Explanatory Document for `tms_1.m`: Battery Cooling Simulation

Matriculation Number : 5014405

May 14, 2025

Contents

1 Introduction

The `tms_1.m` script is a MATLAB program designed to simulate the thermal behavior of a 48V, 4kWh lithium-ion battery pack (13s10p configuration) under a constant 2C discharge rate. It models a liquid cooling system using a cooling plate, aiming to maintain cell temperatures near 35°C. The script supports both Simulink (if licensed) and a numerical ODE solver (Euler method) for thermal dynamics, avoiding Simscape dependencies. This document explains each section of the code in detail, covering its purpose, logic, and implementation.

2 Code Structure

The script is organized into the following sections:

- License and Environment Check
- Battery Pack Parameters
- Cooling System Parameters
- Simulation (Simulink or Numerical)
- Results
- Plotting
- Troubleshooting Notes

3 License and Environment Check

```matlab
clear; clc; close all;

matlab_ver = ver;
simscape_licensed = license('test', 'Simscape');
simulink_licensed = license('test', 'Simulink');
fprintf('MATLAB Version: %s\n', matlab_ver(1).Version);
fprintf('Simscape Licensed: %s\n', mat2str(simscape_licensed));
fprintf('Simulink Licensed: %s\n', mat2str(simulink_licensed));

use_simulink = simulink_licensed;
if use_simulink
    try
        add_block('simulink/Commonly Used Blocks/Integrator',
            'test/Integrator');
        delete_block('test/Integrator');
        bdclose('test');
        fprintf('Simulink Integrator block available.\n');
    catch e
        warning('Simulink Integrator block unavailable: %s. Using
            numerical solver.', e.message);
        use_simulink = false;
    end
else
    fprintf('Simulink not licensed. Using numerical ODE solver.\n');
end
```

This section initializes the environment by clearing variables, command window, and figures. It checks the MATLAB version and licenses for Simscape and Simulink using the `ver` and `license` functions. The script then tests if Simulink is available and can use the Integrator block by attempting to add and delete a test block. If successful, `use_simulink` is set to `true`; otherwise, it falls back to a numerical solver (Euler method), ensuring compatibility with systems lacking Simulink.

4 Battery Pack Parameters

```
cell_voltage = 3.6; % Nominal voltage (V)
cell_capacity = 8.55; % Capacity (Ah, adjusted for ~4kWh)
cell_resistance = 0.02; % Internal resistance (Ohm)
cell_mass = 0.07; % Mass per cell (kg)
cell_specific_heat = 900; % Specific heat (J/kg-K)

num_series = 13;
num_parallel = 10;
pack_voltage = num_series * cell_voltage; % 46.8V
pack_capacity = num_parallel * cell_capacity; % 85.5Ah
pack_energy = pack_voltage * pack_capacity / 1000; % ~4kWh

discharge_rate = 2; % 2C
current_total = discharge_rate * pack_capacity; % 171A
current_cell = current_total / num_parallel; % 17.1A per cell

heat_gen_cell = (current_cell^2) * cell_resistance + 0.5; % ~5.84W + 0.5W entropic
total_heat_base = heat_gen_cell * num_series * num_parallel; % ~799.2W

T_ambient_base = 35; %  C  (nominal ambient)
T_max_limit = 45; %  C
T_coolant_in = 30; %  C  (realistic for cooling)
T_initial = 25; %  C  (room temperature)
```

This section defines the battery pack's electrical and thermal properties for a 21700 lithium-ion cell. Key parameters include: - **Cell Specs**: 3.6V nominal voltage, 8.55Ah capacity, 20 mOhm resistance, 70g mass, and 900 J/kg-K specific heat. - **Pack Configuration**: 13 cells in series (13s) and 10 in parallel (10p), yielding 46.8V and 85.5Ah, approximately 4kWh. - **Discharge**: A constant 2C rate (171A total, 17.1A per cell). - **Heat Generation**: Calculated as $I^2R + 0.5$ W entropic heating per cell, totaling 799.2W for the pack. - **Ambient Conditions**: Nominal ambient at 35°C, coolant inlet at 30°C, initial cell temperature at 25°C, and a 45°C maximum limit.

These parameters establish the baseline for thermal simulations, assuming a constant load and simplified resistance model.

5 Cooling System Parameters

```
pipe_surface_area = 0.2; % m^2 (multi-pass plate)
h_conv_base = 800; % Base convective coefficient (W/m^2-K)
k_control = 75; % Proportional control gain (W/m^2-K/ C , tuned for practical response)
tau_delay = 30; % Coolant delay time constant (s)

mass_factor = 1.2; % 20% increase for casing and coolant mass
ambient_noise_std = 0.5; % Ambient temperature noise ( 0 .5 C )
heat_variation_amplitude = 0.05; %  5 % variation in heat generation
```

```
heat_variation_period = 600; % Variation period (s)
```

The cooling system uses a liquid cooling plate with: - **Surface Area**: 0.2 m^2 for heat transfer. - **Convective Coefficient**: 800 W/m^2-K, typical for liquid cooling. - **Proportional Control**: A gain of 75 W/m^2-K/°C to adjust cooling based on temperature error. - **Coolant Delay**: 30s time constant for realistic response lag. - **Practical Effects**: 20

These parameters model a practical cooling system with dynamic adjustments.

6 Simulation

```
mCp = mass_factor * cell_mass * num_series * num_parallel *
   cell_specific_heat; % J/K
T_max = NaN; T_final = NaN;
temp_profile = []; time_data = [];

if use_simulink
    % Create Simulink model
    model_name = 'BatteryCoolingModel';
    bdclose(model_name);
    new_system(model_name);
    ...
    % Simulation settings
    set_param(model_name, ...
        'StopTime', '3600', ...
        'Solver', 'ode23t', ...
        'RelTol', '1e-4', ...
        'AbsTol', '1e-6');
    try
        sim(model_name);
        temp_var = evalin('base', 'TempData');
        T_max = max(temp_var.signals.values);
        T_final = temp_var.signals.values(end);
        temp_profile = temp_var.signals.values;
        time_data = temp_var.time;
    catch e
        fprintf('Simulation failed: %s\n', e.message);
        ...
    end
    bdclose(model_name);
else
    t = 0:0.01:3600; dt = 0.01;
    T = zeros(size(t)); T(1) = T_initial;
    Q_cool_delayed = 0; dt = 0.01;
    for j = 2:length(t)
        total_heat = total_heat_base * (1 + heat_variation_amplitude *
           sin(2 * pi * t(j-1) / heat_variation_period));
        T_ambient = T_ambient_base + ambient_noise_std * randn();
        delta_T = T(j-1) - T_coolant_in;
        h_eff = h_conv_base / (1 + 0.01 * delta_T);
        h_control = 0;
        if T(j-1) > T_ambient
            h_control = k_control * (T(j-1) - T_ambient);
        end
        h_total = h_eff + h_control;
        Q_cool = h_total * pipe_surface_area * (T(j-1) - T_coolant_in);
```

```
        Q_cool_delayed = Q_cool_delayed + (Q_cool - Q_cool_delayed) * (dt / tau_delay);
        dT_dt = (total_heat - Q_cool_delayed) / mCp;
        T(j) = T(j-1) + dT_dt * dt;
    end
    T_max = max(T); T_final = T(end);
    temp_profile = T; time_data = t;
end
```

This section simulates the battery's thermal behavior over 3600s (1 hour). The thermal mass mCp accounts for the pack and casing. The simulation branches based on `use_simulink`:

- **Simulink Path**: Creates a model with blocks for heat generation (constant + sinusoidal variation), ambient temperature (constant + noise), proportional cooling, coolant delay, and thermal dynamics. It uses the `ode23t` solver for stiff systems, with tight tolerances (10^{-4}, 10^{-6}). Results are extracted from a scope. - **Numerical Path**: Uses a simple Euler method with a 0.01s step. For each step: - Heat varies sinusoidally (±5%). - Ambient temperature includes Gaussian noise (±0.5°C). - Cooling adjusts nonlinearly ($h_{\text{eff}} = h_{\text{base}}/(1+0.01\Delta T)$) and proportionally when $T > T_{\text{ambient}}$. - Coolant delay is modeled as a first-order lag. - Temperature updates via $\frac{dT}{dt} = \frac{Q_{\text{heat}} - Q_{\text{cool}}}{mCp}$.

The numerical approach is simpler but less robust than Simulink's solver.

7 Results

```
fprintf('Cell Temperature (Max): %.2f  C \n', T_max);
fprintf('Cell Temperature (Final): %.2f  C \n', T_final);
fprintf('Coolant Temperature: %.2f  C \n', T_coolant_in);
fprintf('Ambient Temperature (Nominal): %.2f  C \n', T_ambient_base);
if T_max <= T_max_limit && ~isnan(T_max)
    fprintf('Cell temperature is within limit (%.0f C).\n', T_max_limit);
    if abs(T_final - T_ambient_base) <= 1
        fprintf('Cell temperature stabilized within ambient range (%.0f    1 C ).\n', T_ambient_base);
    else
        fprintf('Cell temperature stabilized outside ambient range (%.0f    1 C ) but within limit.\n', T_ambient_base);
    end
else
    warning('Cell temperature exceeds limit (%.0f C) or simulation failed.', T_max_limit);
end
```

This section prints the maximum and final cell temperatures, coolant inlet temperature, and nominal ambient temperature. It checks if $T_{\text{max}} \leq 45$°C and if T_{final} is within 35°C ± 1°C, providing feedback on thermal stability and safety.

8 Plotting

```
figure('Name', 'Temperature Profile');
if ~isempty(temp_profile)
    plot(time_data, temp_profile, 'b-', 'LineWidth', 2, 'DisplayName', 'Cell Temp');
    hold on;
```

```
end
plot([0 3600], [T_coolant_in T_coolant_in], 'c--', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('Coolant Temp (%.0f C)', T_coolant_in));
plot([0 3600], [T_ambient_base T_ambient_base], 'm--', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('Ambient Temp (%.0f C)', T_ambient_base));
plot([0 3600], [45 45], 'r--', 'LineWidth', 1.5, 'DisplayName', '45 C Limit');
xlabel('Time (s)'); ylabel('Temperature ( C )');
title('Cell Temperature Over Time (Practical Model)');
grid on; legend('Location', 'best');
```

A figure plots the cell temperature profile (blue line) against time, with reference lines for coolant temperature (cyan dashed), ambient temperature (magenta dashed), and the 45°C limit (red dashed). The grid and legend enhance readability.

9 Troubleshooting Notes

```
% If cell temperature exceeds 45 C or doesn't stabilize near 35 C :
% 1. Increase h_conv_base (e.g., 900 W/m^2-K) or pipe_surface_area (e.g., 0.25 m^2).
% 2. Adjust k_control (e.g., 100 150 W/m^2-K/ C for stronger control).
% 3. Reduce tau_delay (e.g., 15s) for faster response.
% 4. Set T_coolant_in closer to 35 C (e.g., 32 C ).
% 5. Verify heat generation (total_heat_base 799.2W).
% General troubleshooting:
% 6. Run 'ver' to list toolboxes.
% 7. Check Simulink: 'license('test', 'Simulink')'.
% 8. List Simulink blocks: 'find_system('simulink')'.
% 9. For Simscape issues, contact university IT or MathWorks...
```

These comments provide guidance for addressing high temperatures or simulation failures, suggesting adjustments to cooling parameters and diagnostic commands to check MATLAB's configuration.

10 Summary

The `tms_1.m` script provides a foundational model for battery thermal simulation with a constant 2C discharge. It supports Simulink or numerical methods, includes realistic cooling dynamics (proportional control, coolant delay, noise), and visualizes results. Limitations include a single discharge rate and lack of a GUI, which are addressed in later versions.