a numpy ndarray
69     def load_CSV(filename):
70         stuff = np.asarray(np.loadtxt(open(filename, 'rb'), delimiter=','))
71
72         return stuff
```