Assignment #2: ECMS

Table of Contents

Group information

Group number: 4

Students:

- Giacomo Chiavetta, 348027
- Massimo Stella, s348062
- Tommaso Massone, s349235

Introduction

In this second project, it is required to develop an improved control strategy compared to the less sophisticated rule-based controller built in the first project.

Starting from a different homologation cycle, called the Artemis Road Driving Cycle (ARDC), several simulations are performed using the same car architecture employed in the previous report, namely a series hybrid electric vehicle (HEV).

Both the cycle and the vehicle specifications are provided in the "data" folder.

Additionally, as in the previous project, some functions used to model the battery, the longitudinal dynamics, and the performance of the devices are available in the "models" folder. The vehicle layout includes an Internal Combustion Engine (ICE) that drives a Generator, a Battery, and a Traction Motor, which is supplied by both the Battery and the Generator.

The variables within the vehicle system can be classified as:

Exogenous Inputs:

- Vehicle Speed - *VehSpd*
- Vehicle Acceleration - *VehAcc*

State Variables:

- State of Charge - *SOC*

Control Variables:

- Engine Speed - *EngSpd*
- Engine Torque - *EngTrq*

Stage Cost:

- Fuel Flow Rate - *fuelFlwRate*

The improved control strategy that we are now exploiting is called ECMS (Equivalent Consumption Minimization Strategy). It is a local optimization strategy applied over the considered cycle that results in a sub-optimal optimization despite the advance knowledge of the entire cycle. This inaccuracy is due to the nature of control strategy which is made instant by instant without exploiting the whole knowledge of the cycle.

Nevertheless, the ECMS control trajectory provides a solution that is very close to the optimal one, although properly tuning of the base parameter requires some effort to adapt it to the considered cycle or the actual driving conditions. It is relevant to notice that the cost function is the fuel consumed only since the battery state of charge at the end of the cycle must coincide with the one at the beginning to achieve battery sustaining mode.

The controller working principle consists of a point-by-point minimization of the equivalent fuel consumption, which combines the actual engine fuel consumption and a virtual contribution term that takes into account the future usage of the electric part, supplied back from the battery, or its cost when the generator will have to recharge the battery.

This virtual contribution contains the corresponding battery electrical power converted into fuel grams per second, that in case of pure electric (PE) or charge depleting (CD) modes is positive and assumes the meaning of the fuel that will be required to produce the energy just consumed to power the traction motor. In case of battery charging mode (BC), instead, this contribution is negative and assumes the meaning of the fuel that will be saved in the future thanks to the electric part after having generated it.

It can be understood that the knowledge of the future operating points conditions is fundamental to perform correctly the energy conversion forecasts. More specifically, the knowledge of the efficiencies of the entire power chain must be approximated through a single parameter called equivalence factor, which assumes the value of the reciprocal of the mean value of the total efficiencies of the entire model over the cycle considering also the battery charging during regenerative braking. This parameter is very sensitive to the different driving conditions and, as will be demonstrated, it strongly affects the effectiveness of this control strategy.

Load the cycle and vehicle data

Project starts with the provided folders acquisition by means of the following commands:

```
clear;clc;close all

addpath("models");      % Add the "models" directory to the search path
addpath("data");        % Add the "data" directory to the search path
addpath("utilities");   % Add the "utilities" directory to the search path
```

Load cycle data

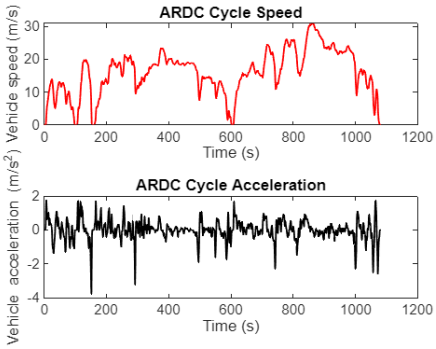
The project considers the ARDC (Artemis Road Driving Cycle), whose corresponding data are stored within the following directory.

```
mission = load("data\ARDC.mat"); % ECMS data are now contained in the struct
mission variable
time = mission.time_s; % (s)  Vector of the time instants within the considered
cycle
vehSpd = mission.speed_km_h ./ 3.6; % (m/s) Vector of the instant velocities
referred to each time value
vehAcc = mission.acceleration_m_s2; % (m/s^2) Vector of the instant acceleration
referred to each time value
```

Plot the mission cycle data

The velocity and acceleration profiles of the ARDC cycle are now being plotted as a function of time.

```
t = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

nexttile(1) % Select the first tile (upper plot)
plot(time, vehSpd,"r-",'LineWidth', 1); % Plot vehicle speed over time
title("ARDC Cycle Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time, vehAcc,"k-", 'LineWidth', 1); % Plot vehicle acceleration over time
title("ARDC Cycle Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2)");
```



Load the vehicle data

The vehicle parameters have been retrieved and the relevant data pertaining to our workgroup have been substituted through the provided function "scaleVehData".

```
veh = load("data\vehData.mat"); % Vehicle data are now contained in the struct veh
variable

% Scale vehicle data according to group parameters

veh = scaleVehData(veh, 66e3, 82e3, 945); % Rated motor power, Rated engine power,
Battery nominal energy, respectively
```

Simulation Loop

```
% Initialize the state of charge
SOC1 = zeros(length(time),1); % Create a vector to store SOC values over time
SOC1(1) = 0.6; % Initial SOC value (60%)
eqFactor = 2.5; % First equivalance factor trial

fuelFlwRate1 = zeros(length(time),1); % Create a vector to store Fuel Flow Rate
values over time

for k = 1: length(time)
    if k < length(time)
        [engSpd, engTrq] = ecmsControl(SOC1(k), vehSpd(k), vehAcc(k), eqFactor,
veh);
        [SOC1(k+1), fuelFlwRate1(k), unfeas(k), prof1(k)] = hev_model(SOC1(k),
[engSpd, engTrq], [vehSpd(k), vehAcc(k)], veh);
    else
        [engSpd, engTrq] = ecmsControl(SOC1(k), vehSpd(k), vehAcc(k), eqFactor,
veh);
        [SOC1(k), fuelFlwRate1(k), unfeas(k), prof1(k)] = hev_model(SOC1(k),
[engSpd, engTrq], [vehSpd(k), vehAcc(k)], veh);
    end
end
```

Equivalence factor calibration

Introduction

This script implements a bisection algorithm to calibrate the equivalence factor used in the ECMS (Equivalent Consumption Minimization Strategy) controller. The goal is to ensure a charge-sustaining operation, where the final State of Charge (SOC) matches the initial value within a tolerance of ±0.01. Key variables are initialized at the beginning: *SOC*(1) is set to 0.6 (representing 60% initial charge), *eqFactor_low* and *eqFactor_high* define the initial search interval, and *phi_low* and *phi_high* store the terminal SOC deviations at each bound. The simulation uses the functions *ecmsControl* and *hev_model* to compute engine controls and SOC evolution over time, and iteratively narrows the interval until convergence is reached, this final value is achieved thanks to, indeed, the calibration on which the equivalnet factor is based.

The variables within the script can be classified as follows:

State and Control Variables:

- *SOC* – Vector storing the battery State of Charge over time.
- *fuelFlwRate* – Vector storing the fuel flow rate at each time step.
- *engSpd* – Engine speed selected by the ECMS controller at a given time step.
- *engTrq* – Engine torque selected by the ECMS controller at a given time step.
- *unfeas* – Logical flag indicating whether a control action is infeasible.

Driving Cycle and Vehicle Inputs:

- *time* – Vector containing the time points of the driving cycle.
- *vehSpd* – Vehicle speed profile (m/s) over time.
- *vehAcc* – Vehicle acceleration profile (m/s²) over time.
- *veh* – Struct containing vehicle parameters (engine, motor, battery data, etc.).

Equivalence Factor Parameters:

- *eqFactor_low* – Lower bound of the equivalence factor interval.
- *eqFactor_high* – Upper bound of the equivalence factor interval.
- *eqFactor_average* – Midpoint value of the current equivalence factor interval.
- *eqFactor_cal* – Final calibrated equivalence factor found after convergence.

SOC Deviation Metrics:

- *phi_low* – SOC deviation after simulation with `eqFactor_low`.
- *phi_high* – SOC deviation after simulation with `eqFactor_high`.
- *phi_average* – SOC deviation after simulation with `eqFactor_average`.

Indexing and Control:

- *k* – Loop index used to iterate over time steps.
- Loop condition: *while abs(phi_average(end)) > 0.01* – Checks convergence of the bisection algorithm.

Code description

This MATLAB script performs a numerical calibration of the equivalence factor used in the ECMS controller by applying a bisection algorithm. The aims of ECMS are to minimize an equivalent fuel consumption by combining the actual fuel usage and the electrical energy drawn from the battery, scaled through the equivalence factor. Since the control policy is highly sensitive to the value of this factor, a proper calibration is crucial to ensure a charge-sustaining operation, meaning the final battery State of Charge (SOC) should match the initial SOC within a small tolerance (±0.01).

The algorithm starts with a simulation using two initial guesses for the equivalence factor - `eqFactor_low` and `eqFactor_high` -that are assumed to bracket the solution, checks the sign of the SOC deviation at both ends,

in the case that the value of *phi_high* gives a negative deviation, the initial interval does not bracket a solution, and the algorithm is invalidand, stopped to prevent incorrect results. Assuming a valid interval, the bisection procedure proceeds: the midpoint between the two bounds is calculated, and another full simulation is executed with this *eqFactor_average*. The deviation is computed again, and based on its sign, either the upper or lower bound is updated, effectively halving the search interval. This process is repeated iteratively in the `while` loop until the absolute deviation becomes smaller than the specified tolerance. Once convergence is achieved, the resulting *eqFactor_cal* is considered optimal for charge-sustaining performance over the given driving cycle.

The final calibrated value of *eqFactor_cal* can be used in further simulations or saved alongside other performance metrics such as fuel consumption and fuel economy.

```
% Initialize the State of Charge
SOC = zeros(length(time),1); % Create a vector to store SOC values over time
SOC(1) = 0.6; % Initial SOC value (60%)

fuelFlwRate = zeros(length(time),1); % Create a vector to store Fuel Flow Rate
values over time

eqFactor_low = 2.5; % Defines the initial lower limit of the search interval for
the equivalence factor

phi_low = SOC(length(time)) - SOC(1); % Calculates the SOC deviation at the lower
bound; though this is placed before SOC is updated, it likely should be computed
after the corresponding simulation

SOC(1) = 0.6; % Sets the initial state of charge to 60% for the simulation

eqFactor_high = 3; % Sets the upper limit of the equivalence factor range for the
first simulation

% This loop simulates the full drive cycle using the eqFactor_high value to
determine how the SOC evolves over time with this factor
for k = 1: length(time)
    if k < length(time)
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k), eqFactor_high,
veh);
        [SOC(k+1), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd, engTrq],
[vehSpd(k), vehAcc(k)], veh);
    else
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k), eqFactor_high,
veh);
        [SOC(k), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd, engTrq],
[vehSpd(k), vehAcc(k)], veh);
    end
end

phi_high = SOC(length(time)) - SOC(1); % Calculates the SOC deviation after the
simulation using eqFactor_high
```

```
% Checks if the upper bound gives a negative deviation
if phi_high < 0
    error
end

% Core step of the bisection method;
% computes the midpoint between the current upper and lower bounds another full
simulation is performed, this time with the midpoint value of the equivalence
factor
eqFactor_average = eqFactor_low + ((eqFactor_high - eqFactor_low)/2);
for k = 1: length(time)
    if k < length(time)
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k),
eqFactor_average, veh);
        [SOC(k+1), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd, engTrq],
[vehSpd(k), vehAcc(k)], veh);
    else
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k),
eqFactor_average, veh);
        [SOC(k), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd, engTrq],
[vehSpd(k), vehAcc(k)], veh);
    end
end

phi_average = SOC(length(time)) - SOC(1); % SOC deviation after simulating with the
averaged equivalence factor

% Core logic of narrowing the bisection interval;
% updates the bounds of the interval based on the sign of the deviation
if phi_average > 0
    phi_high = phi_average;
    eqFactor_high = eqFactor_average;

elseif phi_average < 0
    phi_low = phi_average;
    eqFactor_low = eqFactor_average;

end
% Main loop: continues to narrow down the search interval until the deviation is
within tolerance
while abs(phi_average(end)) > 0.01

    eqFactor_average = eqFactor_low + ((eqFactor_high - eqFactor_low)/2); % Re-
calculates the midpoint at each iteration for the next test
    % Simulates again using the newly calculated midpoint of the interval
    for k = 1: length(time)
        if k < length(time)
            [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k),
eqFactor_average, veh);
```

```
            [SOC(k+1), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd,
engTrq], [vehSpd(k), vehAcc(k)], veh);
        else
            [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k),
eqFactor_average, veh);
            [SOC(k), fuelFlwRate(k), unfeas(k)] = hev_model(SOC(k), [engSpd,
engTrq], [vehSpd(k), vehAcc(k)], veh);
        end
    end

    phi_average = SOC(length(time)) - SOC(1); % SOC deviation check after each new
simulation
    % Adjusts the interval endpoints based on the latest simulation results,
ensuring the root remains bracketed
    if phi_average > 0
        phi_high = phi_average;
        eqFactor_high = eqFactor_average;
        eqFactor_cal = eqFactor_average;

    elseif phi_average < 0
        phi_low = phi_average;
        eqFactor_low = eqFactor_average;
        eqFactor_cal = eqFactor_average;
    end
end
```

Run a simulation with the calibrated equivalence factor

Run a simulation with the calibrated equivalence factor, so you can inspect the behavior of the charge-sustaining ECMS controller.

```
for k = 1: length(time)
    if k < length(time)
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k), eqFactor_cal,
veh);
        [SOC2(k+1), fuelFlwRate2(k), unfeas(k), prof2(k)] = hev_model(SOC(k),
[engSpd, engTrq], [vehSpd(k), vehAcc(k)], veh);
    else
        [engSpd, engTrq] = ecmsControl(SOC(k), vehSpd(k), vehAcc(k), eqFactor_cal,
veh);
        [SOC2(k), fuelFlwRate2(k), unfeas(k), prof2(k)] = hev_model(SOC(k),
[engSpd, engTrq], [vehSpd(k), vehAcc(k)], veh);
    end
end
```

Results and Post-Processing

State of Charge (SOC) evolution

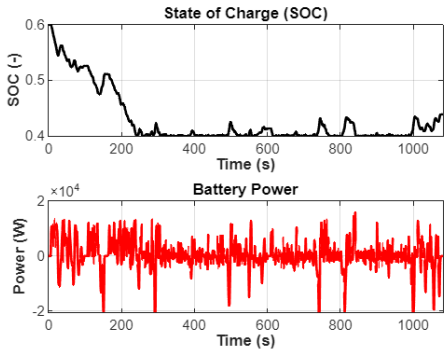
- **Without equivalence factor calibration**

The plot shows the SOC trend over time when the equivalence factor (s) is not properly balanced. The SOC rapidly decreases during the first 300 seconds, reaching the lower limit of 0.4. Subsequently, the SOC oscillates around the minimum boundary, indicating that the energy management strategy fails to maintain charge balance without an appropriate adjustment of the equivalence factor. This behavior highlights the necessity of tuning the equivalence factor, typically achieved through the method of the bisection algorithm, to ensure optimal energy distribution and SOC stabilization.

As we can see the evolution of the State of Charge (SOC) is closely linked to the vehicle's speed and acceleration profiles throughout the driving cycle. During acceleration phases, the power demand from the electric motor increases significantly, leading to a more rapid depletion of the SOC. In contrast, when the vehicle maintains a steady-state speed, the SOC tends to decrease more gradually, influenced by the load conditions and the overall efficiency of the propulsion system. Moreover, during deceleration or braking phases, especially when regenerative braking is active, a partial recovery of the SOC can be observed.

Analyzing the ARDC cycle profiles, it becomes evident that the frequent and intense acceleration events contribute to the rapid decrease in SOC, particularly noticeable in the case where the initial value is unbalanced. Conversely, during periods characterized by speed plateaus or low acceleration, the SOC variation appears less pronounced, occasionally stabilizing.

```
%State of Charge
figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(time, SOC1, "k-", 'LineWidth', 1.5)
grid on
title("State of Charge (SOC)", 'FontWeight', 'bold');
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time(1) time(end)])

%Battery Power
ax2 = nexttile; % Select the second tile (lower plot)
battPower = [prof1.battVolt] .* [prof1.battCurr];
plot(time, battPower, 'r-', 'LineWidth', 1.5)
grid on
title("Battery Power", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Power (W)", 'FontWeight', 'bold')
xlim([time(1) time(end)])

linkaxes([ax1, ax2], 'x') % Synchronizes zoom on the x-axis between ax1 and ax2
```



- **With equivalence factor calibration**

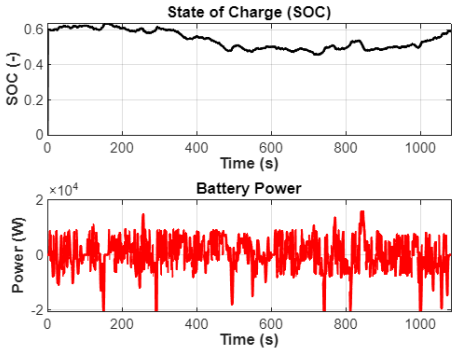
The plot shows the SOC trend after tuning the equivalence factor (s) using the bisection algorithm. Initially, the SOC remains stable around 0.6, indicating a good balance between electric and thermal power contributions. As the simulation progresses, the SOC gradually increases up to 160 seconds due to relatevely low power demands which make more advantageous to recharge the battery during this interval for a future usage.

Subsequenlty, the SOC trend starts to decrease with a more pronounced drop around 350 seconds, reaching a minimum of approximately 0.45 between 600 and 800 seconds. In the final phase a recovery trend appears consistently with the control boundaries which force the state of charge to be restored at the initial value.

Compared to the non calibrated case, the SOC evolution is much smoother and remains consistently within the acceptable limits [0.4, 0.8] thus exploiting the assistance of the electrical part attributing the proper cost to its usage. The final SOC value of 0.59 confirms the effectiveness of the control strategy, staying within the ±1% error window relative to the initial value. This highlights the success of the equivalence factor tuning in ensuring proper charge management and overall energy efficiency.

```
%State of Charge
figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(time, SOC2, "k-", 'LineWidth', 1.5)
grid on
title("State of Charge (SOC)", 'FontWeight', 'bold');
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time(1) time(end)])

%Battery Power
ax2 = nexttile; % Select the second tile (lower plot)
battPower = [prof2.battVolt] .* [prof2.battCurr];
plot(time, battPower, 'r-', 'LineWidth', 1.5)
grid on
title("Battery Power", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Power (W)", 'FontWeight', 'bold')
```

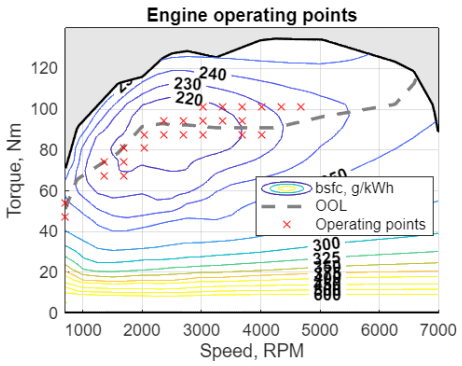
```
xlim([time(1) time(end)])

linkaxes([ax1, ax2], 'x') % Synchronizes zoom on the x-axis between ax1 and ax2
```



Engine bsfc map with operating points

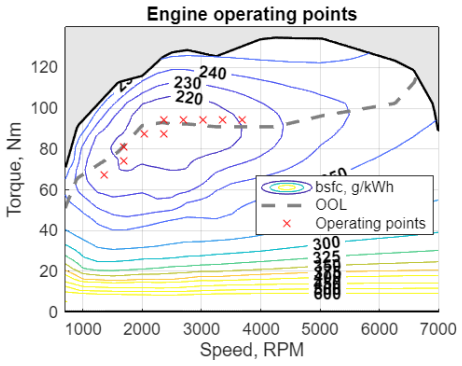
- **<u>Without</u> equivalence factor calibration**

```
engMapWithPF(veh.eng, prof1, 'bsfc');
```



- **<u>With</u> equivalence factor calibration**

```
engMapWithPF(veh.eng, prof2, 'bsfc');
```



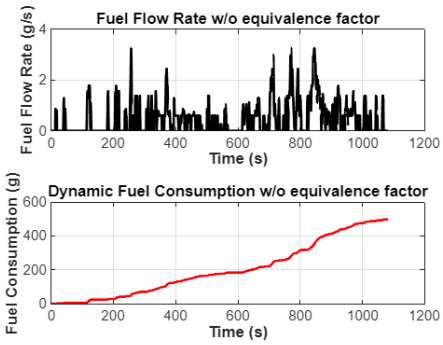
Result extrapolation

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption = trapz(fuelFlwRate)/1e3; % (Kg)

% Convert the fuel consumed from mass to volume quantity
Fuel_l = fuelConsumption/(veh.eng.fuelDensity*1e3); % (l)

% Calculate the cycle total dinstance using numerical integration (trapz)
distance = trapz(vehSpd)/1e3; % (Km)

fuelEconomy = Fuel_l/distance * 100; % (l/100km)

finalSOC = SOC2(end);
```

Fuel Flow Rate and Fuel Consumed

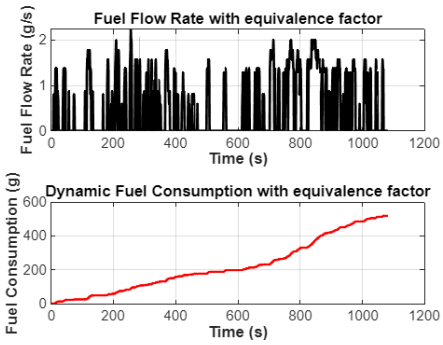
- **Without equivalence factor calibration**

```
% Calculate cumulative fuel consumption using numerical integration (trapz)
%FlwCnsp_dyn = 0; % Initialize the "Dynamic Fuel consumption" variable
FlwCnsp_dyn = zeros(length(time), 1);
for n = 1: length(time)-1
    % Perfom iterative summation of each instant fuel consumption contribution
    FlwCnsp_dyn(n+1) = trapz(fuelFlwRate1(n:n+1)) + FlwCnsp_dyn(n);
end

% Fuel Flow Rate
figure;
ax1 = nexttile;
plot(time, fuelFlwRate1, 'k-', 'LineWidth', 1.5)
grid on
title("Fuel Flow Rate w/o equivalence factor", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Fuel Flow Rate (g/s)", 'FontWeight', 'bold')

% Dynamic Fuel Consumption
ax2 = nexttile;
plot(time, FlwCnsp_dyn, 'r-', 'LineWidth', 1.5)
grid on
title("Dynamic Fuel Consumption w/o equivalence factor", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Fuel Consumption (g)", 'FontWeight', 'bold')

linkaxes([ax1, ax2], 'x')
```



- **<u>With</u> equivalence factor calibration**

```
% Calculate cumulative fuel consumption using numerical integration (trapz)
%FlwCnsp_dyn = 0; % Initialize the "Dynamic Fuel consumption" variable
FlwCnsp_dyn = zeros(length(time), 1);
for n = 1: length(time)-1
    % Perfom iterative summation of each instant fuel consumption contribution
    FlwCnsp_dyn(n+1) = trapz(fuelFlwRate2(n:n+1)) + FlwCnsp_dyn(n);
end

% Fuel Flow Rate
figure;
ax1 = nexttile;
plot(time, fuelFlwRate2, 'k-', 'LineWidth', 1.5)
grid on
title("Fuel Flow Rate with equivalence factor", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Fuel Flow Rate (g/s)", 'FontWeight', 'bold')

% Dynamic Fuel Consumption
ax2 = nexttile;
plot(time, FlwCnsp_dyn, 'r-', 'LineWidth', 1.5)
grid on
title("Dynamic Fuel Consumption with equivalence factor", 'FontWeight', 'bold')
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Fuel Consumption (g)", 'FontWeight', 'bold')
linkaxes([ax1, ax2], 'x')
```




```
```

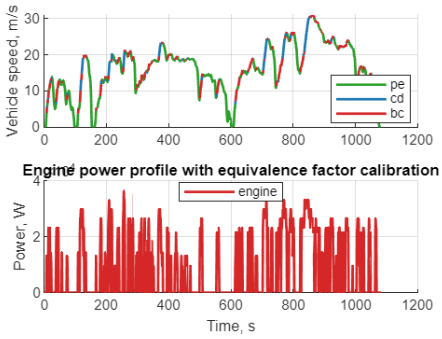
Engine Power Profiles

- **<u>Without</u> equivalence factor calibration**

```
% Plot the speed profile and the engine power over time
powerProfiles(prof1, 'eng');
title("Engine power profile w/o equivalence factor calibration")
```

- **<u>With</u> equivalence factor calibration**

```
% Plot the speed profile and the engine power over time
powerProfiles(prof2, 'eng');
title("Engine power profile with equivalence factor calibration")
```



Motor Power Profiles

The power traces in both plots appear identical and rightly so, this is because they are based on the same driving cycle, the ARDC, and the underlying power demand profile is determined by exogenous inputs, such as vehicle speed and acceleration. These inputs are common to both simulations and do not depend on the control strategy. As a result, the overall shape and intensity of the motor power curve remain unchanged between the two graphs.

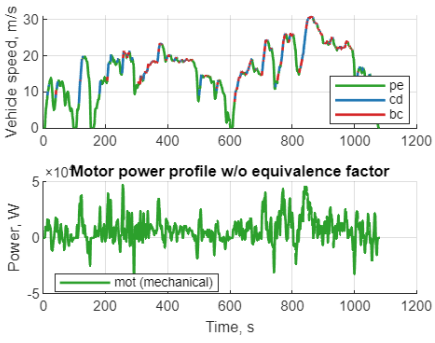
However, when examining the mode of operation of vehicle speed substantial differences become evident. In the first plot, which reflects the scenario without equivalence factor calibration, there is a noticeably higher presence of pe (pure electric) mode. This occurs because the control strategy in this case tends to favor the electric motor more aggressively, even when it may not be energy-optimal in the long run. Alongside this, there is also an increased presence of the cd (charge depleting) mode, which indicates that the battery is being used extensively to provide traction power. The internal combustion engine (ICE) plays a more secondary role here, stepping in only when the battery charge drops too low or during high load demands.

In contrast, the second plot with the equivalence factor properly calibrated shows a calibration in strategy. There is less reliance on pe and cd modes, and a significant increase in bc (battery charging) phases. This reflects a more balanced approach, where the ICE and the electric motor are used in a complementary way. The calibrated equivalence factor allows the controller to maintain energy neutrality more effectively, making it possible to preserve battery charge over the course of the cycle. As a result, the battery can be recharged in a controlled and efficient manner, instead of being continuously drained.

In summary, while the power demand remains constant due to the shared drive cycle, the difference in energy management strategies leads to clearly different operating behaviors. The uncalibrated version prioritizes electric drive at the cost of battery depletion, while the calibrated version adopts a more sustainable and optimized strategy that better integrates both energy sources.

- **Without equivalence factor calibration**

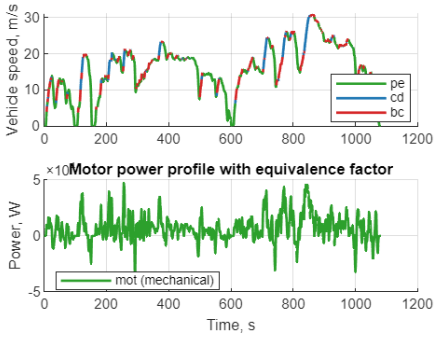
```
% Plot the speed profile and the motor power over time
powerProfiles(prof1, 'mot');
title("Motor power profile w/o equivalence factor")
```



- **With equivalence factor calibration**

```
% Plot the speed profile and the motor power over time
```

```
powerProfiles(prof2, 'mot');
title("Motor power profile with equivalence factor")
```



Traction motor map with operating points

The two graphs illustrate the motor power output over time during the ARDC driving cycle under two different conditions: The first plot, labeled *"*Motor power profile w/o equivalence factor*"*, shows the results of the energy management strategy without applying any calibrated equivalence factor. The second plot, titled *"*Motor power profile with equivalence factor*"*, includes the use of a calibrated equivalence factor obtained through a bisection method aiming to balance the final and initial State of Charge (SOC) of the battery.

In both figures, different drive modes are color-coded: pe (pure electric), bc (battery charging), and cd (charge depleting). These modes illustrate the operating state of the hybrid vehicle at each moment of the cycle. Following the typical peaks and going through of the ARDC speed profile the distribution and dominance of the drive modes show notable differences between the two cases.

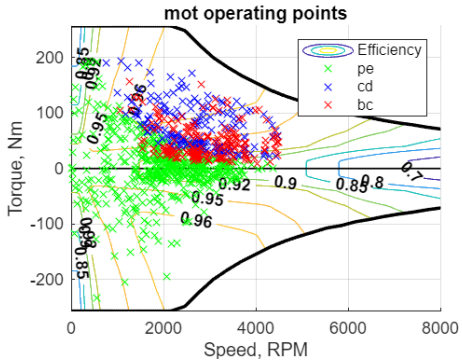
In the first figure, without equivalence factor calibration, the system tends to favor the cd more aggressively. This indicates that the hybrid controller allows deeper discharge of the battery during acceleration phases and relies heavily on electric power. As a result, pe and cd segments are more frequent, and the battery charging moments appear to be slightly scattered and inconsistent. This unbalanced approach can lead to a significant deviation of the SOC from its initial value by the end of the cycle, which would compromise long-term battery health and energy optimization.

On the other hand, the second figure demonstrates a more refined control strategy. Thanks to the calibrated equivalence factor, the distribution of the operating modes is noticeably more balanced. The pe phases are still present obviously, but there is a more regular and strategic use of bc mode, indicating that the controller are able to manages the charging events more effectively to compensate for energy depletion. This results in a smoother SOC trajectory that tends to return to its starting value, maintaining the battery's energy neutrality over the cycle. Additionally, the cd mode is less dominant, suggesting that the vehicle no longer overdependence on electric propulsion, which prevents excessive use on the battery.

Overall, the application of the equivalence factor leads to a more energy-optimal and sustainable use of the hybrid drivetrain. The calibrated strategy not only ensures compliance with energy balance requirements but also introduces a smarter alternation between electric drive and battery charging, improving efficiency and long-term reliability of the system.

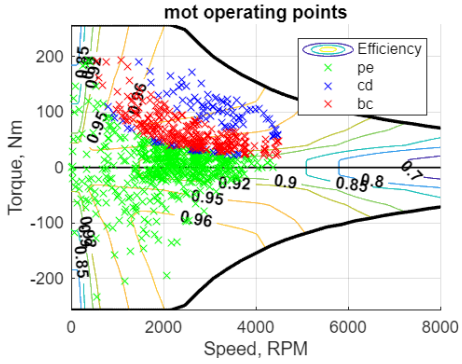
- **Without** equivalence factor calibration

```
% Plot traction motor efficiency map with all exploited working points marked
[~,~] = emMapWithPF(veh.mot, prof1, "mot","all");
```



- **With** equivalence factor calibration

```
% Plot traction motor efficiency map with all exploited working points marked
[~,~] = emMapWithPF(veh.mot, prof2, "mot","all");
```



Generator map with operating points

These two outline plots illustrate the operating points of the generator which is linked with the ICE within the hybrid electric vehicle system during the ARDC cycle, plotted over the generator efficiency map. The x-axis represents generator speed in RPM, and the y-axis shows torque in Newton-meters. Overlap on the efficiency lines are colored markers representing the generator's operating modes: red crosses indicate battery charging (bc), while blue crosses correspond to charge depleting (cd) phases.

Although the two generator efficiency plots may initially appear similar, there are substantial differences between them, which are analyzed in more detail below. These differences are not only visible in the distribution and density of the operating points, but also in the behavior of the internal combustion engine (ICE) and the overall energy management strategy employed in each case. However, when comparing the optimized and non-optimized versions, two key differences become evident. The first is related to the location of the operating points, which are slightly shifted due to the effect of the gear ratio. The second, and more significant, difference concerns the number of operating points present in each plot. In the non-optimized version, we observe a larger

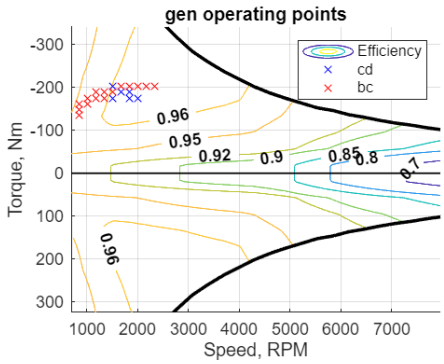
number of generator operating points. This is because once the battery's state of charge drops to its minimum threshold (around 0.4), the internal combustion engine (ICE) is forced to take over most of the energy demand. Since the battery can no longer support the vehicle, the ICE must respond to the full load, which is variable throughout the driving cycle. As a result, it operates across a wider range of points, increasing the spread of generator activity.

On the other hand, the optimized case shows fewer operating points. This is because the ICE is used more strategically and, on average, less intensively. The presence of a calibrated equivalence factor allows for a better balance between fuel and electric energy, meaning that the battery can provide consistent support and reduce the load on the engine. Consequently, the generator operates in a more confined and efficient region.

Despite these differences in usage patterns, the efficiency zones where the generator tends to operate remain largely similar in both cases. The majority of the points lie within high-efficiency areas of the map, although in the optimized version there is a slightly improved tendency for the generator to work at slightly higher efficiency levels. This reflects a more refined energy management strategy and highlights the benefits of optimization in hybrid vehicle control.

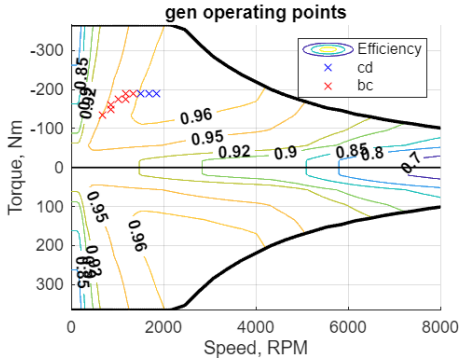
- **Without equivalence factor calibration**

```
% Plot generator efficiency map in which all working points are marked
[~,~] = emMapWithPF(veh.gen, prof1, "gen","all");
```



- **With equivalence factor calibration**

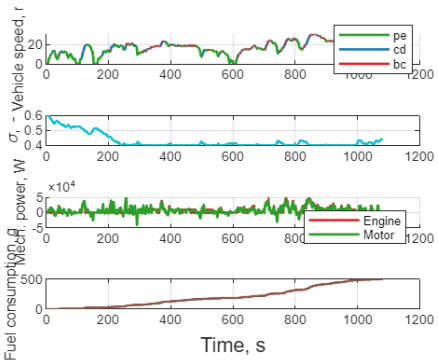
```
% Plot generator efficiency map in which all working points are marked
[~,~] = emMapWithPF(veh.gen, prof2, "gen","all");
```



Main Profiles

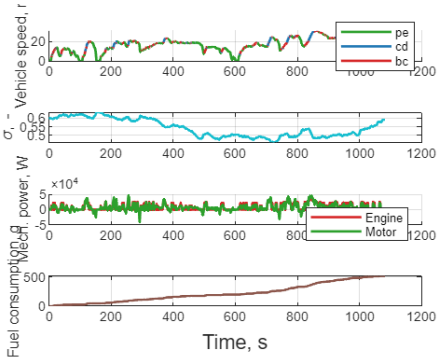
- **Without equivalence factor calibration**

```
[~,~] = mainProfiles(prof1)
```



- **With equivalence factor calibration**

```
[~,~] = mainProfiles(prof2)
```



Save results

Save the time profiles, fuel consumption (in kg), fuel economy (in l/100km), final SOC and calibrated equivalence factor in a mat-file named `results.mat`. Make sure they are named exactly as in the following code section.

If you have not done it before, use this code to convert your time profiles into scalar structures and pack them into a parent structure:

```
% prof is already a struct array containing all signals (no need to rebuild)
```

You can move this code wherever you need.

Finally, use this code to save your results (based on your calibrated equivalence factor)

```
% Store results
save("results.mat", "prof2", "fuelConsumption", "fuelEconomy", "finalSOC",
"eqFactor_cal");
```

- `prof`, `fuelConsumption` , `fuelEconomy`, `finalSOC` are as in Lab 01.
- `eqFactor` is the calibrated equivalence factor.

Controller analysis

Analyze the behaviour and the performance of the calibrated ECMS controller; support your analysis with relevant plots. You can use the postprocessing functions we provided and/or produce your own plots.

The ECMS controller

Describe your implementation of the ecms controller, by using code comments and this text section. **You will not get full marks if your function is not documented appropriately.**

In particular:

- Describe all inputs and outputs and why you need them.
- Name/briefly describe any other parameter that you create.
- Clearly explain in plain text how your controller works.

Remember that your ECMS must enforce the condition that the SOC never exceeds the following upper and lower thresholds:

- $\sigma_{\text{up}} = 0.8$,
- $\sigma_{\text{lo}} = 0.4$.

Also, not that this section *must* come at the end of the live script, or your function will not be properly recognized by the preceding part of the script. Local functions, including this one and any other function that you may want to define, must always be defined at the end of your script.

The code implements a controller based on the ECMS (Equivalent Consumption Minimization Strategy) approach for the optimal management of the internal combustion engine in a hybrid vehicle. The objective is to determine, at each simulation timestep, the optimal combination of engine torque *T_eng* and engine speed *w_eng* that minimizes the equivalent fuel consumption.

Definition of Candidate Controls

A discrete set of engine speed values is generated by initially adding the value 0 and then uniformly discretizing the interval between the idle speed *w_idle* and the maximum speed *w_max* using 20 equally spaced points (`linspace`). Similarly, the engine torque *T_eng* is discretized from 0 up to the maximum torque calculated over the engine speed range.

These values are then combined using a grid (`ndgrid`) to explore all possible control combinations.

Equivalent Consumption Calculation

For each candidate pair (w_eng,T_eng), the function `hev_cell_model` is called, returning:

- The future value of the state of charge (`SOC_next`)
- The instantaneous fuel consumption (`stageCost`)
- A feasibility indicator (`unfeas`)

The equivalent fuel consumption is calculated according to the formula:

$$\dot{m}_{f,\text{eq}} = \dot{m}_f - s \cdot \text{const} \cdot (\text{SOC}_{\text{next}} - \text{SOC})$$

where `const` is an energy-to-fuel conversion factor that accounts for the nominal battery energy (E_b) and the lower heating value of the fuel (*Q_LHV*), and is given by: $\text{const} = \frac{E_b}{Q_{\text{LHV}}} \cdot 1000 \cdot 3600.$

Constraint Handling

To avoid physically infeasible solutions:

- Combinations that cause the SOC to exceed the desired limits [0.4, 0.8] are penalized by adding a very large numerical penalty to the equivalent cost.
- Combinations marked as infeasible by the model (`unfeas`) are discarded by assigning them an infinite value.

Optimal Control Selection

Finally, the combination (w_eng,T_eng) that minimizes the equivalent fuel consumption among all feasible options is selected.

```
function [engSpd, engTrq] = ecmsControl(SOC, vehSpd, vehAcc, eqFactor, veh)

% Define the engine speed boundaries
w_idle = veh.eng.idleSpd; % Engine idle speed (rad/s)
w_max = veh.eng.maxSpd; % Maximum engine speed (rad/s)

% Generate a discrete set of engine speed values:
w_speed = [0 linspace(w_idle, w_max, 20)]; % (rad/s)

% Compute the maximum engine torque corresponding to the defined speed set
T_max = max(veh.eng.maxTrq(w_speed)); % (Nm)

% Generate 21 equally spaced torque values from 0 to T_max
T_eng = linspace(0, T_max, 21); % (Nm)

% Create a full grid of all (w_eng, T_eng) control combinations
[w_set,T_set] = ndgrid(w_speed,T_eng);

% Evaluate the HEV model for each control combination; returns:
% - SOC_next: predicted future value of the State of Charge
```

```
% - stageCost: fuel consumption rate (g/s)
% - unfeas: mask indicating infeasible control combinations
[SOC_next, stageCost, unfeas, ~] = hev_cell_model({SOC}, {w_set, T_set}, {vehSpd,
vehAcc}, veh) ;

% Compute the constant used to convert battery energy change to equivalent fuel
const = ((veh.batt.nomEnergy)/veh.eng.fuelLHV) * 1000 * 3600; % (g)

% Replicate current SOC to match the dimensions of the cost array
SOC = SOC * ones(size(stageCost)); % (-)

% Compute the equivalent fuel consumption:
% actual fuel flow minus electric contribution scaled by equivalence factor
fuelFlwRate_eq = stageCost - eqFactor * const * (SOC_next{1} -SOC); % (g/s)

% Penalize controls that cause SOC to exceed safe operational boundaries
fuelFlwRate_eq(SOC_next{1} < 0.4 | SOC_next {1} > 0.8) = fuelFlwRate_eq ( SOC_next
{1} < 0.4 | SOC_next {1} > 0.8) + 10^6;

 % Set cost to infinity for infeasible control combinations
fuelFlwRate_eq (unfeas == 1) = inf;

% Find the control combination that minimizes equivalent fuel consumption
[min_val, idx] = min(fuelFlwRate_eq, [], 'all');

min_fuelFlwRate_eq = min_val;

[a,b] = ind2sub(size(fuelFlwRate_eq), idx);

% Return the optimal engine speed and torque
engSpd = w_speed(a);
engTrq = T_eng(b);
end
```