Assignment #2: ECMS

Table of Contents

Group information

Group number: 12

Students:

- Emanuele Landolina, s349706
- Giuseppe Maria Marchese, s348145
- Walter Maggio, s343988

Load the cycle and vehicle data

In this initial phase, the necessary folders are added to the MATLAB path to ensure that all required functions and tools are available for the script.

The **vehicle data** and the **ARDC driving cycle** are then loaded into the workspace. From the ARDC cycle, the main variables (**time vector**, **vehicle speed**, and **vehicle acceleration)** are extracted and stored for further use in the simulation.

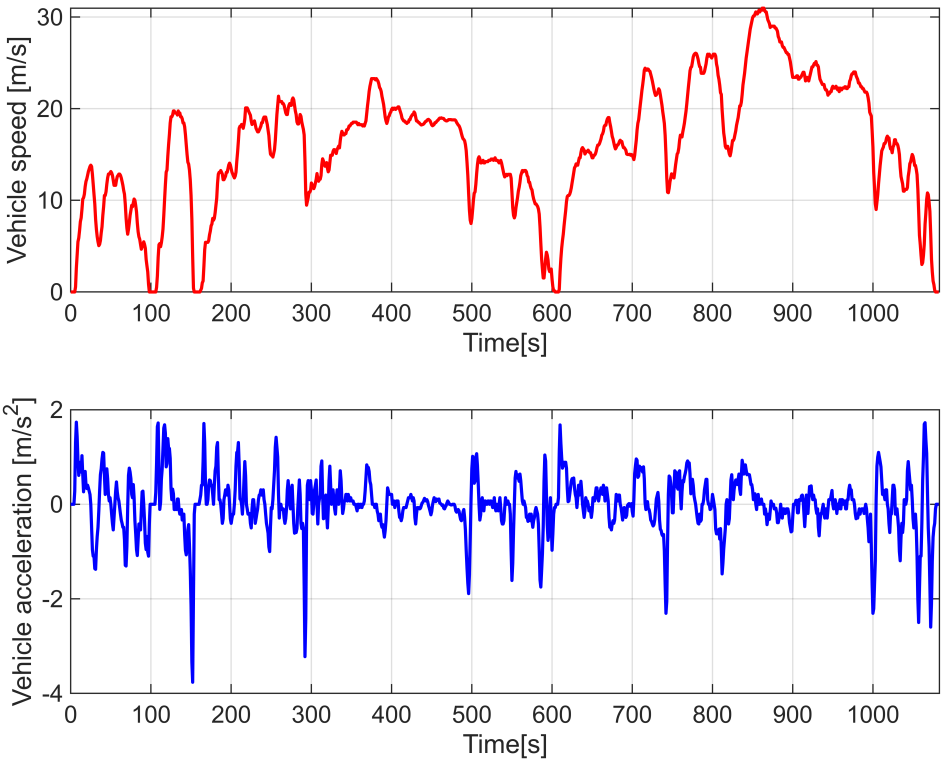
To gain a better understanding of the driving cycle being analyzed, two plots are generated:

- **Vehicle Speed vs. Time**
- **Vehicle Acceleration vs. Time**

These plots provide a clear visualization of the ARDC cycle's dynamics, helping to interpret the cycle's behavior and the demands it will impose on the vehicle during the simulation.

```
clear all
close all
clc

addpath("utilities");
addpath("models");
addpath("data");

%load vehicle data
vehData_raw = load("vehData.mat");

%load mission data
mission = load("ARDC.mat");

%time vector
time_vector = mission.time_s;

%Velocity and acceleration
vehAcc = mission.acceleration_m_s2;       %[m/s^2]
vehSpd = mission.speed_km_h/3.6;        %[m/s]
```

```
%% Vehicle speed and acceleration

%plot mission variables
t = tiledlayout(2,1);

nexttile(1)
plot(time_vector,vehSpd,"r",LineWidth=1.2)
xlabel("Time[s]")
ylabel("Vehicle speed [m/s]")
xlim([0 length(time_vector)])
grid on

nexttile(2)
plot(time_vector,vehAcc,"b",LineWidth=1.2)
xlabel("Time[s]")
ylabel("Vehicle acceleration [m/s^2]")
xlim([0 length(time_vector)])
grid on
```



The ARDC cycle is made by repeated sequences of acceleration and deceleration. It takes into account constant speed, sudden acceleration and stops. It is designed to stress the vehicles realistically and completely.

In addition to the plotting phase, a **scaling operation** is performed. The **vehicle data** provided are adjusted to align with the specific parameters assigned to the group.

```
%vehicle characteristic [Group 12]
engine_power = 66000;    %[W]
E_machine_power = 70000; %[W]
battery_energy = 945;    %[Wh]


%Scaled vehicle data
vehData = scaleVehData(vehData_raw,E_machine_power,engine_power,battery_energy);
```

Equivalence factor calibration

The optimal equivalence factor is determined using the bisection method. To guarantee finding the function's zero, two initial values, **`sLo=0`** and **`sHi=15`**, are selected to ensure opposite deviation signs (one positive and one negative). The deviation target is set to 0.1 to maintain charge-sustaining operation.

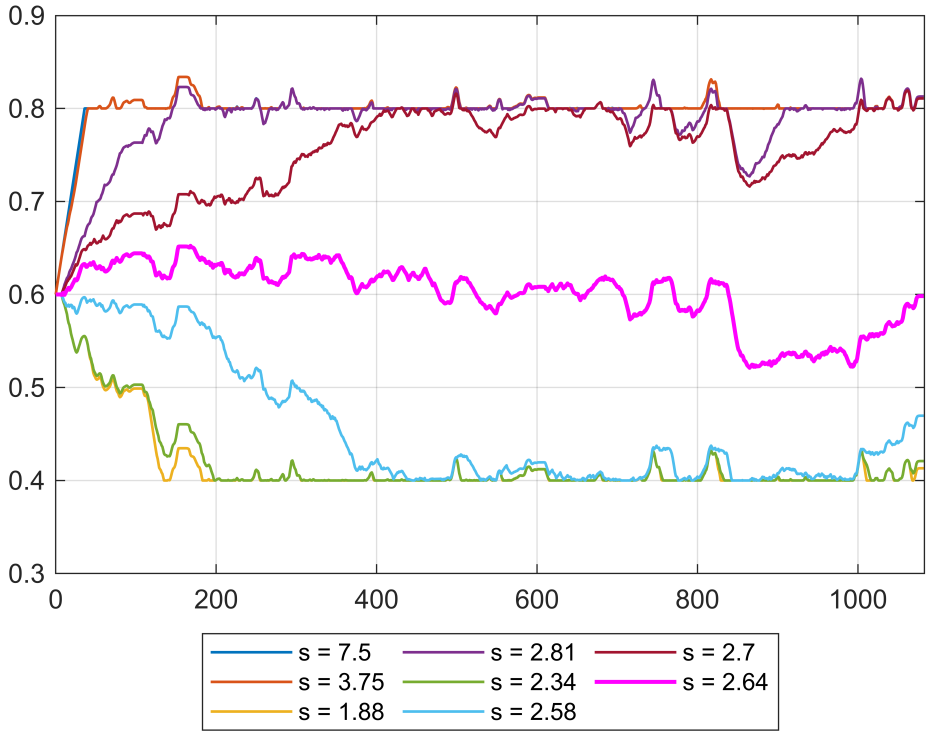
```
% Equivalence factor thresholds
sLo = 0;
sHi = 15;
% Deviation target
```

```
devTarget = 0.01;

[s_opt,SOC_try,lg_labels] =
bisectionAlgorithm(sLo,sHi,devTarget,mission,vehData);
```

```
Iteration number: 1
Boundaries:
    -sLow: 0.00
    -sHigh: 15.00
s value: 7.500
SOC deviation: 0.213
-------------------------------------------
Iteration number: 2
Boundaries:
    -sLow: 0.00
    -sHigh: 7.50
s value: 3.750
SOC deviation: 0.213
-------------------------------------------
Iteration number: 3
Boundaries:
    -sLow: 0.00
    -sHigh: 3.75
s value: 1.875
SOC deviation: -0.187
-------------------------------------------
Iteration number: 4
Boundaries:
    -sLow: 1.88
    -sHigh: 3.75
s value: 2.812
SOC deviation: 0.213
-------------------------------------------
Iteration number: 5
Boundaries:
    -sLow: 1.88
    -sHigh: 2.81
s value: 2.344
SOC deviation: -0.179
-------------------------------------------
Iteration number: 6
Boundaries:
    -sLow: 2.34
    -sHigh: 2.81
s value: 2.578
SOC deviation: -0.130
-------------------------------------------
Iteration number: 7
Boundaries:
    -sLow: 2.58
    -sHigh: 2.81
s value: 2.695
SOC deviation: 0.211
-------------------------------------------
Iteration number: 8
Boundaries:
    -sLow: 2.58
    -sHigh: 2.70
s value: 2.637
SOC deviation: -0.002
-------------------------------------------
```

```
% Iteration SOC plot
figure()
plot(time_vector,SOC_try(1:end-1,1:length(time_vector)),LineWidth=1)
hold on
```

```
plot(time_vector,SOC_try(end,1:length(time_vector)),"m",LineWidth=1.5)
grid on
axis([0 length(time_vector) 0.30 0.90])
legend(lg_labels,'Location',"southoutside",'NumColumns', 3)
```



This plot illustrates the functioning of the bisection algorithm: starting from the first iteration, the battery SOC quickly reaches the upper limit and remains there throughout the mission. In the following iterations, the s value is progressively lowered, leading to the opposite trend. Subsequently, the s value is finely adjusted to meet the target deviation, stabilizing around 2.6. The optimal equivalence factor identified by the algorithm is 2.64, corresponding to an SOC deviation of -0.002.

Simulation run

The simulation is run using the function `simulationLoop` which sees as inputs:

- the equivalence factor (`s_opt`), previously determined using the bisection algorithm;
- the `mission;`
- the vehicle data stored in the struct `vehData`;

The outputs are:

- the State of Charge (`SOC`) over the entire mission;
- the fuel flow rate (`fuelflwRate`);
- the number of unfeasibilities (`unfeas`);

- the profiles (`prof`);
- the engine state (`engState`), indicating whether the engine is on or off.

```
%Setup for simulation
[SOC, fuelflwRate, unfeas, prof,engState] =
simulationLoop(s_opt,mission,vehData);
```

Results

To analyze the performance of the controller and obtain the objective function of this project three metrics are evaluated:

- **`fuelConsumption`** is the total fuel consumption for the whole mission (in kg). It is evaluated performing an integral of the fuel flow rate over time by using the function trapz. The arguments of the function are the `time_vector` and the `prof_struct`, that is a scalar structure where all the information about the vehicle state are saved instant by instant during the simulation. The goal was to remain around 0.5 kg of fuel consumption for the entire mission, and this target was fully achieved. This result validates the correct functioning of the designed thermostat controller.

```
prof_struct = structArray2struct(prof);

fuelConsumption = trapz(time_vector,prof_struct.fuelFlwRate)/1000;
fprintf("Fuel consumption: %.3f kg\n", fuelConsumption);
```

```
Fuel consumption: 0.515 kg
```

- **`fuelEconomy`** represents the distance-specific fuel consumption for the entire mission, expressed in l/100 km, to provide a clearer and more explicit result. To compute this, the mission distance is first evaluated by integrating the speed profile over time, and then the fuel economy is calculated, taking into account the fuel density (`fuel_rho`).

$$fuelEconomy = \frac{fuelConsumption}{fuel\_rho \cdot distance} \cdot 100$$

```
distance = trapz(time_vector,vehSpd)/1000;
fuel_rho = vehData.eng.fuelDensity; %[Kg/l]
fuelEconomy = (fuelConsumption/(fuel_rho*distance))*100;
fprintf("Fuel economy: %.3f l/100km\n", fuelEconomy );
```

```
Fuel economy: 3.795 l/100km
```

- **finalSOC** represents the final state of charge (SOC) of the battery, indicating the battery's status at the end of the mission and, consequently, the amount of energy still available. This value is crucial for verifying the proper functioning of the ECMS, whose objective is

to ensure that the terminal SOC deviation remains within 0.01, meaning that the final SOC should be as close as possible to the target value.

```
finalSOC = SOC(end);
fprintf("Final SOC: %.3f\n", finalSOC );
```

```
Final SOC: 0.598
```

- **equivalence factor** is the parameter used to evaluate the weight of the electrical energy with respect to the fuel consumption of the engine, in order to calibrate the distribution of energy supply between engine and motor. The final result has been 2.637, in line with the expectations (referring to other cycles parameter).

```
eqFactor = s_opt;
fprintf("Equivalence factor: %.3f\n", eqFactor );
```

```
Equivalence factor: 2.637
```

Save results

The obtained results are saved in a .mat file.

```
% Store results
save("results.mat", "prof", "fuelConsumption", "fuelEconomy", "finalSOC",
"eqFactor")
```

Controller results analysis

To analyze the behavior and the performance of the calibrated ECMS controller, different plots are used:

- **SOC** trend with operating modes;
- **Main profiles**
- **Generator operating point** with operating modes;
- **Motor operating point** with operating modes;
- **Requested power** by the electric motor
- **Engine operating point** with operating modes;

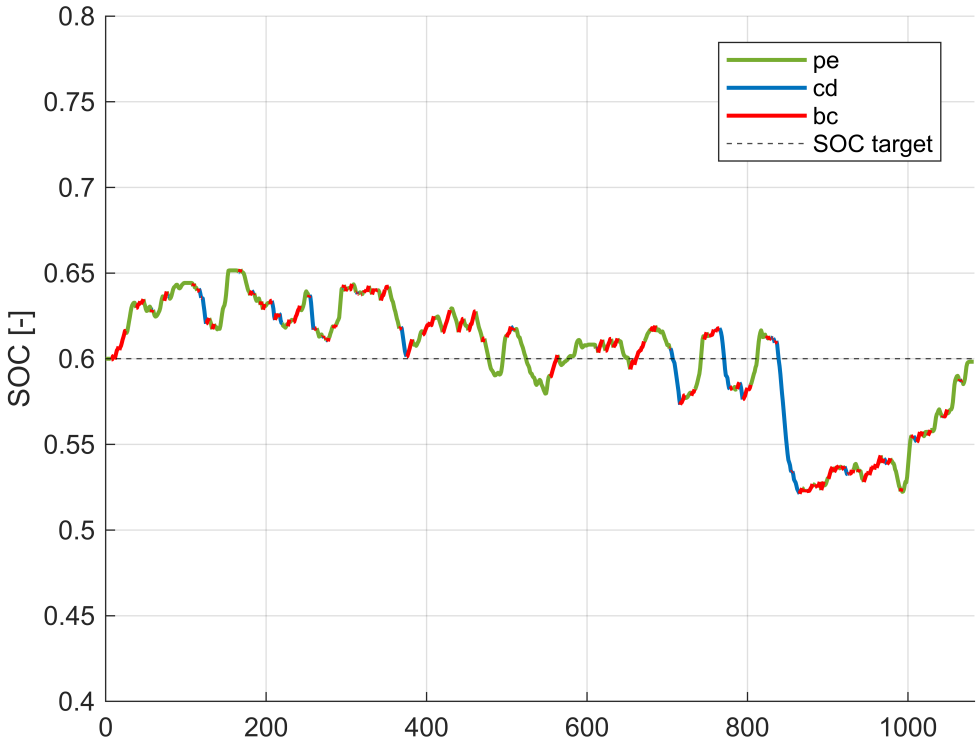
SOC vs time

This plot shows the vehicle speed throughout the ARDC cycle (0-1200s).

It is used a color-code based on three operating modes:

- **Green (pe):** pure electric mode;
- **Blue (cd):** charge-depleting mode;
- **Red (bc):** battery charging mode (engine active to recharge the battery).

```
SOCwithPF(prof);
```



The trend shows the desired behavior of **charge sustaining**, where SOC has an average value almost always between +-0,05.

A peculiarity of this controller, which uses a bisection algorithm to tune the equivalence factor, is that near the end of the cycle it depletes the battery to exploit the big deceleration in last part of the cycle and then charge it to achieve the goal of having final SOC equal to the initial while minimizing fuel consumption. This happens clearly because the mission is known in advance.

The controller mostly operates in pure electric, trying to recover as much energy as possible from the braking maneuvers. Only a few times it operates in charge depleting, when very high power peaks are requested.

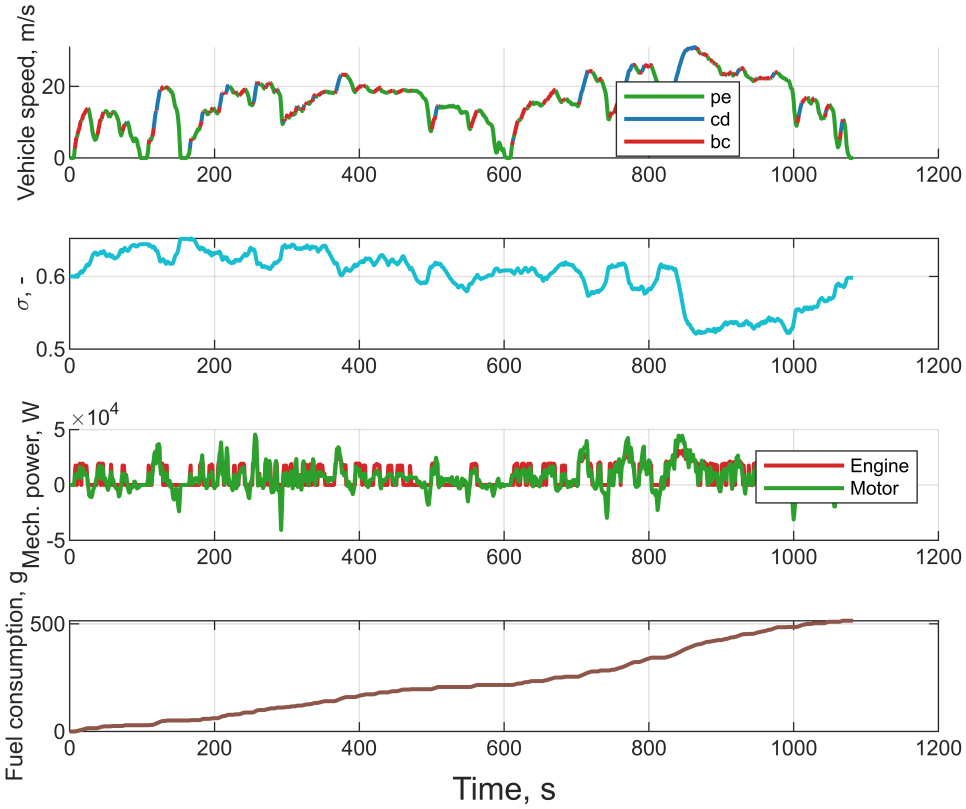
Main profiles

The main profiles consist of four key plots that should be analyzed together to fully understand the control system’s behavior throughout the entire cycle.

- **Vehicle speed** trend with operating modes;
- **SOC** trend;
- **Mechanical power** trend with engine/motor mode;
- **Fuel consumption** over the entire mission.

The same color-coding as the SOC vs time plot is eventually used.

```
mainProfiles(prof);
```



Plot 1: Vehicle Speed vs Time

Considering the nature of the ARDC cycle, which is a transient one, frequent variations of velocity are present.

At high acceleration peaks therefore when high speeds are reached, the system operates in a **charge-depleting mode** to fully exploit the electric part of the powertrain giving extra power thanks to the use of the engine.

At medium speeds, the engine typically works in **battery-charging mode**, providing power both to the wheels and to recharge the battery.

At low velocities, the vehicle usually operates in **pure electric mode** to maximize efficiency.

When speed decreases, due either to braking maneuvers or simple deceleration, the vehicle uses electric drive either to improve efficiency or to recharge the battery through **regenerative braking**.

Plot 2: State of Charge vs Time

This plot shows the SOC evolution over time, highlighting the charge and discharge phases between two set limits: **0.4 and 0.8**.

The initial SOC is set at **0.6**, meaning that 60% of the battery's energy is available at the start of the mission. This value is also enforced as the final SOC, ensuring the energy balance required by the control strategy.

Plot 3: Mechanical Power vs Time

The cycle has a lot of transients, so the power request oscillates significantly, especially for the electric motor, whose power has high peaks in input and output. That happens because accelerations and braking maneuvers suddenly follows one after the other.

The engine, in contrast, tends to operate always at constant power level, to optimize the fuel consumption. Around t = 850 s, as the battery becomes discharged, the engine produces its **maximum output power**, resulting in a slight drop in efficiency. This is done to exploit the hybridization capability of the powertrain and reach a final SOC equal to the initial.

Plot 4: Fuel consumption vs Time

Fuel consumption during the cycle remains almost constant trend, largely because the ARDC cycle has not different phases (like the WLTP that is divided into urban, suburban and highway) but maintains a consistent driving pattern all along the mission.

The steepest increase in fuel consumption occurs between 700 and 1000 s, corresponding to the part of the cycle where the battery gets discharged, the engine increases the output power and the highest speed is reached.

Overall, this plot demonstrates the effective performance of the controller, which maintains low and steady fuel consumption despite the highly dynamic conditions of the driving cycle.

Generator operating points

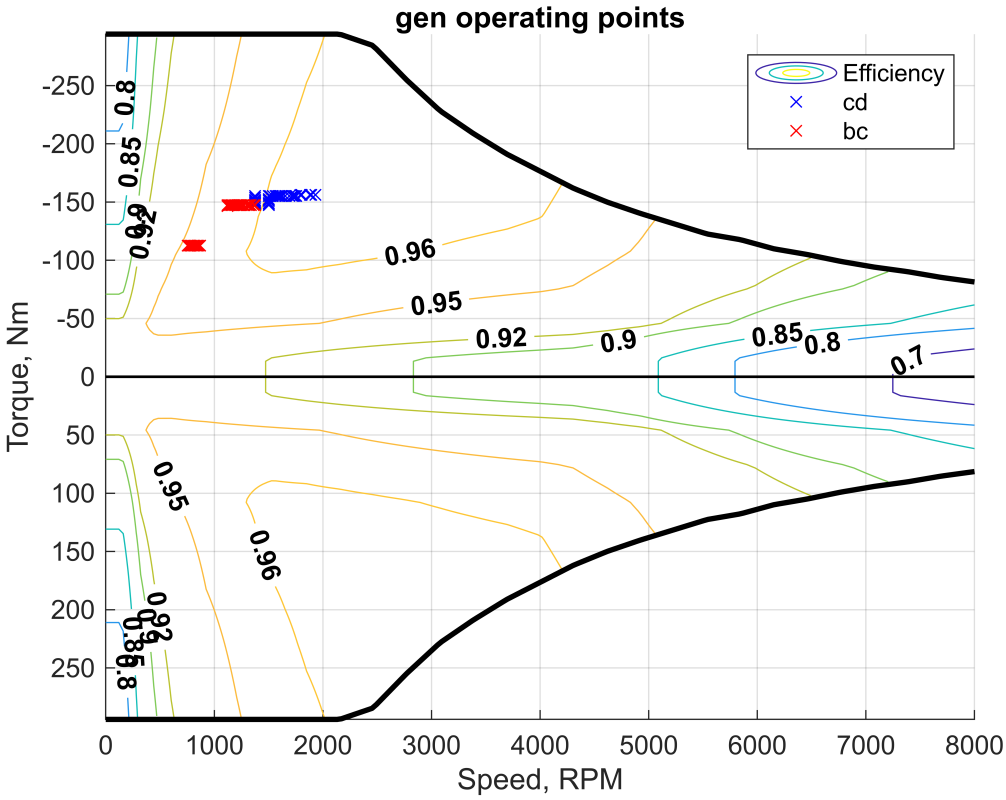
In the following plot, the generator efficiency map is presented, with all the working points plotted.

It is possible to distinguish two operational modes: **battery charging** (**bc**) and **charge depleting** (**cd**), which uses the engine combined to the battery energy.

The plot demonstrates the effective behavior of the controller, which consistently maintains the operating points within the high-efficiency region (0.95–0.96). It does not always operate exactly at 0.96, likely because it prioritizes minimizing engine fuel consumption, which represents a more optimal overall strategy.

Notably, the charge-depleting mode tends to operate at higher efficiencies compared to the battery charging phase, for the same reason mentioned above.

```
emMapWithPF(vehData.gen,prof,"gen","all");
```



Motor operating points

In the following plot, the Electric Motor efficiency map is presented, with all the working points plotted. It is possible to distinguish three operating modes: **battery charging** (**bc**), **charge depleting** (**cd**) and **pure electric** (**pe**).

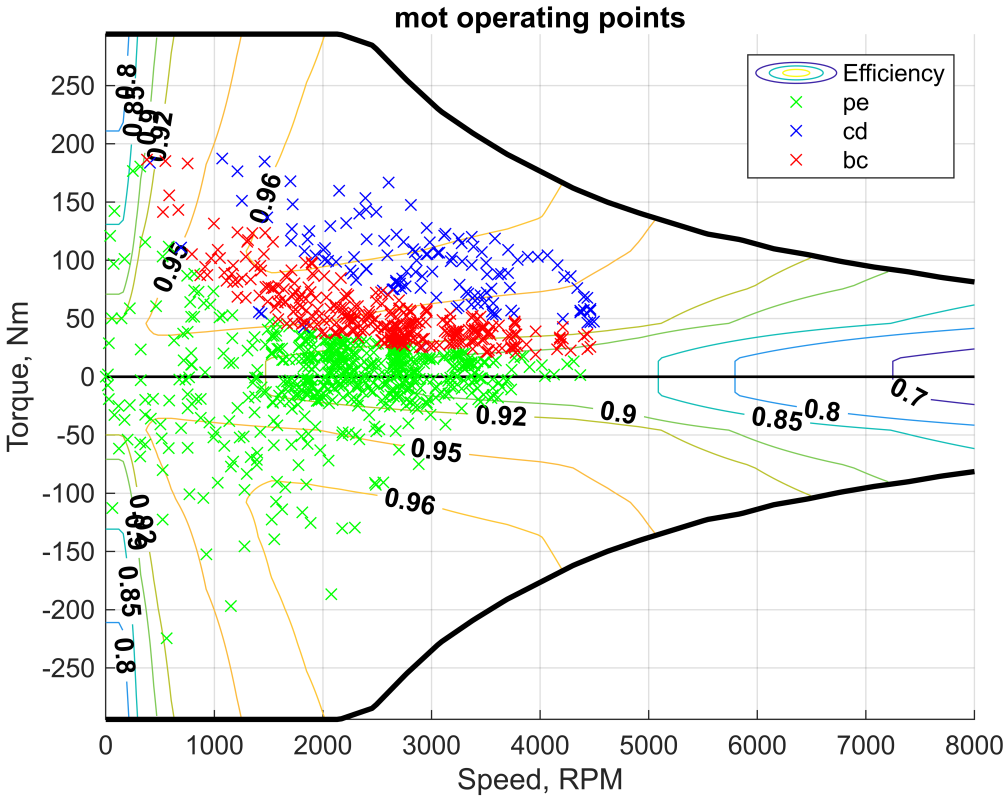
The pure electric operating points can be subdivided in two parts: the ones in the positive quadrant (the upper one) represent the pure electric traction mode, while the ones in the negative quadrant represent the charging mode through regenerative braking.

It is noticeable a clear division in the operating points in function of the amount of power request:

- the low ones are managed by the pure electric mode;
- the medium ones are managed by the engine in the battery charge mode;
- the high ones are managed both by engine and electric pathway.

This strategy's aim is to exploit the high efficiency at low regime of the E-Motor and the lowest fuel consumption operating points of the engine that lie at medium loads.

```
emMapWithPF(vehData.gen,prof,"mot","all");
```



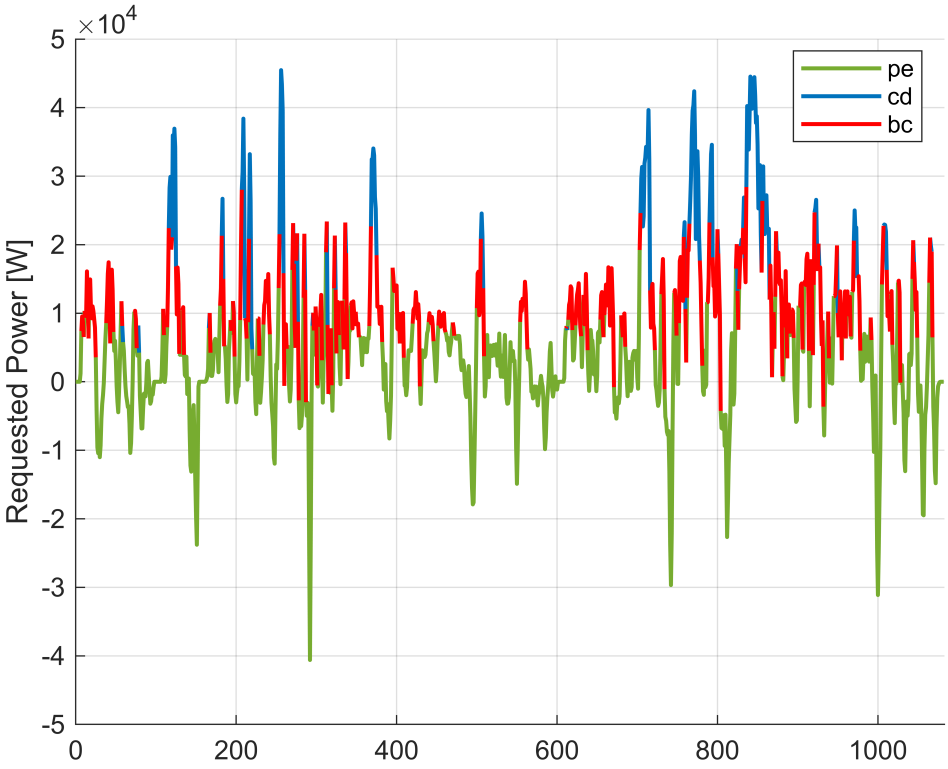
Requested power

As outlined in the previous plot, the next graph illustrates how the controller operates. At low power levels, the system tends to favor pure electric mode to take advantage of the electric motor's high efficiency, thereby avoiding the use of the internal combustion engine, which operates with very low efficiency under these conditions.

At moderate power demand, the ECMS controller prefers to charge the battery. This increases the overall power required from the engine, allowing it to operate at higher speeds and in its optimal efficiency range.

When the requested power exceeds the battery's output limits, the engine supplements the remaining power demand. This strategy ensures that the electric motor can still deliver its maximum potential, optimizing overall system performance.

```
reqPwr_PF(prof);
```

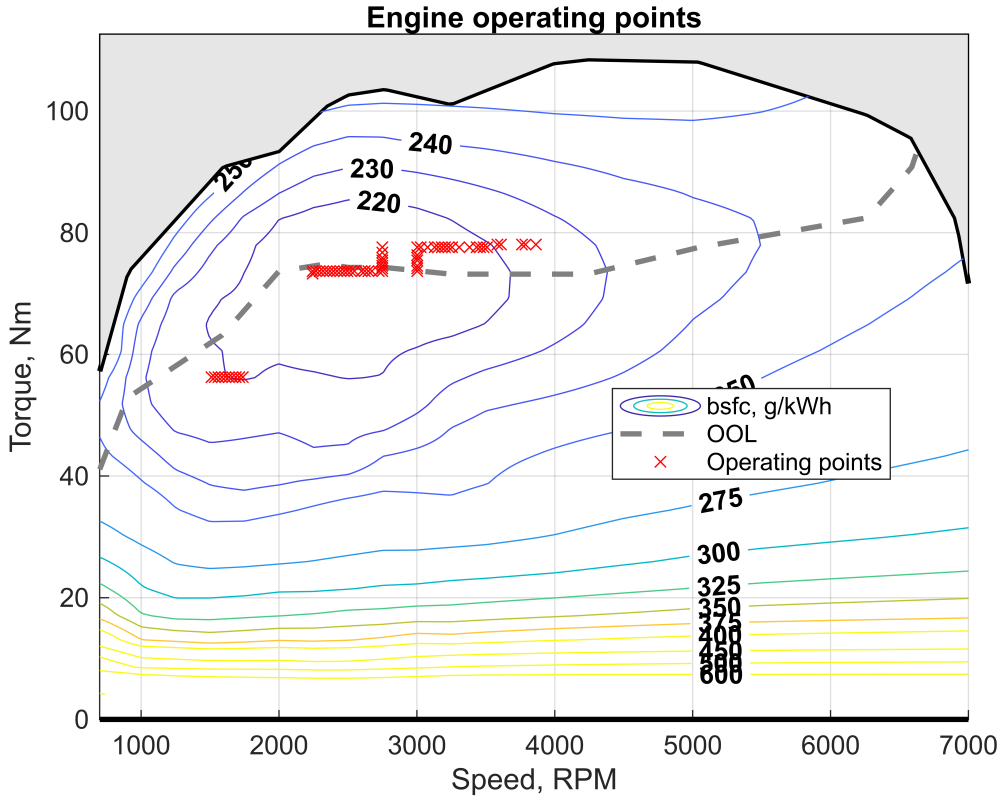


Engine operating points

The plot shows the engine map with **Brake Specific Fuel Consumption (BSFC)** contours, where the controller's operating points throughout the entire driving cycle are indicated by red crosses. This visualization helps assess the effectiveness of the energy management strategy, highlighting how the controller selects operating points that fall within the areas of **highest efficiency**, that corresponds to the regions with the lowest BSFC values.

The gray dashed line represents the **Optimal Operating Line (OOL)**, which indicates the engine's most efficient operating region. As shown in the plot, the operating points tend to align with or remain close to this line, demonstrating how the controller effectively exploits the engine's high-efficiency zones while meeting the torque and speed demands of the mission.

```
engMapWithPF(vehData.eng,prof,"bsfc");
```



Battery power

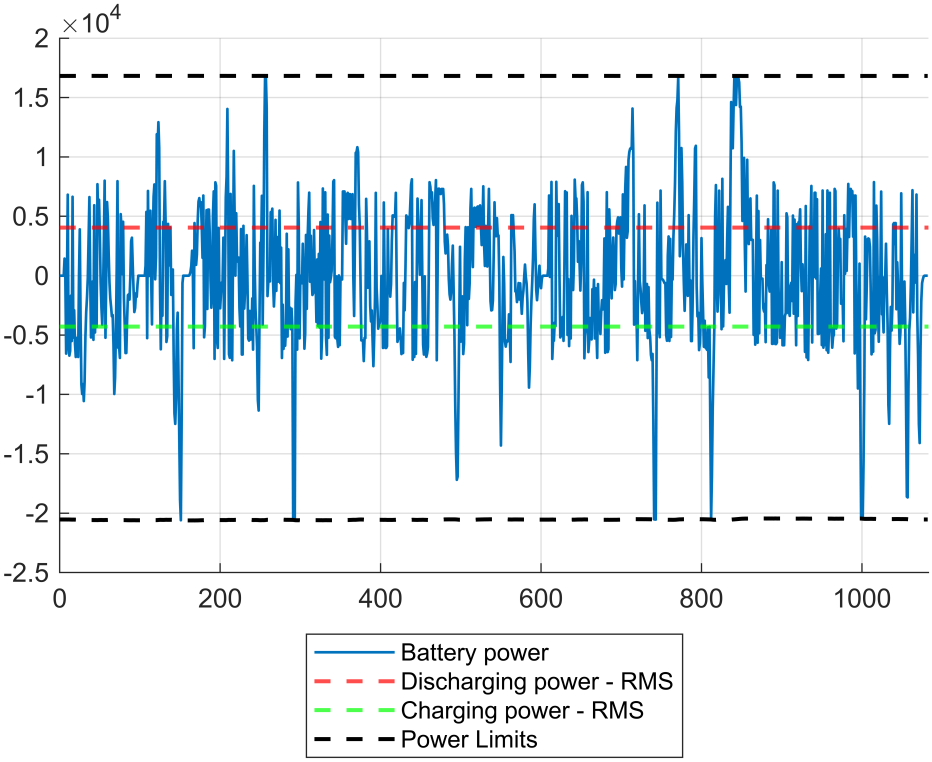
This plot shows the battery power over time during the mission. Positive values indicate power delivered by the battery to the powertrain (discharge), while negative values represent regenerative braking or charging phases (charge).

The power fluctuates significantly due to the transient nature of the ARDC cycle, which involves frequent accelerations and decelerations.

The dashed black lines indicate the **maximum and minimum power limits** imposed by the battery specifications. The controller ensures that battery operation stays within these boundaries throughout the cycle, demonstrating an effective energy management and compliance with safety constraints.

Overall, the controller exploits an appropriate use of the battery with tendentiously a maximum power of around 7 kW in discharging and 6 kW in charging, few high power peaks are present but not affect to much the aging of the battery. Analyzing RMS values, simulation shows a result of 4 kW in discharging and 4,2 kW in charging that are easily manageable for a battery designed for hybrid vehicles.

```
battPwrPlot(time_vector,prof,vehData);
```



Histogram engine time on and off

The **thermal management** of the engine and the after-treatment system (ATS) plays a key role in minimizing fuel consumption and pollutant emissions. To achieve maximum efficiency, both the engine and the ATS must reach optimal operating temperatures. In particular, the ATS becomes highly effective only after reaching its **light-off temperature**.

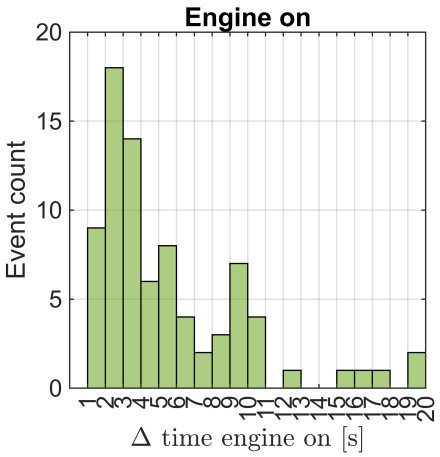
The simulation does not account for the thermal state of the engine and ATS. However, some useful observations can still be made. As illustrated in the histogram on the left, the **engine is typically turned on for only a few seconds** (1-4 seconds). This is a very short duration for a gasoline engine to operate, potentially causing thermal inefficiencies and related issues.

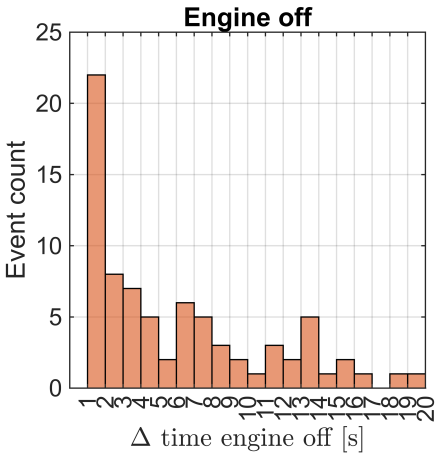
On the other hand, the histogram on the right shows that the **engine is often turned off for very brief periods** (1-2 seconds). In such cases, the thermal inertia of the engine and ATS allows them to maintain an acceptable temperature range despite frequent cycling.

Overall, it is evident that the **engine is cranked multiple time**, up to 15 times per minute. This frequent start-stop behavior leads to significant energy losses, as each cranking event requires the electric motor to accelerate the engine from a standstill, overcoming its inertia and consuming energy that is ultimately wasted when the engine shuts off again.

In conclusion, the control strategy could be optimized by reducing the frequency of engine start-stop events. By **penalizing unnecessary switching**, it would be possible to ensure better thermal conditions for both the engine and the ATS, leading to improved performance and reduced energy waste.

```
hist_engstate(engState);
```





Simulation Loop function

This section presents the simulation of the ECMS controller, whose objective is to optimize the selected objective function, in this case, fuel consumption, while operating in charge-sustaining mode.

The simulation begins by setting the initial conditions:

- **State of Charge (SOC)**: 0.6
- **Engine state:** OFF (corresponding to the boolean value `engState` = 0)

The simulation is performed using a for-loop iterated over the time steps of the ARDC cycle. Exogenous inputs are extracted from the vehicle data structure and the ARDC cycle profile.

The main functions used during the simulation are:

- **`ecmsControl`**: Given the SOC, vehicle speed and acceleration, vehicle dataset, and equivalence factor s, this function determines the control variables (engine speed `engSpd` and engine torque `engTrq`) that minimize the instantaneous fuel consumption cost function.
- **`hev_model`**: Using a backward approach, this model takes as input the current SOC, vehicle speed and acceleration, dataset, and control variables to compute the next system states. Specifically, it outputs the updated SOC and the fuel flow rate.

- **unfeas** and **prof** structures: These utilities help assess the controller's performance and identify any infeasible operating points.

```
function [SOC, fuelflwRate, unfeas, prof, engState] =
simulationLoop(s,mission,vehData)

%time vector
time_vector = mission.time_s;

%Velocity and acceleration
vehAcc = mission.acceleration_m_s2;       %[m/s^2]
vehSpd = mission.speed_km_h/3.6;        %[m/s]

%Initializing SOC and engine state
SOC(1) = 0.6;
engState(1) = 0; % engine is off

% Simulation Loop
for n = 1:length(time_vector)

    % Control variables computed using ECMS control
    [engSpd(n), engTrq(n)] = ecmsControl(SOC(n),vehSpd(n),vehAcc(n),vehData,s);

    % Using the vehicle model the states variables are evaluated
    [SOC(n+1), fuelflwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n),
[engSpd(n), engTrq(n)], [vehSpd(n), vehAcc(n)], vehData);

    % engine state for each instant is saved
    engState(n+1) = engSpd(n) >= vehData.eng.idleSpd & engTrq(n) > 0;

end

end
```

Bisection algorithm function

A **bisection algorithm** is developed to calibrate the **equivalence factor** `s`. The function used to evaluate it is the final State Of Charge (SOC), which must be minimized until the deviation (the difference between the initial and final SOC) is equal to or less than `devTarget`.

To achieve this, a `while` loop is used, which runs until the **target condition** is met (**`deviation <= devTarget`**). The loop is terminated by a `break` statement placed inside an `if` condition that checks this requirement.

Two initial values, sLo and sHi, are set using tuned parameters to produce SOC deviations with opposite signs.

Inside the `while` loop, a variable `s_Try` is calculated through an average of `sLo` and `sHi`. During the iteration, this value is used in the `simulationLoop` to obtain a corresponding deviation value (`devTry`), which is then compared to the target in the `if` condition.

If the target is not met, `s_Try` replaces `sLo` if the deviation is negative or `sHi` if the deviation is positive. The final value, when the loop converge, is saved in the variable `s_opt`.

```
function [s_opt,SOC_try,labels] =
bisectionAlgorithm(sLo,sHi,devTarget,mission,vehData)


iterationCount = 0;

% Implementation of bisection algorithm
while true

    % Iteration information and display
    iterationCount = iterationCount+1;

    fprintf("Iteration number: %d \n",iterationCount)
    fprintf("Boundaries: \n")
    fprintf("    -sLow: %.2f \n",sLo)
    fprintf("    -sHigh: %.2f \n",sHi)

    % s factor of the iteration
    s_Try = (sLo+sHi)/2;

    % Plot labels
    labels(iterationCount,:) = join(["s =",string(round(s_Try,2))]);

    fprintf("s value: %.3f \n",s_Try)

    % SOC output of the simulation
    SOC_try(iterationCount,:) = simulationLoop(s_Try,mission,vehData);

    % Deviation of SOC
    devTry = SOC_try(iterationCount,end)-SOC_try(iterationCount,1);

    fprintf("SOC deviation: %.3f \n",devTry)
    fprintf("------------------------------------------- \n")

    % Check section
    if abs(devTry) <= devTarget % Target fulfill
        s_opt = s_Try;
        break
    else % Deviation out of target
        if devTry > 0
            sHi = s_Try;
        else
            sLo = s_Try;
        end
    end
```

```
end
end
```

The ECMS controller

The **ECMS** (Equivalent Consumption Minimization Strategy) controller is used to optimized the fuel consumption over the **ARDC cycle**. The cycle is known in advance, therefore it is not necessary to give a look-ahead capability to the controller (designing a penalty function) but the equivalence factor can be easily obtained using the bi-sectional algorithm. Despite that ECMS controller perform a **local optimization** which means that at each time instant find the **sub-optimal solution**.

The controller function receives the following **inputs**:

- **`SOC:`** from the previous iteration. Be an exogenous variable, the controller need the previous SOC to compute the next.
- **`vehSpd:`** the vehicle speed at a given moment;
- **`vehAcc:`** the vehicle's acceleration at the same instant. These inputs are fed into the `hev_cell_model` to compute the following vehicle states.
- **`vehData:`** structure encompasses all relevant vehicle and mechanical data.
- **`s:`** equivalence factor necessary to give an appropriate weight to the battery energy converting it into equivalent fuel consumption.

The function **outputs** the following control variables:

- **`engSpeed`**: The speed at which the engine should operate during the current iteration.
- **`engTorque`**: The torque at which the engine should operate during the current iteration.

Controller logic

The aim of the controller is to analyze all possible working points of the engine and find optimal combination at each time step. In order to do that first of all two vectors are defined:

- **engSpd_vect**: vector of engine speed that swipe from idle to the maximum speed with a defined discretization (`res` variable). In addition, the 0 point is added to simulate engine off state.
- **engTrq_vect**: vector of engine torque that swipe from 0 to the maximum torque value with a defined discretization (`res` variable). An additional 0 value point is added to set congruent dimensions of the two vectors.

Successively, exploiting the `ndgrid` function ang giving to it the two vector as input a grid of all the possible **control candidate combinations** is generated.

In the following step the engine speed and torque matrices, SOC, vehicle speed, acceleration and data are given as input to the `hev_cell_model`. The function compute for each couple of control candidates the future SOC and the stage cost (fuel flow rate) and eventually unfeasibility.

With this parameters is possible to implement the ECMS formulation shown below:

$$\dot{m}_{f,eq} = \dot{m}_f - s \cdot \frac{E_b}{Q_{LHV}} \cdot \dot{SOC}$$

Where

$$\dot{SOC} = \frac{SOC_{i+1} + SOC_i}{dt}$$

The result is a matrix (`res+1xres+1`) that assign to each control candidates the equivalent fuel flow rate (`mf_eq`).

Not all of them are acceptable due to unfeasibility or SOC exceeding its thresholds (`SOChi` and `SOClo`) therefore all these points are penalized setting their equivalent fuel consumption equal to `inf`.

At the end, considering the minimum equivalent fuel consumption, the corresponding couple of control candidates are taken into account to set the output engine speed and torque.

Controller implementation

In the first part of the code the principle parameters of the controller are set:

- `SOChi` and `SOClo`: state of charge thresholds
- `dt`: time discretization of the simulation useful to compute the SOC derivative.
- `batteryEnergy`: energy capacity of the battery necessary for the implementation of equivalent fuel consumption.
- `Q_LHV`: lower heating value of the fuel used in the equivalent fuel consumption formula
- `res`: discretization parameter of engine map set equal to 250 in order to guarantee a reliable result without compromise the simulation time

As described below, the engine speed and torque are created using `linspace` function (length equal to `res`) and meshed in a grid thanks to `ndgrid` function. Using the `hev_cell_model` the next SOC, the stage cost and unfeasibility are obtained:

- `SOC` as cell data frame where a matrix `(res+1)x(res+1)` is stored
- `mf_eng` (engine fuel flow rate) as matrix `(res+1)x(res+1)`
- `unfeas` as logical matrix `(res+1)x(res+1)`

Once computed the equivalent fuel consumption matrix, the unacceptable working points must be penalized and to do that the matlab `logical indexing` is exploited. Matlab logical indexing allows selecting and modifying elements of an array that satisfy specific conditions, enabling efficient and concise manipulation of matrix values without the need of explicit loops. Therefore, all the points

of equivalent fuel consumption matrix (`mf_eq`) that correspond to `unfeas == 1`, `SOC <= SOClo` or `SOC >= SOChi` are set equal to `inf`.

Using the same method, the value of engines speed and torque that corresponds to the minimum of `mf_eq` are extrapolated and set as output `engSpeed` and `engTorque`. In case of multiple minimum points only the first at minimum speed is taken into account.

```
function [engSpeed, engTorque] = ecmsControl(SOC,vehSpd,vehAcc,vehData,s)

% Define the higher threshold of the SOC
SOChi = 0.8;

% Define the lower threshold of the SOC
SOClo = 0.4;

% Time discretization of the simulation
dt = vehData.dt;  %[s]

% Define energy capacity of the battery
batteryEnergy = vehData.batt.nomEnergy*3600; %[W*s]

% Define lower heating value of the fuel
Q_LHV = vehData.eng.fuelLHV/1000;  %[J/g]

% Define the resolution of the grid (discretization of the maps)
res = 250;

% Create the speed vector that swipe from idle to max speed
engSpd_vect = linspace(vehData.eng.idleSpd,vehData.eng.maxSpd, res); %[rad/s]
% Zero speed is included in the vector to consider engine off case
engSpd_vect = [0,engSpd_vect];

% Define the maximum torque that can be exploited by the engine
engMaxTrq = max(vehData.eng.maxTrq(engSpd_vect));  %[Nm]

% Create the torque vector that swipe from 0 to maximum torque
engTrq_vect = linspace(0,engMaxTrq,res);  %[Nm]

% Zero is added to the torque vector in correspondence to the zero speed
% point
engTrq_vect = [0,engTrq_vect];

% Grid of engine speed and torque points is created
[engSpd_grid, engTrq_grid] = ndgrid(engSpd_vect,engTrq_vect);

% The vehicle model is used to define SOC, fuel consumption and eventual
% unfeas after each working point of the grid
[SOC_next,mf_eng,unfeas] = hev_cell_model({SOC},{engSpd_grid,engTrq_grid},
{vehSpd,vehAcc},vehData);

% Conversion cell to array
```

```
SOC_next = SOC_next{1};

% The derivative of the SOC is computed for each point of the matrix
SOC_dot = (SOC_next-SOC)./dt;

% Equivalent consumption formulation is implemented
mf_eq = mf_eng -s.*(batteryEnergy/Q_LHV).*SOC_dot; %[g/s]

% mf_eq -> matrix dimension (res x res)
% SOC_next -> matrix dimension (res x res)
% unfeas -> matrix dimension (res x res)

% For a given point penalty is given in case of an unfeas is present
mf_eq(unfeas == 1) = inf;

% For a given point penalty is given in case of SOC
% overcame the higher SOC threshold
mf_eq(SOC_next > SOChi) = inf;

% For a given point penalty is given in case of SOC
% go below the lower SOC threshold
mf_eq(SOC_next < SOClo) = inf;

% Extract the minimum equivalent fuel consumption
minEqMf = min(mf_eq(:));

% Engine speed corresponding to the minimum equivalent fuel consumption is
% extracted
engSpeed = engSpd_grid(mf_eq == minEqMf);

% Engine torque corresponding to the minimum equivalent fuel consumption is
% extracted
engTorque = engTrq_grid(mf_eq == minEqMf);

% In case of multiple minimum points only the first, at lower speed, is
% considered
engSpeed = engSpeed(1);
engTorque = engTorque(1);


end
```

Conclusion

The result analysis confirms that the ECMS controller effectively managed the energy balance and fuel consumption across the ARDC driving cycle. A final fuel consumption of 0.515 kg was achieved, corresponding to an impressive fuel economy of 3.795 l/100km, meeting the project's target of staying around 0.5 kg of fuel use. Furthermore, the final State of Charge (SOC) closely matched the initial value, with a deviation well within the ±0.01 margin, validating the success of the equivalence factor calibration.

The controller behavior showed that pure electric operation was favored whenever possible, exploiting regenerative braking to maximize battery efficiency. Engine usage was concentrated in high-efficiency zones, as seen in the operating maps, while battery charging and discharging stayed within the designed limits, preserving battery health. However, frequent engine start-stop events introduced some inefficiencies, suggesting potential for improvement in future strategies by penalizing unnecessary engine cranking.

Overall, the ECMS controller displayed a strong ability to minimize fuel consumption, maintain battery balance, and operate the vehicle efficiently under dynamic driving conditions, demonstrating the effectiveness of the control strategy and calibration process.