
Universidade Federal de São Paulo
Instituto de Ciência e Tecnologia
Arquitetura e Organização de Computadores - Turma IB

# Predição de *End of Life* de baterias de veículos elétricos com modelagem computacional de baixo nível

**Integrantes:**
Felipe Freitas D Elia - 176.284
João Pedro Soldera Snabaitis Markues - 176.531
Lucas Gonzaga Prado - 176.572
Matheus Cahú Monteiro dos Santos - 176.623
**Docente responsável:** Profa. Dra. Thaína A. A. Tosta

São José dos Campos
2025/2


Universidade Federal de São Paulo
Instituto de Ciência e Tecnologia
Arquitetura e Organização de Computadores - Turma IB

# Predição de *End of Life* de baterias de veículos elétricos com modelagem computacional de baixo nível

Segundo relatório do projeto de Arquitetura e Organização de Computadores (Turma IB) da Universidade Federal de São Paulo para cumprimento dos requisitos de avaliação e aprovação na disciplina.

São José dos Campos
2025/2

**Resumo**

Com a crescente conscientização ecológica, o transporte elétrico se apresenta como uma alternativa promissora ao motor a combustão. Contudo, a viabilidade econômica e ambiental desta tecnologia reside na gestão eficaz da vida útil das baterias de íons de lítio, equipamentos caros e com potencial poluente significativo caso não sejam descartadas corretamente. A predição do *End of Life* (EoL) e a estimativa do *State of Health* (SoH) são, portanto, cruciais para otimizar o uso e mitigar os impactos ambientais e econômicos. Este trabalho tem como objetivo o desenvolvimento e a validação de um sistema de estimativa de SoH baseado em regressão linear. A implementação é realizada em *Assembly*, demonstrando o uso de modelagem computacional de baixo nível para realizar o treinamento do modelo (determinação dos coeficientes) a partir de dados de degradação da bateria e, posteriormente, a previsão do SoH.

# 1 Introdução

Diante dos desafios modernos de mobilidade e sustentabilidade, os veículos híbridos e elétricos se destacam como uma alternativa promissora. Durante o uso, eles emitem significativamente menos $CO_2$ em comparação aos veículos a combustão [1] — contribuindo para a redução de poluentes urbanos — e utilizam a energia de forma mais eficiente [2], já que convertem grande parte da energia armazenada em movimento útil, ao contrário dos motores a combustão, que desperdiçam boa parte em calor. Além disso, esses veículos abrem novas perspectivas, pois sua crescente comercialização — impulsionada por incentivos fiscais, redução de tarifas [3] e maior apelo social — estimula o desenvolvimento de tecnologias com tração elétrica. Esse aumento na demanda favorece a inovação, resultando em produtos com menores custos finais, maior autonomia e soluções cada vez mais eficientes.

Consonante à ascensão dessas tecnologias, no caso dos carros híbridos e elétricos, as baterias passaram a ser alvo de intenso debate, uma vez que a produção e o descarte ainda apresentam grandes impactos ambientais relevantes das quais vale ressaltar a degradação do solo proveniente da extração de lítio, cobalto [4] e níquel, matérias primas das baterias e a poluição química causada pelo seu descarte inadequado [5]. Dilemas econômicos e logísticos também são levados em consideração no debate [6], como o alto custo na substituição da bateria de um VE (veículo eletrificado) e a escassez de certos minerais, o que pode aumentar custos ou dificultar reposição.

Paralelamente, aumentar a vida útil da bateria influencia diretamente em quanto tempo o veículo pode ser usado antes de exigir a sua substituição [7]. Isto é uma característica essencial para mitigar estes impactos ambientais, já que menos baterias precisam ser produzidas ao longo da vida do carro e o número de descartes é reduzido, diminuindo a degradação ambiental causada pela mineração e minimizando o risco de poluição química. Além disso, os desafios econômicos são vencidos, pois com o adiamento da necessidade de substituição da bateria, o custo relativo referente à posse do carro diminui, visto que o mesmo veículo tem uma vida útil maior.

Nesse contexto, compreender como o desgaste da bateria se comporta e prever o fim da vida útil de baterias de carros elétricos é fundamental para a adoção de medidas que estendam esse prazo. Dessa forma, torna-se essencial antecipar as variações na capacidade total armazenada e identificar corretamente o fim da vida útil das baterias de íon-lítio [8]. O estudo em questão apresenta um método capaz de estimar, com elevada precisão, tanto o momento em que a bateria atinge o fim da descarga quanto o seu *End-of-Life* (EoL). Essa capacidade preditiva é obtida a partir de um modelo eletroquímico — que descreve o transporte de íons de lítio e as reações internas ao longo dos ciclos de carga e descarga — combinado ao Filtro de Kalman não linear (UKF) [9], ferramenta empregada para estimar o estado interno da bateria e seus parâmetros de envelhecimento com base em medições de tensão e corrente.

O trabalho é dividido em duas partes principais:

- **Estimativa de estados e parâmetros:** Durante o uso, o modelo estima o estado atual da bateria (nível de carga, tensão, etc.) em tempo real. Após cada ciclo de carga/descarga, ele atualiza os parâmetros de envelhecimento, como: capacidade total (quantidade de íons ativos); resistência interna e constante de difusão (ligada à velocidade dos íons).

- **Previsão (prognóstico):** Com base nesses parâmetros, é feita a previsão de fim

de descarga (quando a tensão cairá abaixo do limite) e a previsão de fim de vida útil (quando a capacidade cairá abaixo do limite de vida). Essas previsões usam simulações do modelo eletroquímico para diferentes condições de uso.

Portanto, este trabalho mostra que é possível combinar modelagem eletroquímica e algoritmos de estimativa (como o UKF) para acompanhar o envelhecimento da bateria para assim, realizar previsões de quando será necessário recarregá-la ou substituí-la.

## 1.1 Objetivos

A partir dessas motivações, pretende-se desenvolver um sistema computacional capaz de estimar o estado de saúde (*State of Health*, SoH) [10][11][12] de baterias de íons de lítio utilizadas em veículos elétricos, com base em variáveis operacionais como corrente, tensão, temperatura e número de ciclos.

- Construir um sistema que estime o SoH da bateria de veículos híbridos e elétricos.
- Implementar um sistema de regressão linear para calcular os parâmetros de cada métrica da bateria.
- Utilizar uma base de dados externa para treinamento do modelo.

# 2 Materiais e métodos

A regressão linear é um modelo de aprendizado de máquina supervisionado e, por este motivo, é necessário ter um conjunto de dados suficientemente grande, que contenha os parâmetros operacionais e a característica a ser estimada. A fim de realizar o treinamento, um algoritmo com algumas funções específicas deve ser implementado [13] [14].

## 2.1 Bases de dados

Para este trabalho, utilizou-se uma base de dados contendo os resultados de uma pesquisa da *National Aeronautics and Space Administration* (NASA) sobre baterias de íons de lítio [15]. O estudo envolveu a modelagem de três diferentes padrões de baterias para testes em laboratório, submetendo-as a ciclos contínuos de carga e descarga sob diferentes condições experimentais. O objetivo foi analisar como distintos parâmetros de descarga influenciam a vida útil remanescente e o estado geral de saúde das baterias. A base de dados é composta por sete variáveis de entrada e três variáveis de saída, descritas a seguir:

**Parâmetros de entrada:**

Tabela 1: Significado e Unidades dos Parâmetros de Entrada

| **Sigla** | **Nome Completo** | **Unidade** | **Descrição** |
|---|---|---|---|
| Cycle | Ciclos | – | O número de ciclos completos de carga e descarga, sendo o principal indicador de envelhecimento. |
| chI | Corrente de Carga | A (Ampères) | Corrente elétrica aplicada à bateria durante a fase de carga. |
| chV | Tensão de Carga | V (Volts) | Tensão elétrica da bateria medida durante a fase de carga. |
| chT | Temperatura de Carga | °C (Celsius) | Temperatura da bateria registrada durante o processo de carga. |
| disI | Corrente de Descarga | A (Ampères) | Corrente elétrica fornecida pela bateria durante a fase de descarga. |
| disV | Tensão de Descarga | V (Volts) | Tensão elétrica da bateria medida durante a fase de descarga. |
| disT | Temperatura de Descarga | °C (Celsius) | Temperatura da bateria registrada durante o processo de descarga. |
| BCt | Capacidade da Bateria | Ah (Ampère-hora) | Capacidade total de carga elétrica que a bateria consegue armazenar/fornecer no ciclo atual. |

**Parâmetros de saída:**

| **Variável** | **Unidade de medida** |
|---|---|
| Capacidade Medida da Bateria | Ah |
| Estado de Saúde | % |
| Vida Útil Restante | $n$ |

**Descrição das Variáveis**

- **Corrente de Carga e Descarga:** Corresponde à intensidade de corrente elétrica à qual a bateria é submetida. A corrente elétrica é a variação da carga elétrica em função do tempo, representada por $\frac{dQ}{dt}$. Em termos práticos, a corrente de carga ou descarga no ciclo $n$ pode ser aproximada pela razão entre a carga inicial do ciclo $Q_0(n)$ e a duração do respectivo processo ($T_c$ ou $T_d$), dada por:

$$\frac{Q_0(n)}{T_c} \quad \text{ou} \quad \frac{Q_0(n)}{T_d}$$

- **Tensão de Carga e Descarga:** Refere-se à diferença de potencial elétrico aplicada à bateria, representando a tensão entre seus terminais.

- **Temperatura de Carga e Descarga:** Corresponde à temperatura da bateria ao longo de um determinado ciclo.
- **Ciclo:** Um ciclo é definido como o período completo envolvendo uma etapa de carga seguida de uma etapa de descarga.
- **Capacidade Medida da Bateria:** Corresponde ao valor efetivo da capacidade total da bateria registrado durante o ciclo de descarga, expresso em ampère-hora (Ah).
- **Estado de Saúde (SoH):** Representa a proporção entre a capacidade atual da bateria ($Q(n)$) e sua capacidade inicial ($Q_0$), podendo ser expresso por:

$$SoH = \frac{Q(n)}{Q_0}$$

- **Vida Útil Restante (RUL):** Dado um limite de fim de vida (*End of Life* – EoL), a vida útil restante corresponde ao número de ciclos que a bateria ainda pode realizar antes de atingir esse limite. Pode ser estimada por:

$$RUL(n) = \frac{SoH(n) - SoH_{EoL}}{\text{Taxa Média de Degradação}(SoH/n)}$$

Referência: [8]

## 2.2 Regressão Linear com Gradiente Descendente

### 2.2.1 Modelo Matemático

Na regressão linear simples [13], busca-se ajustar uma reta que melhor descreve a relação entre uma variável independente $x$ e uma variável dependente $y$. O modelo é definido por:

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

onde:

- $\hat{y}_i$ é o valor previsto para a observação $i$;
- $\beta_0$ é o termo independente (intercepto);
- $\beta_1$ é o coeficiente angular (inclinação da reta);
- $x_i$ é a variável de entrada correspondente à observação $i$.

O objetivo é determinar $\beta_0$ e $\beta_1$ que minimizem o erro entre as previsões e os valores observados.

### 2.2.2 Função de Custo

Para medir o erro entre as previsões e os valores reais, define-se a função de custo como o erro quadrático médio:

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Substituindo $\hat{y}_i = \beta_0 + \beta_1 x_i$, temos:

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\beta_0 + \beta_1 x_i - y_i)^2$$

onde $m$ é o número de amostras do conjunto de dados. O fator $\frac{1}{2}$ é incluído para simplificar as derivadas.

### 2.2.3 Gradiente Descendente

O gradiente descendente é um método iterativo utilizado para minimizar a função de custo $J(\beta_0, \beta_1)$, ajustando os parâmetros em direção ao ponto de mínimo.

**Derivadas parciais** As derivadas parciais de $J$ em relação aos parâmetros são dadas por:

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)$$

$$\frac{\partial J}{\partial \beta_1} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) x_i$$

**Atualização dos parâmetros** Os parâmetros são atualizados iterativamente de acordo com:

$$\beta_0 := \beta_0 - \alpha \frac{\partial J}{\partial \beta_0}$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial J}{\partial \beta_1}$$

onde $\alpha$ é a *taxa de aprendizado*, responsável por controlar o tamanho do passo dado em cada iteração.

**Passo a Passo** A seguir, é apresentado um pequeno passo a passo do algoritmo gradiente descendente:

1. Inicialize os parâmetros $\beta_0$ e $\beta_1$ (geralmente com valores pequenos aleatórios).
2. Para cada iteração, realize os seguintes passos:
   (a) **Cálculo de previsões:**
   $$\hat{y}_i = \beta_0 + \beta_1 x_i$$

(b) **Cálculo dos gradientes:**

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$
$$\frac{\partial J}{\partial \beta_1} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_i$$

(c) **Atualização dos pesos:**

$$\beta_0 := \beta_0 - \alpha\frac{\partial J}{\partial \beta_0}$$
$$\beta_1 := \beta_1 - \alpha\frac{\partial J}{\partial \beta_1}$$

3. Repita essas instruções até que o erro seja suficientemente pequeno ou até atingir o número máximo de iterações.

**Considerações Finais**

O gradiente indica a direção de maior crescimento da função de custo, e o método de gradiente descendente ajusta os parâmetros na direção oposta a ele, buscando o ponto de mínimo [16]. A taxa de aprendizado $\alpha$ deve ser escolhida com cuidado:

- Se $\alpha$ for muito grande, o algoritmo pode ultrapassar o mínimo.
- Se $\alpha$ for muito pequena, a convergência será lenta.

## 2.3 Implementação

A implementação do sistema em Assembly foi estruturada em módulos lógicos que simulam um pipeline de Machine Learning. A seguir, detalham-se as etapas de manipulação de dados, preparação, treinamento e validação.

### 2.3.1 Manipulação de Arquivos

**2.3.1.1 Importação dos dados** Nesta etapa, utiliza-se a Syscall 13. Para essa operação, o processador exige a configuração prévia de registradores específicos: o endereço da string com o caminho do arquivo é carregado em `$a0`, enquanto o modo de abertura é definido em `$a1`. Neste projeto, `$a1` é configurado com o valor 0, instruindo o a abrir o arquivo apenas para leitura (*Read-Only*), protegendo os dados originais.

O retorno dessa operação é disponibilizado pelo sistema no registrador `$v0`. Este valor não é o conteúdo do arquivo, mas sim um *File Descriptor* (descritor de arquivo) – um identificador numérico que atua como um índice na tabela de arquivos abertos do processo. Por ser volátil, esse descritor é imediatamente movido de `$v0` para um registrador seguro (`$s0`) para ser utilizado nas etapas subsequentes. O código ainda implementa uma verificação se o valor em `$s0` é válido (não negativo) através de instruções de desvio (blt), garantindo que o programa não tente ler de um arquivo inexistente.

**2.3.1.2 Leitura dos dados** Após a obtenção do descritor de arquivo, inicia-se a transferência para a memória principal através da Syscall 14. Esta instrução requer a configuração precisa de três parâmetros nos registradores: `$a0` recebe o descritor (identificando o arquivo), `$a1` aponta para o endereço do buffer reservado na memória, e `$a2` define o limite máximo de bytes a serem lidos (150.000 bytes).

Ao executar a chamada, o buffer é preenchido com os caracteres ASCII do arquivo e retorna em `$v0` a contagem de bytes transferidos. Este valor é preservado no registrador `$s2`, pois representa o tamanho real do dataset. Através da syscall 16 o arquivo é fechado explicitamente utilizando o descritor. Após isso, uma instrução condicional (beq) verifica se a contagem de bytes lidos (`$s2`) é zero, interrompendo a execução caso o arquivo esteja vazio para evitar erros de processamento. Ao final dessa etapa, o código calcula o endereço final dos dados (Endereço Base do Buffer + Quantidade de Bytes Lidos) e insere o valor nessa posição para indicar que o conteúdo do arquivo terminou.

**2.3.1.3 Transformar em dados estruturados** O objetivo desta etapa é transformar o fluxo contínuo de bytes carregado no buffer em uma estrutura lógica de matriz (linhas e colunas) para cálculos matemáticos. Para isso, foi utilizado um Parser (analisador sintático) manual.

O processo se inicia ignorando a primeira linha do arquivo (metadados textuais). Para isso, é carregado o endereço inicial do buffer no registrador `$s1`. O algoritmo percorre o buffer (`$s1`) comparando cada byte com o código ASCII 10 (Nova Linha `\n`). Ao encontrá-lo, o cursor é posicionado imediatamente no início dos dados numéricos.

O arquivo é varrido caractere por caractere e o código monitora delimitadores específicos na Tabela ASCII para decidir como tratar o fluxo de dados: Vírgula (44), que indica o fim de uma célula (token); Nova Linha (10), que indica o fim de um registro (linha de dados); e Nulo (0), que indica o fim físico do arquivo e encerra o loop.

Ao identificar um delimitador válido (vírgula ou enter), o sistema realiza a tokenização: o delimitador é substituído na memória por um terminador nulo. Isso isola a sequência de caracteres anterior, transformando-a em uma String válida. O programa mantém dois ponteiros para essa operação: `$s1` (cursor atual) e `$s4` (marcador de início do número). O trecho isolado entre `$s4` e `$s1` é então submetido à função de conversão numérica (`converter_e_salvar_float`), exceto para a Coluna 0 (ID da bateria), que é descartada.

O registrador `$s3` atua como índice da coluna atual. A cada token processado, `$s3` é incrementado. Ao atingir o limite de 11 colunas (tamanho do dataset), o sistema reconhece o fim da linha, incrementa o contador global de linhas (`contador_linhas`) e reseta `$s3` para zero. Essa lógica permite organizar os dados linearmente na memória RAM. Ao final, ao concluir a varredura, o sistema exibe no console o total de linhas processadas.

**2.3.1.4 Tratamento dos dados** Depois que os dados foram isolados pelo parser, foi implementado um algoritmo que converte a representação textual em valores de ponto flutuante binários. A conversão baseia-se na lógica posicional decimal. Para isso, o algoritmo percorre a string caractere por caractere, realizando três etapas principais dentro de um loop que envolvem: a obtenção do valor numérico inteiro; transição desse valor inteiro para registradores de ponto flutuante; e aplicação da fórmula para converter os dígitos em um número só. Para isso, é aplicada a fórmula Valor = Acumulador $\times$ 10.0 + novo dígito. O registrador `$f10` é pré-carregado com a constante 10.0 e o acumulador `$f0` inicia zerado.

Para salvar o valor convertido (`$f0`) no local correto, o sistema calcula o endereço

físico utilizando a fórmula:

$$\text{Endereço} = \text{BaseArray}_{\text{coluna}} + (\text{ÍndiceLinha} \times 4)$$

O código acessa uma tabela de referência (`ponteiros_arrays`) para obter o endereço base da coluna atual (ex: vetor de Voltagem) e soma a ele o deslocamento calculado com base no contador de linhas. A instrução `swc1` (Store Word Coprocessor 1) finaliza o processo, gravando o número de ponto flutuante na célula de memória calculada.

**2.3.2 Divisão de Treino x Teste** A partir desse ponto, inicia-se a função `Iniciar_machine_learni` Nessa etapa, o código será responsável por fazer a segmentação dos dados entre conjunto de treinamento (70%) e teste (30%). A partir do número de linhas calculado anteriormente, realiza-se a multiplicação em ponto flutuante para se obter as linhas de treino e as linhas de teste. $0.7 \times$ número de linhas indica o limite das linhas de treino. O resultado desta operação é armazenado na variável `n_treino` (representando o limite de iterações do algoritmo de aprendizado). A variável complementar, `n_teste`, é calculada por subtração simples (Total - número de linhas de treino) e armazenada.

**2.3.3 Normalização** Em Assembly é necessário implementar um Loop Externo. Este loop itera sobre o índice das colunas (de 0 a 8), utilizando a Tabela de Ponteiros para carregar o endereço base do vetor correspondente (`$s0`) a cada iteração, garantindo que o tratamento estatístico seja aplicado sequencialmente a cada característica. O algoritmo realiza duas passagens completas sobre os dados de cada coluna para extrair os parâmetros estatísticos:

Média Aritmética: Um loop de acumulação soma todos os valores do vetor. A limitação da arquitetura do MIPS exige a conversão do contador de linhas (Inteiro) para Ponto Flutuante (cvt.s.w) antes de realizar a divisão final, armazenando o resultado no registrador `$f10`.

Desvio Padrão: Utilizando a média calculada, um segundo loop computa a soma das diferenças quadráticas. O resultado é dividido pelo número de amostras e submetido à instrução de raiz quadrada (sqrt.s). O código implementa um mecanismo (bc1f) para evitar erros de divisão por zero. Caso a variância de uma coluna seja nula (todos os valores iguais), o desvio padrão é forçado para 1.0, garantindo estabilidade numérica do sistema.

Imediatamente após o cálculo, a Média e o Desvio Padrão de cada coluna são salvos em vetores dedicados na memória (`array_medias` e `array_stds`), indexados pelo contador de colunas. Essa etapa é necessária para desnormalizar os resultados finais. A etapa final aplica a transformação normalização a cada elemento. O valor bruto é carregado do endereço de memória, processado na FPU (unidade de ponto flutuante) e o resultado normalizado sobrescreve imediatamente o valor original no mesmo endereço físico (`swc1`). Ao final do processo, a memória RAM contém apenas os dados padronizados para o algoritmo.

**2.3.4 Treinamento (Gradiente Descendente)** O núcleo do algoritmo está no Loop de Épocas (`loop_epoch`), responsável por repetir o processo de ajuste de pesos múltiplas vezes (neste projeto, fixado em 1000 iterações) para minimizar a função de custo. Para isso, o registrador `$s7` atua como o iterador incremental, representando a época atual. O registrador `$s6` armazena a constante epochs, que representa o número máximo de ciclos.

No início de cada ciclo, o processador compara o contador com o limite. Caso sejam iguais, o fluxo é desviado para a rotina de encerramento (`fim_treino`).

**2.3.4.1 Reset do Erro Acumulado** No início do loop é realizado o reset dos acumuladores.

Distinção entre Pesos (W) e Gradientes (`grad_W`): O Vetor W (Pesos) representa o "conhecimento"acumulado pelo modelo, sendo uma estrutura persistente que evolui continuamente. Já o Vetor `grad_W` (Gradientes) atua como um acumulador temporário (volátil) que armazena a somatória dos erros apenas da época corrente.

O vetor de direção (`grad_W`) deve ser calculado exclusivamente com base nos erros da rodada atual. Se os valores da época anterior permanecessem na memória, o erro se acumularia indevidamente (Erro Época 1 + Erro Época 2...), gerando vetores de ajuste com magnitude exagerada e impedindo a convergência.

Para evitar isso, o código implementa uma rotina de limpeza no início. O endereço base do vetor de derivadas (`grad_W`) é carregado em `$t0` e o FPU é preparado com um valor nulo (0.0) no registrador `$f0`. Dessa maneira, o sistema realiza uma varredura na memória para zerar os vetores de gradiente (`grad_W` e `grad_b`). No Assembly, isso é feito através de um loop dedicado (`zerar_grads`) que utiliza aritmética de ponteiros para percorrer cada posição do array de pesos na RAM, sobrescrevendo os valores antigos com 0.0.

**2.3.4.2 Cálculo da Predição (Produto Escalar)** Depois que os gradientes foram zerados, iniciamos o cálculo da previsão para a linha atual. A fórmula é: $\hat{Y} = (W_0 \cdot X_0) + (W_1 \cdot X_1) + ... + b$, sendo iterado característica por característica. O jeito mais fácil de somar isso no processador é começar pelo $b$ e ir somando o resto.

O registrador `$f4` atuará como a variável temporária da predição, servindo de base para as somas subsequentes dos produtos entre pesos e características. Ele começa valendo $b$ (o valor atual de $b$) e vai crescer conforme multiplicamos os pesos.

O loop principal de treinamento (`loop_linha`) é configurado para iterar de 0 até `n_treino` (os 70% dos dados definidos anteriormente).

Recuperação do dado de entrada $X_i$:

- Seleção da Coluna (característica): O índice da coluna (`$t1`) é utilizado para acessar a `ponteiros_arrays` e recuperar o endereço base do vetor correspondente (ex: vetor de Temperatura).
- Seleção da Linha (valor): O índice da linha atual (`$t9`) é multiplicado por 4 (tamanho da palavra) para calcular o deslocamento dentro desse vetor específico.
- A soma desses dois componentes resulta no endereço físico exato do dado, que é carregado no registrador `$f6`.

Simultaneamente, o peso correspondente ($W_j$) é carregado de forma linear no registrador `$f5`.

Com os valores carregados na FPU, é iniciado o cálculo do produto escalar. Na multiplicação, o valor da característica é ponderado pelo seu peso (`mul.s $f6, $f6, $f5`). Logo após, o resultado dessa multiplicação é somado ao registrador acumulador `$f4` (`add.s`). `$f4` foi inicializado com o valor do $b$, garantindo que a equação final da reta seja respeitada.

Ao final de 8 iterações (uma para cada característica), o registrador `$f4` contém a predição completa $\hat{Y}$ para a bateria atual, ou seja, acumula o resultado da equação da reta para a bateria atual.

**2.3.4.3 Cálculo do Erro** Imediatamente após a predição, o sistema precisa avaliar a precisão do modelo comparando o valor estimado ($\hat{Y}$) com o valor real ($Y$).

O valor real do SOH encontra-se armazenado na 9ª coluna da estrutura de dados (índice 8). Para acessá-lo, o algoritmo calcula o endereço base do vetor alvo somando um deslocamento fixo de 32 bytes (8 índices $\times$ 4 bytes) ao endereço da tabela de ponteiros.

Combinando isso com o deslocamento da linha atual, recupera-se o valor de $Y$ para o registrador `$f7`.

A operação fundamental do aprendizado é realizada através da subtração em ponto flutuante (`sub.s`): Erro (dif) = $\hat{Y}$(\$f4) $-$ $Y$(\$f7).

O resultado, armazenado no registrador temporário `$f8`, representa o resíduo da predição: valores positivos indicam superestimação, enquanto negativos indicam subestimação.

**2.3.4.4 Cálculo e Acumulação do gradiente do Bias** O erro calculado em `$f8` é imediatamente somado à variável global `grad_b` na memória. Esta acumulação contínua garante que, ao final do loop de todas as linhas de treino, a variável `grad_b` contenha o somatório total dos erros $\sum(\hat{Y} - Y)$ que é o componente principal para a derivada parcial do Viés.

**2.3.4.5 Cálculo e Acumulação dos Gradientes dos Pesos** Após determinar o erro escalar da predição (dif), o algoritmo precisa calcular a contribuição individual de cada uma das 8 características para esse erro.

Iteração sobre Características (Features): O código inicia um loop interno (`loop_grad`) que percorre as colunas de 0 a 7. Para cada iteração, recupera-se o valor de entrada $X_{ij}$ correspondente à linha e coluna atuais, utilizando a mesma lógica de Tabela de Ponteiros empregada no Cálculo da predição ($\hat{Y}$).

Durante esse loop, a instrução `mul.s $f6, $f6, $f8` executa o núcleo matemático do gradiente descendente para regressão linear: multiplica o valor da característica $X$ pelo erro residual (dif).

Se o valor de entrada $X$ for alto, ele magnifica o gradiente, indicando que aquele peso tem grande responsabilidade no erro final. Se $X$ for zero, o gradiente resultante é nulo, preservando o peso atual.

O resultado dessa multiplicação não substitui o valor anterior, mas é somado a ele. O algoritmo carrega o acumulador `grad_W[j]` da memória, adiciona o novo gradiente parcial (`add.s`) e salva o resultado (`swc1`).

Essa acumulação contínua ao longo de todas as linhas de treino garante que, ao final da época, o vetor `grad_W` contenha a média vetorial da direção de descida para todo o conjunto de dados, permitindo um ajuste de pesos estável e representativo.

**2.3.4.6 Cálculo do Viés (b)** Após a conclusão do loop que processa as linhas de treino (`loop_linha`), os acumuladores de gradiente contêm a soma total dos erros da época. O sistema então transita para a fase de atualização dos parâmetros. A primeira operação foca no Viés ($b$), o termo independente da equação linear. A função `update_pesos` é responsável por atualizar o Viés ($b$).

A regra de atualização do Gradiente Descendente padrão é definida por $\theta_{\text{antigo}} - \alpha \cdot \frac{1}{m} \sum \nabla$.

O código calcula um fator de escala único dividindo a Learning Rate ($\alpha = 0.01$) pelo número de amostras ($m = 700$). O resultado desse fator de escala, armazenado em `$f1`, representa a 'Taxa Efetiva', que será reutilizada para atualizar tanto o Viés ($b$) quanto os Pesos ($W$), eliminando a necessidade de dividir cada gradiente individualmente por $m$.

Nesse caso, o novo valor do Viés ($b$) será dado por $b \leftarrow b - \alpha \frac{1}{m} \sum(\hat{Y} - Y)$.

Essa atualização do viés é realizada em três instruções: O valor acumulado em `grad_b` (Soma dos Erros) e o valor atual de $b$ são carregados na FPU; a soma dos erros é multiplicada pela Taxa Efetiva (`mul.s`); e o valor resultante é subtraído do Viés antigo (`sub.s`). Portanto, se a soma dos erros for positiva (superestimação), o Viés é reduzido; se negativa, é aumentado.

**2.3.4.7 Cálculo dos coeficientes das características (W)** Após o ajuste do viés, o sistema entra no loop `loop_update` para ajustar os coeficientes das 8 características ($W$). Diferente da etapa anterior, esta operação requer acesso simultâneo a dois vetores distintos na memória: o vetor de Pesos Atuais e o vetor de Gradientes Acumulados.

Acesso Sincronizado à Memória: O loop itera o índice $j$ de 0 a 7. Utilizando aritmética de ponteiros, o algoritmo calcula o deslocamento (offset) baseando-se no índice atual e acessa: O peso $W_j$ antigo (endereço base $W$ + offset) e o gradiente correspondente $W_j$ (endereço base `grad_W` + offset).

A correção é aplicada individualmente para cada peso. $W_j \leftarrow W_j - (\text{Taxa Efetiva} \times \nabla W_j)$.

O gradiente específico da característica é multiplicado pela taxa efetiva (previamente calculada em `$f1`) e subtraído do peso original. O novo valor sobrescreve o antigo na memória (`swc1`).

**2.3.4.8 Próxima época** Ao concluir as 8 iterações, o fluxo é desviado para `prox_epoch`, onde o contador global de épocas (`$s7`) é incrementado. O ciclo reinicia-se completamente até que o limite predefinido (1000 épocas) seja atingido.

**2.3.5 Teste Final** O Objetivo do teste é quantificar a capacidade de generalização do modelo utilizando os 30% dos dados que foram segregados no início do programa. No Assembly, é necessário reconstruir manualmente o loop de predição, mas com uma diferença: os pesos ($W$ e $b$) agora são constantes. Não há mais atualização, apenas medição. Serão, portanto, utilizados os pesos ($W$ e $b$) obtidos durante o treinamento.

**2.3.5.1 Configuração do Loop de Teste** O contador de linhas (`$t9`) é inicializado com o valor de `n_treino` (ex: 700). Isso garante que a validação comece na linha nº 701. Também é criado um Acumulador Global: O registrador `$f20` é limpo e reservado exclusivamente para armazenar a Soma dos Erros Quadráticos de todas as linhas de teste.

**2.3.5.2 Cálculo da predição com os pesos treinados** Antes de realizar o cálculo da predição, os valores são normalizados a partir da Média e do Desvio Padrão históricos (salvos durante o treino) correspondentes àquela coluna específica. O algoritmo entra em um loop que vai de 700 até o final do arquivo. Para cada linha, executa-se o cálculo da predição idêntico ao do treino: $\hat{Y} = b + \sum_{j=0}^{7}(W_j \cdot X_{ij})$. A diferença fundamental é que, nesta etapa, não há cálculo de gradiente nem atualização de pesos.

**2.3.5.3 Cálculo dos Erros dos Testes** Após obter a predição ($\hat{Y}$), o sistema recupera o valor real ($Y$) da memória e calcula o resíduo. Para a métrica de desempenho, utiliza-se o erro ao quadrado:

$$Erro^2 = (\hat{Y} - Y) \cdot (\hat{Y} - Y)$$

A elevação ao quadrado é realizada via multiplicação simples (`mul.s $f8, $f8, $f8`). Esta operação cumpre dois papéis: elimina sinais negativos e penaliza desproporcionalmente grandes desvios. O resultado é somado ao acumulador global `$f20`.

**5.4 Média final dos erros do Teste** Ao concluir a varredura de todas as linhas de teste, o acumulador `$f20` contém a soma total dos erros quadrados. Para obter o Erro Quadrático Médio (MSE), o código realiza uma divisão final pelo número de amostras de teste:

$$MSE = \frac{\sum Erro^2}{2 \cdot N_{\text{teste}}}$$

O resultado final, armazenado em `$f12`, é exibido no console via syscall 2.

**2.3.6 Impressão dos Pesos Aprendidos** Finalizado o processo de aprendizagem e validação, o sistema deve apresentar os coeficientes da regressão linear ao usuário. O código carrega o valor final de $b$ da memória para o registrador `$f12` e executa a chamada. Para imprimir os valores de $W$, o código inicializa um Loop de Impressão. Nele, Define-se um índice inicial (`$t0` = 0) e o limite de iteração (`$t1` = 8) correspondente ao número de características.

**2.3.7 Interação com o usuário**

**2.3.7.1 Predição a Partir dos Dados Fornecidos pelo Usuário** Essa parte permite que o usuário insira parâmetros manuais para obter estimativas instantâneas de saúde da bateria (SOH). Nesse caso, o valor base da regressão, $Y'$, é carregado para o registrador `$f12`. No loop principal da interação (`loop_inputs`), ao invés de armazenar as entradas do usuário em um buffer intermediário, o algoritmo realiza o cálculo da predição de forma acumulativa e imediata conforme o usuário for colocando o valor das características. O dado normalizado é multiplicado pelo peso (W) da característica, carregado da memória de pesos treinados e o resultado é somado ao registrador `$f12` (que já continha o Viés). Complementando, os valores de entrada são normalizados a partir da Média e o Desvio Padrão históricos (salvos durante o treino) correspondentes àquela coluna específica.

**2.3.7.2 Desnormalização do SOH obtido** A estimativa acumulada no registrador `$f12` ao final do loop de inputs representa o valor de SOH na escala padronizada (Z-Score). Para apresentar um resultado compreensível ao usuário final (porcentagem real de saúde), o sistema deve inverter a transformação aplicada durante o treinamento. A desnormalização é executada aplicando a função inversa: $SOH_{real} = (SOH_{norm} \cdot \sigma_{SOH}) + \mu_{SOH}$

**2.3.7.3 Finalização** Após exibir o resultado, o sistema solicita ao usuário se deseja realizar um novo teste. Caso sim, o fluxo retorna ao rótulo **input_usuario**. Caso contrário, o fluxo segue linearmente para a chamada de sistema **exit**.

### Uso dos Registradores na Parte 1 (Manipulação de Arquivos)

| Registrador | Função Resumida |
|---|---|
| $v0 | Códigos e Retornos. Recebe o código (13, 14, 16) e retorna o FD ou bytes lidos. |
| $a0 | Arquivo. Passa o endereço do nome do arquivo (abertura) ou o FD (leitura). |
| $a1 | Modo/Buffer. Define modo leitura (0) ou endereço de destino na RAM. |
| $a2 | Tamanho. Define o limite máximo de leitura (150.000 bytes). |
| $s0 | File Descriptor (FD). Guarda o ID do arquivo de forma segura durante o processo. |
| $s1 | Cursor. Ponteiro que percorre o texto do arquivo byte a byte na memória. |
| $s2 | Tamanho do Arquivo. Total de bytes lidos; define o fim do loop de leitura. |
| $s3 | Contador de Colunas. Índice lógico (0 a 10) para saber qual dado está sendo lido. |
| $s4 | Âncora de Token. Marca o início da string do número atual para conversão. |
| $f0 | Acumulador ATOF. Constrói o número float dígito a dígito. |
| $f10 | Constante 10.0. Multiplicador usado na lógica posicional decimal. |

### Uso dos Registradores na Parte 2 (Estruturação de Dados)

| Registrador | Função Específica nesta Etapa |
|---|---|
| $s0 | Ponteiro Base da Coluna. Armazena o endereço inicial do vetor atual na memória RAM. É atualizado a cada troca de coluna (baseado na Tabela |
| $f10 | Acumulador da Média (μ). Primeiro soma todos os valores (X) e, após a divisão, armazena a Média Aritmética final da coluna. |

### Uso dos Registradores na Função ATOF (Conversão ASCII para Float)

| Registrador | Função Resumida |
|---|---|
| $s7 | Contador de Épocas. Itera de 0 até o limite (1000). |
| $s6 | Limite de Épocas. Armazena a constante de paragem (1000). |
| $t9 | Contador de Linhas. Percorre as baterias de treino (0 a 700). |
| $t1 | Contador de Colunas. Percorre as 8 características (features) nos loops internos. |
| $f4 | Predição (Y^). Acumulador que soma b+∑(W·X). |
| $f7 | Valor Real (Y). O "gabarito" (SOH Real) lido da memória. |
| $f8 | Erro (dif). Resultado da subtração: Predição - Real. |
| $f6 | Input (X). Valor da característica atual (ex: Voltagem). |
| $f5 | Peso (W). Valor do peso atual sendo usado ou atualizado. |
| $f1 | Taxa Efetiva. Learning Rate dividida pelo nº de linhas (α/m). |
| $f9 | Gradiente de W. Acumulador temporário para somar o erro ponderado. |
| grad_b | Acumulador do Bias. Soma os erros escalares de todas as linhas. |
| grad_W | Vetor de Gradientes. Soma os erros ponderados pelos inputs. |

### Uso dos Registradores nas Partes 3 e 4 (Normalização e Treinamento)

| Registrador | Função Resumida |
|---|---|
| $t9 | Índice de Teste. Inicializado com o valor de n_treino (700) para garantir o acesso apenas aos dados de validação. |
| $f20 | Acumulador Global (SSE). Soma os erros quadráticos de todas as linhas de teste para o cálculo final. |
| $f8 | Erro Quadrático. Calcula o resíduo (Y^−Y) e é multiplicado por si mesmo (mul.s) para penalizar desvios. |
| $f12 | Resultado MSE. Armazena a Média do Erro Quadrático calculada ao final do loop para exibição (syscall 2). |

# 3 Resultados

Com o projeto finalizado, é possível realizar testes para concluir se o algoritmo está calculando as estimativas de forma correta. Assim, os resultados descritos a seguir demonstram a eficácia do algoritmo e a consistência das predições.

Ao executar o algoritmo, é informado que o conjunto de dados foi aberto e a quantidade de linhas lidas. Em seguida, o treinamento é iniciado usando o conjunto exclusivo para treino e, após a sua finalização, é possível visualizar o erro obtido utilizando o conjunto de testes. Vale ressaltar que o erro obtido, considerando as casas decimais apresentadas, é semelhante para cada execução, desde que seja utilizado o mesmo conjunto de dados, pois é garantido que o erro convergirá devido à eficiência do método da descida do gradiente. Os pesos obtidos para cada parâmetro também são apresentados ao usuário.

Com os pesos encontrados, chegou o momento de estimar a saúde da bateria conforme entradas fornecidas pelo usuário. Os dados são pedidos nesta ordem:

Tabela 2: Siglas e Nomenclatura dos Parâmetros de Entrada

| **Sigla** | **Nome Completo** |
|---|---|
| Cycle | Ciclos |
| chI | Corrente de Carga (Charge Current) |
| chV | Tensão de Carga (Charge Voltage) |
| chT | Temperatura de Carga (Charge Temperature) |
| disI | Corrente de Descarga (Discharge Current) |
| disV | Tensão de Descarga (Discharge Voltage) |
| disT | Temperatura de Descarga (Discharge Temperature) |
| BCt | Capacidade da Bateria (Battery Capacity) |

Um item foi aleatoriamente selecionado para aferimento dos dados calculados. As variáveis não apresentam o ponto delimitando o início das casas decimais depois da variável *Cycle* para facilitar a trativa de dados; todas as outras colunas teriam pontos antes da segunda casa dos números, contudo, foi retirada a necessidade para minimizar os erros humanos.

Tabela 3: Valores de Entrada Utilizados no Teste 1

| **cycle** | **chI** | **chV** | **chT** | **disI** | **disV** | **disT** | **BCt** |
|---|---|---|---|---|---|---|---|
| 114 | 12022955 | 424221432613 | 2445969624 | 2074759012 | 3321320595 | 3502998401 | 1431206520 |

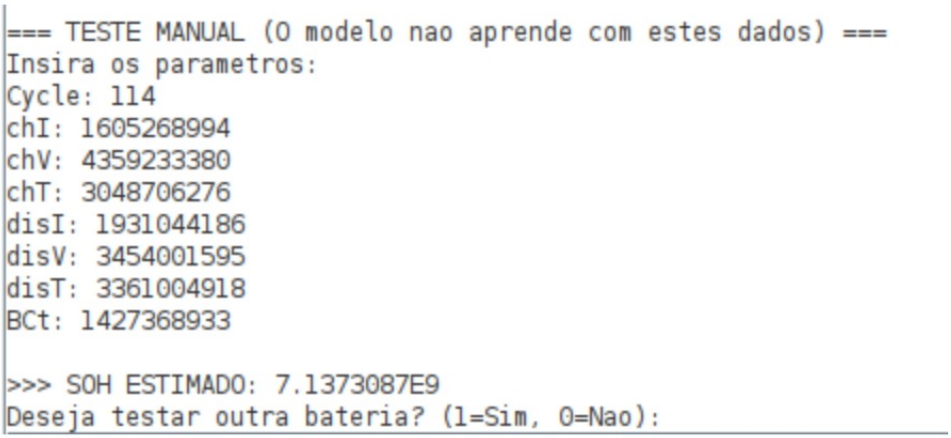


Figura 1: Tela de execução demonstrando os dados de entrada sem pontuação decimal e o resultado da predição.

O valor real do *SoH* é 7,136844666, enquanto o valor estimado pelo modelo foi 7,137087. Demonstrando uma similaridade satisfatória.

Realizando o mesmo teste para outra bateria em um estado de degradação mais avançado, ao passar as suas informações é obtido:

Tabela 4: Valores de Entrada Utilizados no Teste 2

| **cycle** | **chI** | **chV** | **chT** | **disI** | **disV** | **disT** | **BCt** |
|---|---|---|---|---|---|---|---|
| 215 | 1442324745 | 4222444475 | 2583212476 | 1922813798 | 3055228860 | 3072274916 | 0919643710 |

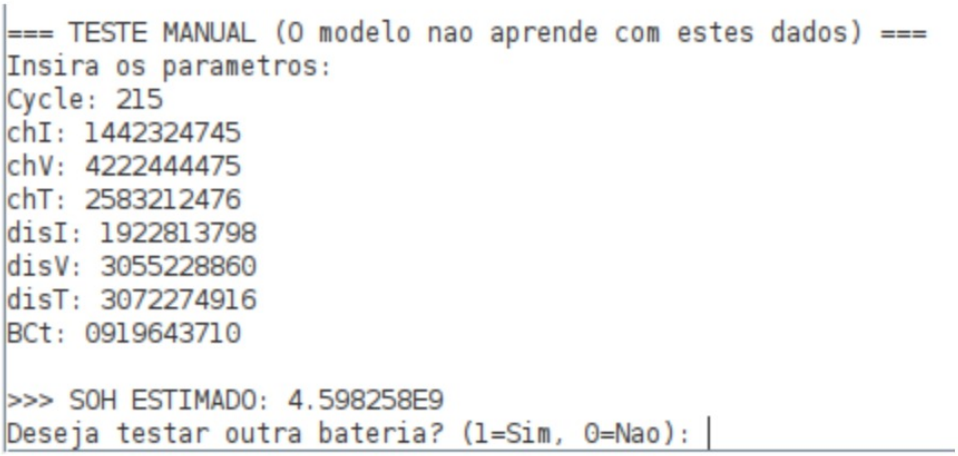


Figura 2: Tela de execução demonstrando os dados de entrada e o resultado da predição.

O valor real do *SoH* é 4,598218552, enquanto o valor estimado pelo modelo foi 4,598258. Demonstrando uma similaridade alta entre os valores.

Realizando o mesmo teste para outra bateria após o primeiro ciclo.

Tabela 5: Valores de Entrada Utilizados no Teste 3

| cycle | chI | chV | chT | disI | disV | disT | BCt |
|---|---|---|---|---|---|---|---|
| 1 | 1440146910 | 4254681794 | 2398873275 | 1894406780 | 3273522542 | 3298083419 | 1986195795 |

```
=== TESTE MANUAL (O modelo nao aprende com estes dados) ===
Insira os parametros:
Cycle: 1
chI: 1440146910
chV: 4254681794
chT: 2398873275
disI: 1894406780
disV: 3273522542
disT: 3298083419
BCt: 1986195795

>>> SOH ESTIMADO: 9.930979E9
Deseja testar outra bateria? (1=Sim, 0=Nao):
```

Figura 3: Tela de execução demonstrando os dados de entrada e o resultado da predição.

O valor real do *SoH* é 9,930978979, enquanto o valor estimado pelo modelo foi 9,930979. Demonstrando uma similaridade alta entre os valores.

Por fim, é fundamental destacar que o programa não se limita a realizar estimativas apenas com base nas amostras já coletadas no dataset. O modelo permite a inserção de qualquer conjunto de valores de entrada inéditos que ainda não foram obtidos para predição. Para demonstrar essa capacidade de generalização, realizou-se um teste com parâmetros arbitrários, simulando uma bateria com características distintas:

Tabela 6: Valores de Entrada Arbitrários (Fora do Dataset) - Teste 4

| cycle | chI | chV | chT | disI | disV | disT | BCt |
|---|---|---|---|---|---|---|---|
| 73 | 1121503572 | 3943978905 | 2065867730 | 1757137308 | 3793587913 | 3098216591 | 1428592068 |

```
=== TESTE MANUAL (O modelo nao aprende com estes dados) ===
Insira os parametros:
Cycle: 73
chI: 1121503573
chV: 3943978905
chT: 2065867730
disI: 1757137308
disV: 3793587913
disT: 3098216591
BCt: 1428592068

>>> SOH ESTIMADO: 7.1417743E9
Deseja testar outra bateria? (1=Sim, 0=Nao):
```

Figura 4: Tela de execução demonstrando a predição com valores de entrada inéditos.

O valor estimado pelo modelo foi 7,1417743. Isso comprova a capacidade do algoritmo de generalizar o aprendizado para novas amostras.

# 4 Planejamento e cronograma de execução

| # | Atividade | Descrição / Subtarefas | Responsáveis | Período |
|---|---|---|---|---|
| | | **PARTE PRÁTICA** | | |
| 1 | Entendimento da regressão linear | Compreender a matemática e a lógica do método. | **João** | Outubro |
| 2 | Implementação em Python | Elaborar código manual em Python para facilitar a tradução. | **João** | Novembro |
| 3 | Escolha do banco de dados | Selecionar um conjunto de dados com embasamento técnico. | **Felipe e Cahú** | Novembro |
| 4 | Estudo e implementação em Assembly | Estudar manipulação de arquivos e traduzir o código Python. | **Felipe, Cahú e Lucas** | Novembro e Dezembro |
| 5 | Testes | Verificar se o código Assembly funciona conforme esperado. | **Todos** | Dezembro |
| | | **PARTE DE ESCRITA** | | |
| 1 | Separação em tópicos | Organizar o conteúdo em seções e tópicos. | **Lucas e João** | Novembro |
| 2 | Documentação | Registrar todas as etapas e analisar o andamento. | **Lucas e João** | Novembro |
| 3 | Escrita final do projeto | Unir entendimento técnico e requisitos da professora. | **Lucas e João** | Dezembro |

Tabela 7: Cronograma de execução das atividades.

# 5 Conclusão

O presente trabalho apresentou uma solução para a estimativa do *State of Health* (SoH) de baterias de veículos elétricos e híbridos. O sistema se mostrou capaz de processar diferentes tipos de dados intrínsecos às baterias para inferir a degradação da bateria. A forma em que foi implementado suporta apenas a análise de dados históricos, oferecendo uma ferramenta robusta e assertiva para o diagnóstico da saúde energética do veículo obtida através da execução de consultas manuais.

Apesar do resultado satisfatório, o sistema possui algumas limitações, como o monitoramento em tempo real. Atualmente, ainda não é possível obter informações enquanto o veículo é utilizado, pois o sistema não foi integrado a algum veículo para a realização da predição de maneira automática, restringindo a ferramenta ao uso manual para consulta da estimativa.

Posteriormente, o sistema poderia ser otimizado substituindo o modelo de regressão linear por um de regressão polinomial, tendo como foco uma maior precisão nas predições. Além disso, o projeto pode ser ampliado e ser integrado diretamente ao veículo com a possibilidade de observar a saúde e os parâmetros operacionais utilizados de maneira visual em tempo real através de um *display* dedicado.

# Referências


[1] M. Negri and G. Bieker, "Life-cycle greenhouse gas emissions from passenger cars in the european union: A 2025 update and key factors to consider," Technical Report ID-392, The International Council on Clean Transportation (ICCT), July 2025.

[2] A. Albatayneh, M. N. Assaf, D. Alterman, and M. Jaradat, "Comparison of the overall energy efficiency for internal combustion engine vehicles and electric vehicles," *Environmental and Climate Technologies*, vol. 24, pp. 669–680, Oct 2020. Open Access.

[3] G1, "Isenção de ipva: saiba quais estados liberam o imposto de carros híbridos e elétricos." `https://g1.globo.com/carros/noticia/2025/02/08/isencao-de-ipva-saiba-quais-estados-liberam-o-imposto-de-carros-hibridos-e-eletricos.ghtml`, feb 2025. Acesso em: 11 dez. 2025.

[4] S. H. Farjana, N. Huda, and M. P. Mahmud, "Life cycle assessment of cobalt extraction process," *Journal of Sustainable Mining*, vol. 18, no. 3, pp. 150–161, 2019.

[5] S. P. Holland, E. T. Mansur, N. Z. Muller, and A. J. Yates, "Are there environmental benefits from driving electric vehicles? the importance of local factors," *American Economic Review*, vol. 106, no. 12, pp. 3700–3729, 2016.

[6] S. Ashcroft, "Evs pose growing fire risk to supply chain, says allianz." `https://supplychaindigital.com/supply-chain-risk-management/evs-pose-growing-fire-risk-to-supply-chain-says-allianz`, July 05 2023. Acesso em: DD Mês AAAA.

[7] M. Slattery, J. Dunn, and A. Kendall, "Transportation of electric vehicle lithium-ion batteries at end-of-life: A literature review," *Resources, Conservation and Recycling*, vol. 174, p. 105755, 2021.

[8] M. J. Daigle and C. S. Kulkarni, "End-of-discharge and end-of-life prediction in lithium-ion batteries with electrochemistry-based aging models," tech. rep., NASA Ames Research Center, 2016. AIAA Infotech@Aerospace Conference paper / NASA Technical Report.

[9] M.-S. R. Ranga, V. R. Aduru, N. V. Krishna, K. D. Rao, S. Dawn, F. Alsaif, S. Alsulamy, and T. S. Ustun, "An unscented kalman filter-based robust state of health prediction technique for lithium ionyour query "ashcroft"is ambiguous. do you mean: - ashcroft and mermin (the solid state physics textbook)? - ashcroft, a place (e.g., ashcroft, british columbia)? - a person with the surname ashcroft (e.g., john ashcroft,

michael ashcroft)? - the company ashcroft (pressure and temperature instruments)? specify which one you're interested in and what you'd like to know (e.g., summary, biography, review, etc.). batteries," *Batteries*, vol. 9, no. 7, p. 376, 2023.

[10] S. E. O'Kane, W. Ai, G. Madabattula, D. Alonso-Alvarez, R. Timms, V. Sulzer, J. S. Edge, B. Wu, G. J. Offer, and M. Marinescu, "Lithium-ion battery degradation: how to model it," *Physical Chemistry Chemical Physics*, vol. 24, no. 13, pp. 7909–7922, 2022.

[11] B. Xu, A. Oudalov, A. Ulbig, G. Andersson, and D. S. Kirschen, "Modeling of lithium-ion battery degradation for cell life assessment," *IEEE transactions on smart grid*, vol. 9, no. 2, pp. 1131–1140, 2016.

[12] K. Uddin, S. Perera, W. D. Widanage, L. Somerville, and J. Marco, "Characterising lithium-ion battery degradation through the identification and tracking of electrochemical battery model parameters," *Batteries*, vol. 2, no. 2, p. 13, 2016.

[13] A. Ng, "Cs229 lecture notes: Machine learning." `https://cs229.stanford.edu/lectures-spring2022/main_notes.pdf`, 2022. Accessed: 2025-10-23.

[14] S. H. Haji and A. M. Abdulazeez, "Comparison of optimization techniques based on gradient descent algorithm: A review," *PalArch's Journal of Archaeology of Egypt/Egyptology*, vol. 18, no. 4, pp. 2715–2743, 2021.

[15] programmer3, "Lithium-Ion Battery Degradation Dataset." `https://www.kaggle.com/datasets/programmer3/lithium-ion-battery-degradation-dataset`, 2023. Accessed: 2025-10-23.

[16] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.