Assignment #2: ECMS

Table of Contents

Group information....1
Load the cycle and vehicle data....1
Equivalence factor calibration....3
Run a simulation with the calibrated equivalence factor....5
Save results....5
Controller analysis....5
The ECMS controller....18
Step 1: initialization of all combinations of engine speed and torque....19
Step 2: evaluation of the fuel flow rate corresponding to each combination....20
Step 3: exclusion of unfeasible combinations due to system constraints....20
Step 4: selection of the engine speed and torque pair that minimizes the fuel flow rate among the feasible options....21

Group information

Group number: 33

Students:

- Loris Fonseca, s342696
- Maurizio Pio Vergara, s346643
- Nikoloz Kapanadze, s342649

Load the cycle and vehicle data

The script is organized in a step-by-step structure that follows the required tasks of the ECMS assignment. It begins by loading the provided ARDC cycle and the hybrid vehicle data, which are used throughout the simulations. The drive cycle velocity and acceleration are also plotted to visualize the conditions the vehicle is expected to handle.

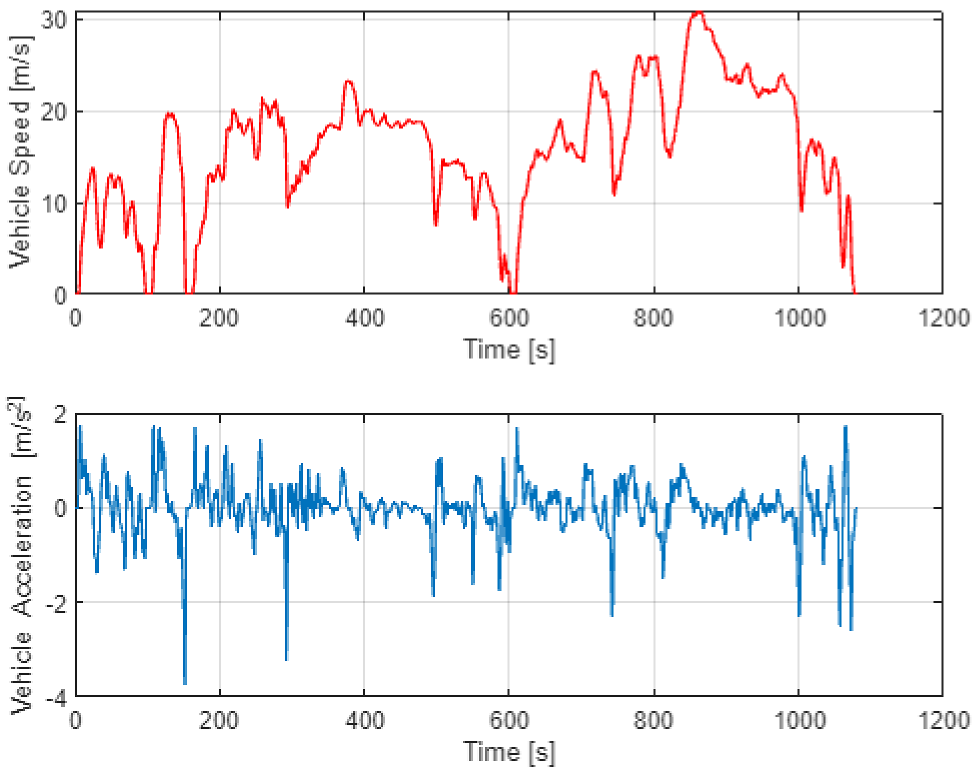
```
clear
close all
clc

% Load folders containing the vehilce model, mission and function
addpath("models");
addpath("data");
addpath("utilities");

% Load cycle data
mission = load("data\ARDC.mat");
time = mission.time_s;                  % [s]
vehSpd = mission.speed_km_h ./ 3.6;     % [m/s]
vehAcc = mission.acceleration_m_s2;     % [m/s^2]

% PLOT OF ARDC CYCLE (speed and acceleration)
% Vehicle Speed plot
nexttile(1);
plot(time, vehSpd, Color='r', LineWidth = 1);
xlabel('Time [s]');
ylabel('Vehicle Speed [m/s]');
grid on;

% Vehicle Acceleration plot
nexttile(2);
plot(time, vehAcc);
xlabel('Time [s]');
ylabel('Vehicle Acceleration [m/s^2]');
grid on;
```



```
veh = load("data\vehData.mat");

% Scale of vehicle data according to group parameters through the given scale
function
veh = scaleVehData(veh, 82000, 55000, 1260);
```

```
% Set up of the simulation loop

% Initialization of the state variable (SOC) and pre-allocation of variables
LHV = veh.eng.fuelLHV;
battEnergy = veh.batt.nomEnergy;
```

```
SOC = zeros(1, length(time));
engSpd = zeros(1, length(time));
engTrq = zeros(1, length(time));
fuelFlwRate = zeros(1, length(time));
unfeas = zeros(1, length(time));

SOC(1) = 0.6;           % the initial value of the battery SOC is assumed to be 60%
```

Equivalence factor calibration

The vehicle model considered in this project is a full hybrid. As it is not characterized by a plug-in capability, the battery cannot be charged from the grid and can only be recharged during vehicle operation. Therefore, to ensure proper functioning of the hybrid system, the battery's State Of Charge at the beginning and end of the mission must be equal. This condition defines the charge-sustaining mode of operation, which the controller implements through a parameter known as the *equivalence factor*. This factor is used in the computation of the equivalent fuel consumption and plays a critical role in the effectiveness of the energy management strategy. It can be tuned to influence how the controller balances the use of electrical and fuel energy, thus directly affecting the SOC evolution. In particular, a low equivalence factor assigns a poor cost to electrical energy usage, leading to excessive usage of the battery. This can lead to significant battery depletion, resulting in the loss of charge-sustaining behavior. On the other hand, a high equivalence factor penalizes the use of electrical energy, restraining battery usage and causing battery overcharging, again violating the charge-sustaining condition. Therefore, careful tuning of the equivalence factor is essential to ensure balanced energy usage and sustained system performance.

In this project, since the driving mission followed by the vehicle is known in advance, the equivalence factor can be accurately tuned using a bisection algorithm. This algorithm requires as input a function for which the root, i.e. the value that makes the function equal to zero, needs to be found. In the context of ECMS, the goal is to achieve the same State Of Charge at both the beginning and end of the driving cycle. Therefore, the function to be zeroed is the difference between the initial and final SOC values. To evaluate this function, multiple simulations are performed, each simulating the full driving cycle using the ECMS controller with a different value of the equivalence factor. After each simulation, the difference between the initial and final SOC is calculated. This allows the construction of a function that describes how the SOC deviation varies as a function of the equivalence factor. The bisection algorithm can then be applied to find the equivalence factor that results in a zero SOC deviation, thus ensuring charge-sustaining operation. The algorithm works by iteratively narrowing the interval within which the root is known to exist, until the midpoint leads a deviation close enough to zero. In this project, the acceptable convergence tolerance is set to 0.01. This means that, being the initial SOC equal to 60%, the final one must lie between 59% and 61%. It is important to notice that to ensure the effectiveness of the algorithm, the SOC deviation function must cross zero within the selected interval of equivalence factors. Therefore, a preliminary analysis is required to identify two values of the equivalence factor: one resulting in a positive SOC deviation, and the other in a negative one. Once these values are determined, the bisection algorithm can be applied with this interval as the starting range, ensuring that a root exists within it.

```
a = 2.5;                                % lower limit of equivalence factor's
starting interval
b = 2.9;                                % upper limit of equivalence factor's
starting interval
```

```
s_vector = linspace(a,b,25);            % 25 equi-spaced values of "s" in the range
between "a" and "b"
psi = zeros(1, length(s_vector));       % pre-allocating the State Of Charge
deviation

% Evaluation of SOC deviation as function of the equivalence factor (s)
s = s_vector;
for i = 1:length(s_vector)
    for n = 1:length(time)
        [engSpd(n), engTrq(n)] = ecmsControl(s(i), SOC(n), LHV, battEnergy,
vehSpd(n), vehAcc(n), veh);         % ecms controller function evaluating the
engine speed and torque according to SOC, engine state and cycle point
        [SOC(n+1), fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n),
[engSpd(n), engTrq(n)], [vehSpd(n), vehAcc(n)], veh);        % function evaluating
the new SOC and the fuel consumption according to the SOC, engine operating point
and cycle point
        prof(n) = structArray2struct(prof(n));          % conversion to scalar
structures containing arrays
    end
    psi(i) = SOC(end) - SOC(1);                         % evaluation of the SOC
deviation for each simulation with a given equivalence factor
end

% Bisection algorithm
c = b;                                                  % "c" acts like a tick: it
starts from the last value of the range and is then adjusted
SOC_dev = interp1(s_vector, psi, c, 'linear');          % using the interpolation
it is possible to obtain the SOC deviation for each desired equivalence factor
finSOC_values = 0.6 + SOC_dev;                          % final State of Charge,
considering initial SOC value and SOC deviation. It is used to observe how the
calibration works depending on the iterations.

while abs(SOC_dev) > 0.01                               % the SOC deviation
tolerance is 0.01. Meanwhile it keeps higher, "c" should be adjusted
    c = (a + b) / 2;                                    % take the interval
midpoint -> new equivalence factor
    SOC_dev = interp1(s_vector, psi, c, 'linear');      % obtain a new SOC
deviation with the new value of equivalence factor
    equivalenceFactor = c;                              % here the value of "c"
is saved as "equivalenceFactor" so that if abs(soc_dev) results lower than the
tolerance (0.01), it is possible to extract the correct value
    finSOC_values(end + 1) = 0.6 + SOC_dev;
    if SOC_dev > 0
        b = c;                                          % if the new SOC deviation
is positive, the computed equivalence factor becomes the new upper interval limit
    elseif SOC_dev < 0
        a = c;                                          % if the new SOC deviation
is negative, the computed equivalence factor becomes the new lower interval limit
    end
end
```

Run a simulation with the calibrated equivalence factor

Once the optimal equivalence factor is determined through the bisection algorithm, the driving mission is simulated again using the ECMS controller with this calibrated value. By applying this optimal equivalence factor, the vehicle can complete the drive cycle without excessive charging or discharging of the battery, thus maintaining a charge-sustaining operation, which is a realistic goal in hybrid control strategies.

```
s = equivalenceFactor;
for n = 1:length(time)
    [engSpd(n), engTrq(n)] = ecmsControl(s, SOC(n), LHV, battEnergy, vehSpd(n),
vehAcc(n), veh);                              % ecms controller function
evaluating the engine speed and torque according to SOC, engine state and cycle
point
    [SOC(n+1), fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n), [engSpd(n),
engTrq(n)], [vehSpd(n), vehAcc(n)], veh);    % function evaluating the new SOC and
the fuel consumption according to the SOC, engine operating point and cycle point
    prof(n) =
structArray2struct(prof(n));
                              % conversion to scalar structures containing arrays
end
```

Save results

```
%% Evaluating fuel consumption
fuelConsumption = cumtrapz(time, fuelFlwRate);                                %
integral of the fuel flow rate over the time [g]
fuelConsumption = fuelConsumption(length(time)) * 10^(-3);                   %
total fuel consumption is the last value of the cumulative integration [kg]

cycleDistance = trapz(time, vehSpd) / 10^3;                                  %
[km]
fuelEconomy = fuelConsumption / veh.eng.fuelDensity * 100/cycleDistance;     %
[L/100km]
finalSOC = SOC(length(time));                                                %
final SOC is the last value of the SOC profile [-]
eqFactor = equivalenceFactor;

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

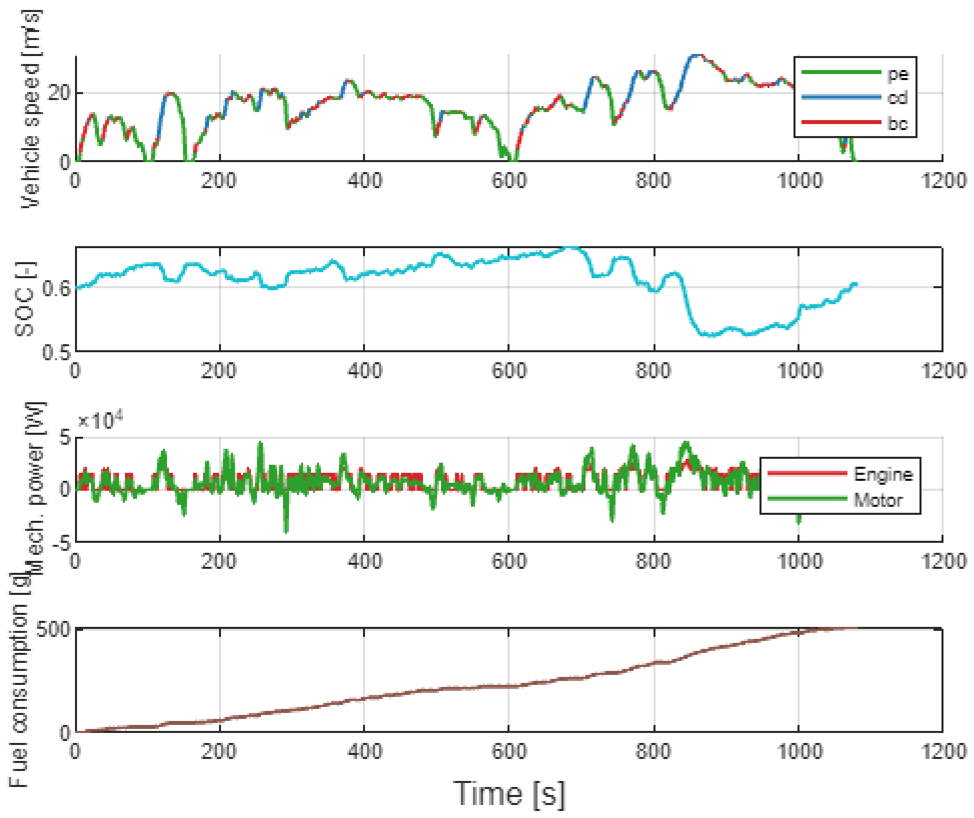
Controller analysis

The key results obtained are summarized below. They illustrate, in sequence, the drivetrain's operating modes according to the vehicle speed profile, the battery State Of Charge, the comparison between engine and motor power, and the fuel consumption, all presented as functions of time.

By analyzing the first plot, the different operating modes of the hybrid powertrain can be identified. The first mode is the pure electric mode (pe), in which the engine is off. In this condition, the battery discharges when the required power at the wheels is positive and charges when the motor operates in regenerative braking. The second mode is the charge-depleting mode (cd), where the engine is running but the battery continues to discharge because the power provided by the engine is lower than the power delivered by the battery to the electric motor. The last mode is the battery charging mode (bc), in which the engine is on and actively charging the battery.

The first analysis to be conducted in order to evaluate the proper functioning of the ECMS controller concerns the variation of the SOC. As previously explained, since the simulated vehicle is a hybrid, its operating mode should follow a charge-sustaining strategy. From the resulting SOC variation chart, it can be observed that the SOC at the end of the driving cycle is 60.5%, which falls within the defined tolerance range. This result confirms the validity of the controller and the correct implementation of both the control strategy and the bisection algorithm, which properly guarantee charge-sustaining operation.

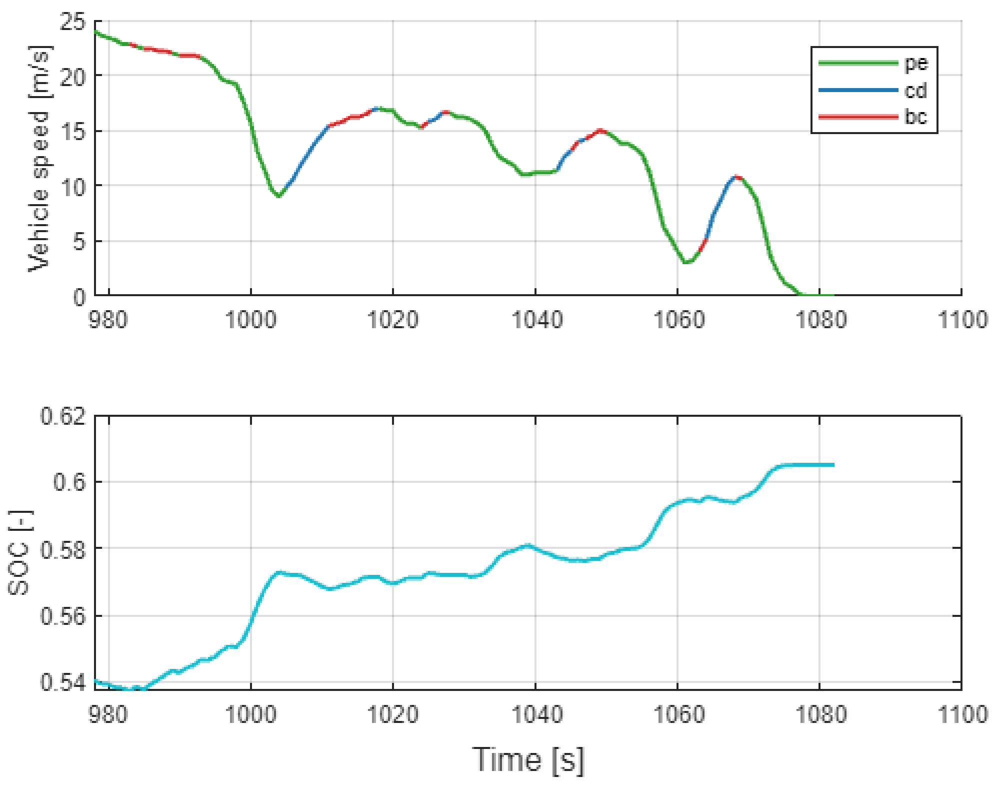
```
% MAIN PROFILES PLOT
[fig, t] = mainProfiles(prof);
```



Another important consideration, which aligns with the typical behavior of the ECMS, concerns the variation of the State Of Charge. As previously explained, the designed controller uses a bisection algorithm to determine the optimal equivalence factor for a given driving cycle. In practice, using this optimal equivalence factor in the real-time energy management strategy is equivalent to having a controller that can "predict the future". To better illustrate this concept, we can analyze the following chart, which shows the SOC variation over time and the drivetrain's operating modes based on the vehicle’s speed profile during the final portion of the driving cycle. This last segment is characterized by a significant number of deceleration phases, many of which involve relatively high deceleration values. Such conditions are ideal for regenerative braking, which allows recovering of mechanical energy at the wheels into electrical energy to recharge the battery. The “future knowledge” of the controller lies in its ability to anticipate this regenerative potential. Knowing that several opportunities to recharge the battery will occur later in the cycle, the controller can afford to draw more energy from the battery earlier on. This is clearly visible in the chart: the SOC drops below 60%, the valule around which the controller should implement the charge sustaining mode, and almost approaches 50%. This behavior demonstrates the earlier point: the system strategically depletes battery energy, confident that it can recover it during the deceleration phases. Indeed, once the minimum SOC is reached, the vehicle enters the deceleration-heavy zone, and the SOC begins to rise again, primarily due to regenerative braking. There is also a minor contribution from engine-based charging, but it is minimal. This is evident from the limited red segments on the speed profile, representing engine-powered charging, compared to the more extensive green sections that, in the analyzed time interval, indicate regenerative braking operation with the engine off.

However, this predictive capability of ECMS also highlights a key limitation: it is not optimized for real-time vehicle controllers. In real-world driving, the future driving profile is not known in advance, meaning that the optimal equivalence factor cannot be accurately determined without prior knowledge of the full driving cycle. As a result, true fuel consumption minimization cannot be guaranteed since no perfect tuning of the equivalence factor is possible. Moreover, the controller is highly sensitive to variations in this parameter. Even slight deviations from the optimal value can lead to suboptimal or even unfeasible behavior, such as a failure to maintain charge-sustaining operation. The reason for this behavior is discussed in greater detail in the final paragraph of the bisection algorithm analysis.

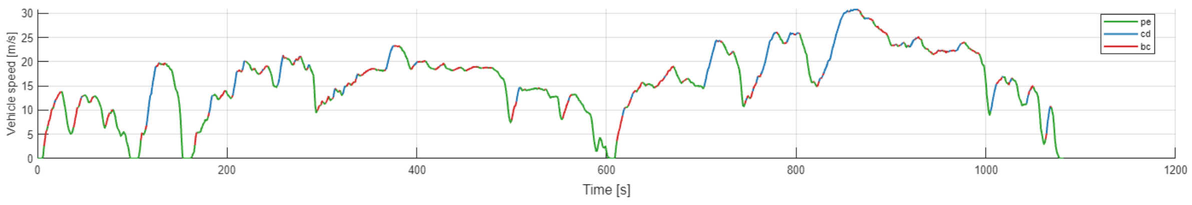
```matlab
% Plot for controller's "predictive behavior"
[fig1, t1] = futurePrediction(prof);
```



The powertrain's operating modes observed along the driving cycle can be analyzed in greater detail to better understand the controller's behavior and how it manages the vehicle's operating conditions, particularly in relation to speed and acceleration. From the chart, it is possible to deduce that the controller switches between different operating modes based on both the magnitude and sign of the acceleration. Specifically, during negative acceleration, i.e. braking conditions, the controller consistently operates the drivetrain in pure electric mode, ensuring the internal combustion engine is always turned off. This strategy allows the vehicle to exploit all the energy coming from the wheels to recharge the battery, minimizing energy losses through mechanical braking that would occur if the engine were on. Given that the architecture under study is a series hybrid, all energy produced by the engine is directed to the battery via the generator. Therefore, if the engine remains on during braking, two energy sources, the regenerative braking system and the engine, would simultaneously supply power to the battery. Since the battery has a limited charging capability, this could result in exceeding its input power limits. To prevent this, the power contribution from one of the sources must be reduced. However, since there is no clutch between the engine and the battery, the engine's energy path cannot be decoupled. The only alternative would be to dissipate the regenerative energy from the wheels through the mechanical brakes, a clearly suboptimal solution, as it wastes electric energy that could otherwise be recovered "for free" in favor of engine generated energy that consumes fuel. This is why the controller engages pure electric mode during braking: it prioritizes energy efficiency by eliminating unnecessary fuel consumption and maximizing regenerative braking potential.

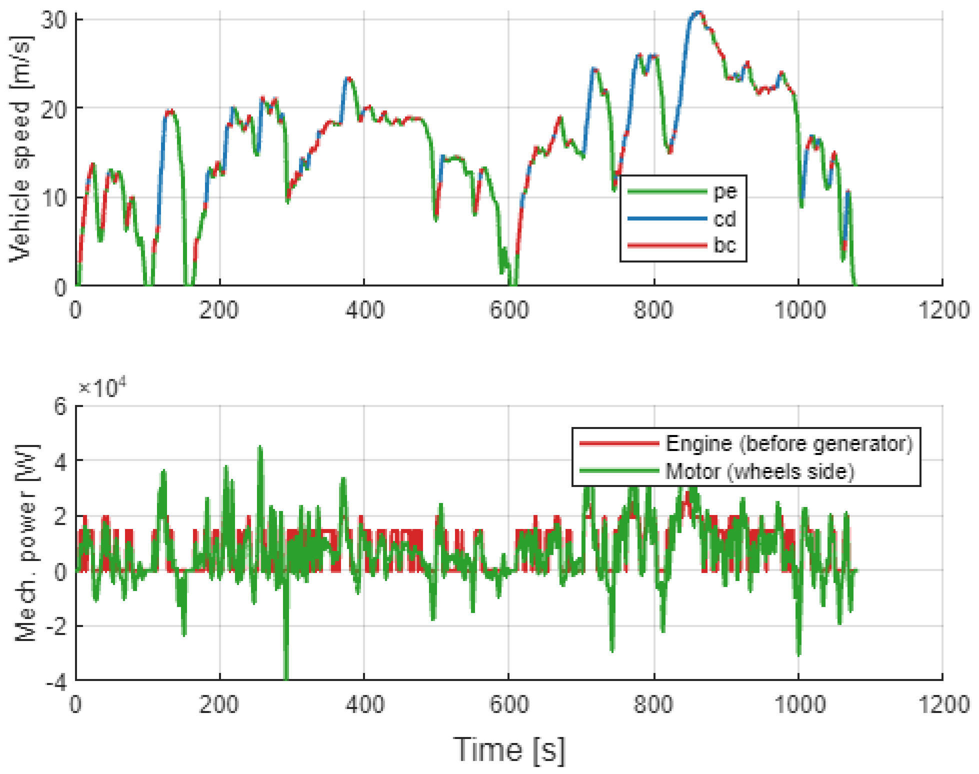
On the other hand, during vehicle acceleration phases, the engine is generally kept on. However, there are exceptions, for instance in the acceleration phase starting around T = 610 s, where the engine remains off at low speed, since the required power is small. When the engine is on, acceleration phases can follow two different operating modes: charge depleting and battery charging. The difference between these modes is related to the power balance between the engine and the electric motor. If the engine provides more power than the one required by the motor requires, the surplus charges the battery. On the other hand, if the engine provides less, the battery discharges. The controller typically activates the engine during acceleration for several reasons. One is related to maximizing regenerative braking energy, as previously discussed. Another reason concerns the physical limits of the battery: during high-power demand phases, such as strong acceleration, the electric motor may require more power than the one the battery can supply. In such cases, the engine must turn on to support the battery by supplying additional power, ensuring that the net power drawn from the battery does not exceed its limits. As a result, even if the control logic aims to keep the engine off to save fuel, it may be forced to turn it on to meet power demands. Therefore, it is better to keep the engine on during acceleration and, if charging is required, operate it at higher loads, allowing it to function within a more efficient operating range.

```
% Plot of drivetrain's operating modes
[fig2, t2] = operatingModes(prof);
```



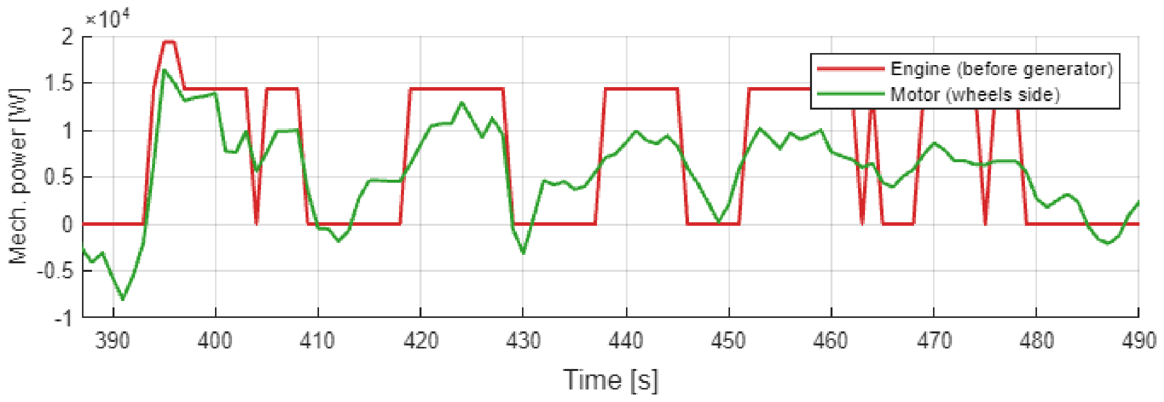
To better analyze the engine behavior throughout the driving cycle, a chart is used to show the time evolution of both mechanical power from the engine and from the electric motor. The data reveals that when the controller activates the engine, it maintains power at a mid level throughout the cycle. This approach keeps the engine operating under higher load conditions, which improves efficiency and reduces fuel consumption.

```
[fig3, t3] = engMotPwrComparison(prof);
```



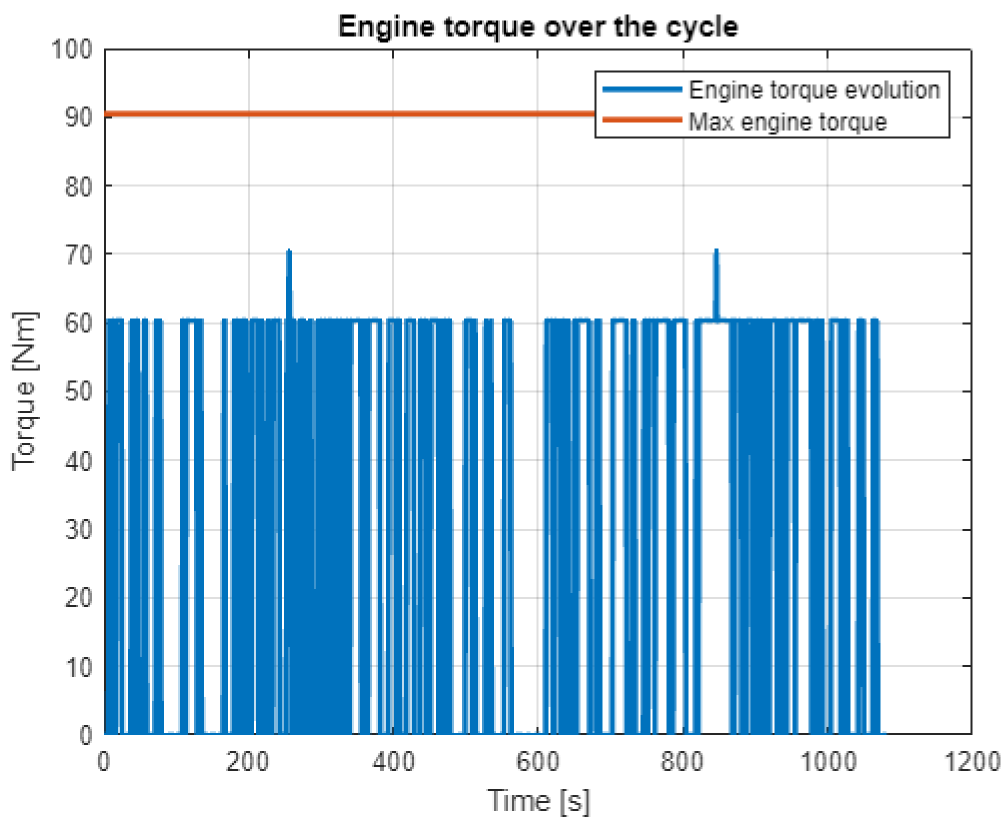
This strategy is further illustrated in the following chart, which provides a zoomed view of the mechanical power comparison around T = 400 s. This chart reveals an oscillating pattern in the engine's power between 0 and approximately 1.4 kW. Rather than continuously operating the engine at a lower power level, the controller adopts a staircase power strategy, alternating it between turning the engine off and operating it at a higher power level. This approach allows the engine to operate more efficiently by avoiding low load conditions, improving the energy management and fuel consumption.

```
[fig4, t4] = engMotPwrComparisonZoom(prof);
```



The consistently high load level can also be observed by analyzing the engine torque profile over the cycle. From the following chart, it is clear that whenever the engine is active, the torque is maintained at a mediumhigh level, corresponding to the area of the engine map where maximum efficiency is achieved. Additionally, the chart reveals that the controller selects between only two distinct torque levels.

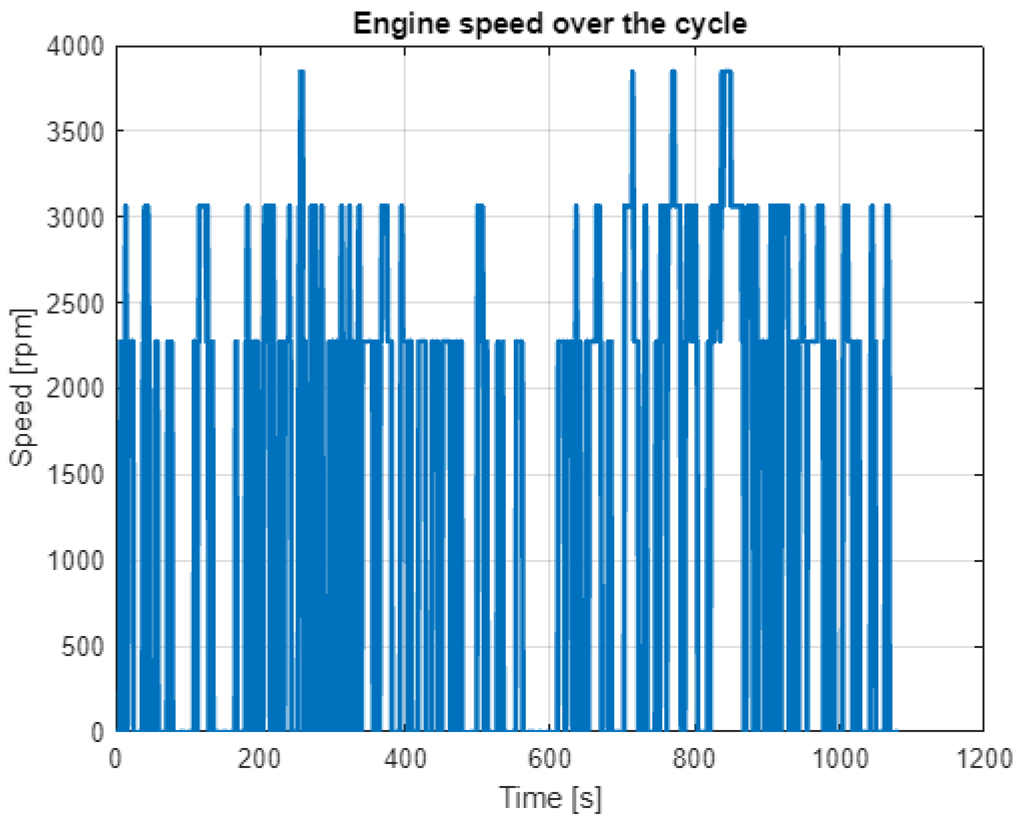
```
% Engine torque plot
figure;
plot(time, engTrq, LineWidth=2);
hold on;
plot([0, length(time)], [max(veh.eng.maxTrq.Values) max(veh.eng.maxTrq.Values)],
LineWidth=2);
xlabel('Time [s]')
ylabel('Torque [Nm]');
title('Engine torque over the cycle');
legend('Engine torque evolution', 'Max engine torque');
grid on;
```



Similarly to what is done for torque, the engine speed evolution can also be plotted to assess the number of speed levels selected by the controller. In this case, three distinct speed levels are used. The limited number of torque and speed levels is likely due to the coarse discretization of the speed-torque grid employed by the *ecmsController* function. In particular, the grid is defined with a resolution of 10x10, which results in only a few points lying within the high-efficiency region of the engine map. This restricts the set of feasible operating points available to the controller, explaining why only a few torque and speed levels are used. By increasing the grid resolution, more operating points within the efficiency zone become available, allowing the controller to select a greater variety of torque and speed levels. However, this comes at the cost of increased computational time.

```
% Engine speed plot
```

```
figure;
plot(time, engSpd * 30/pi, LineWidth=2);
xlabel('Time [s]')
ylabel('Speed [rpm]');
title('Engine speed over the cycle');
grid on;
```



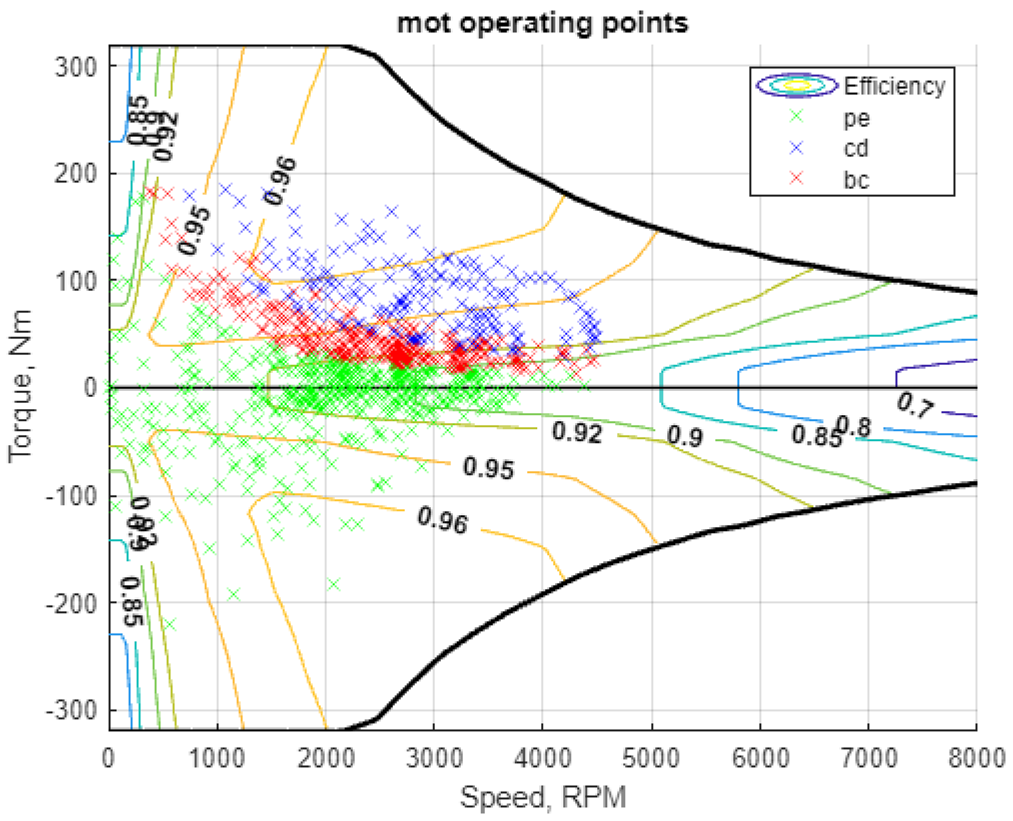
The following charts depict the motor, engine and generator operating points on their corresponding maps.

The motor map represents torque as a function of motor speed. As is typical for electric motors, this relationship features a constant torque profile at lower speeds and a constant power profile at higher speeds, which results in a hyperbolic torque curve. In addition to the torque profile, the map also includes efficiency isolines across the entire operating range of the electric motor. It can be noted that the high efficiency zone extends along a wide area of the motor operating zone. Notably, since this type of machine can generate both positive and negative torque, its operational area extends in both directions. Furthermore, the positive and negative torque profiles are almost symmetrical with respect to the x-axis.

It can be noticed that the electric motor operating points extend in a wide operating zone. This beacause the electric motor operating point cannot be directly controlled, since it depends on the vehicle speed and acceleration, and so on the driving mission, being the motor mechanically connected to the wheels.

The three operating modes are distinctly distributed across specific zones on the motor map, each corresponding to different power levels required at the wheels. These zones, green (pure electric), red (battery charging), and blue (charge depleting), follow a hyperbolic trend, as power levels on the torque-speed chart are characterized by hyperbolic curves. As expected, the pure electric mode is activated only at low power demands; accordingly, this zone lies below a low-power hyperbola. Furthermore, all operating points with negative power are associated with the pure electric mode, confirming that during deceleration the engine is turned off and all the braking energy is exploited to recharge the battery. At moderate power levels, just above the pure electric zone, the battery charging mode is engaged. In this mid-power region, the power demand at the wheels is modest, allowing the engine to operate efficiently and use excess power to recharge the battery. Conversely, in the high-power zone, the power required at the wheels exceeds what the engine can provide at its optimal operating point, making it impossible to charge the battery. As a result, the system operates in charge depleting mode.

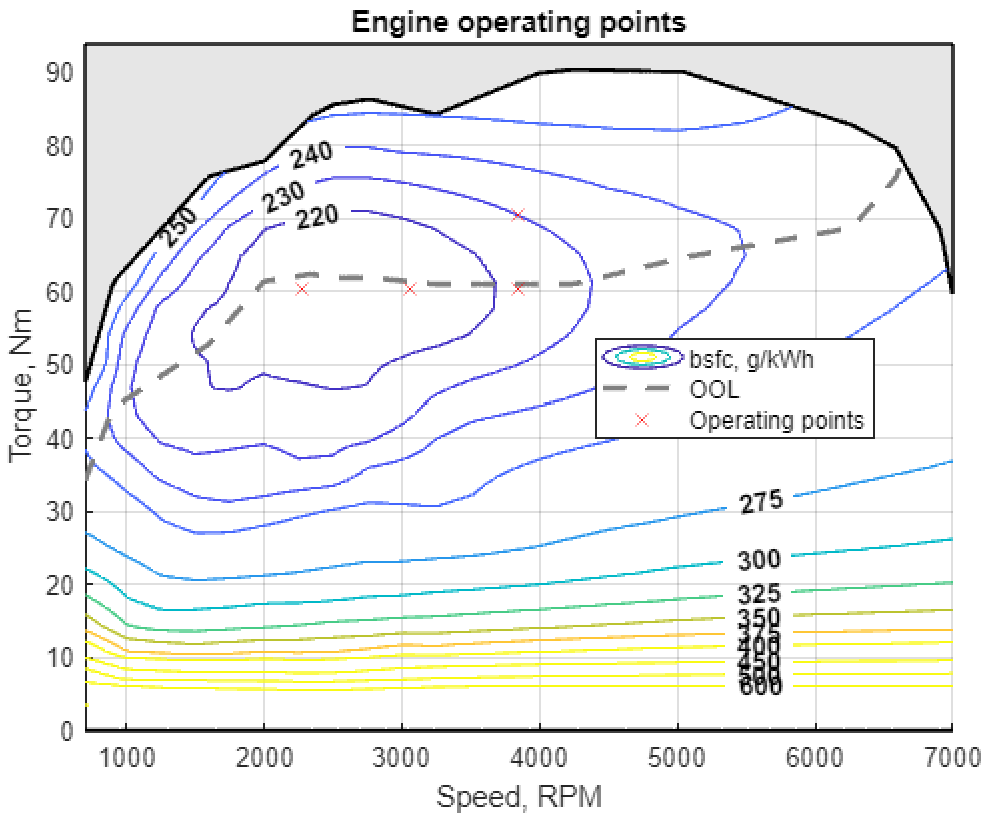
```
% Electric motor map operating points
emMapWithPF(veh.mot, prof, 'mot', 'all');
```



The engine map represents engine effciency, expressed as brake specific fuel consumption, as a function of engine speed and torque. It is important to note that, differently from the electric motor, the speed-torque relationship of an internal combustion engine has a more complex shape, with peak torque occurring at mid-range speeds. Additionally, the highest efficiency zone, corresponding to the minimum brake specific fuel consumption, covers a much smaller area and is located at low-mid speeds under relatively high loads. Alongside the *bsfc* map, the Optimal Operating Line (OOL) is also plotted, indicating the operating points with the lowest *bsfc* for each engine speed.

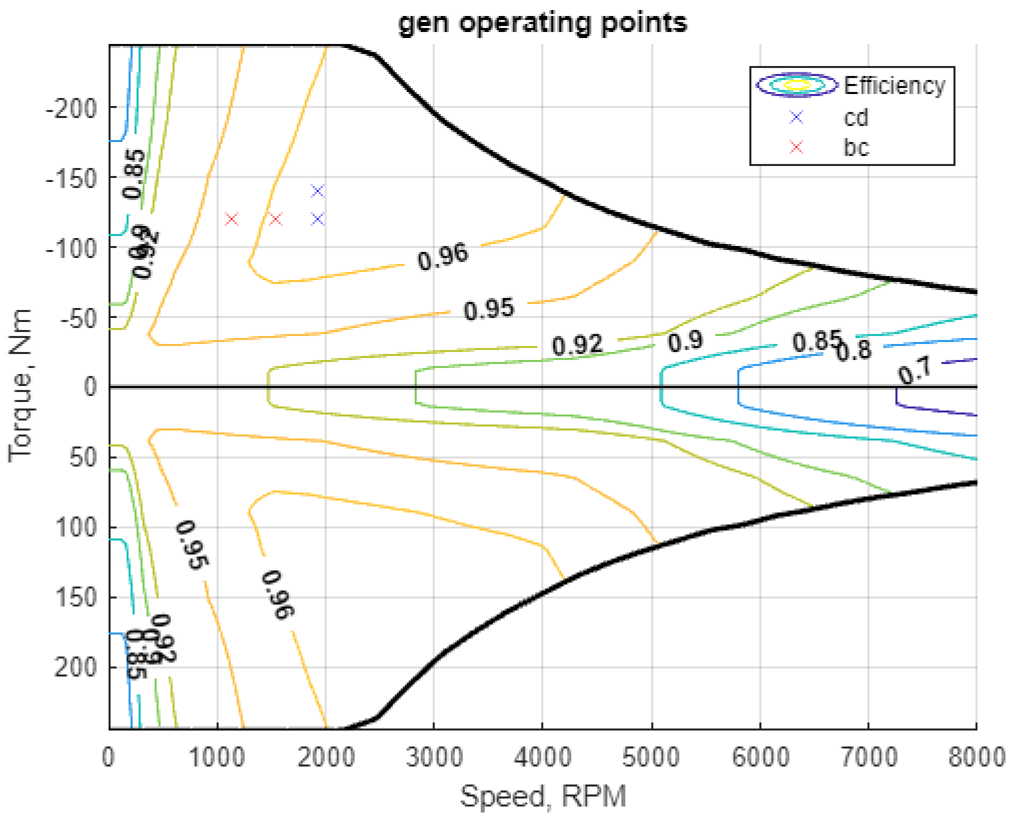
As previously discussed, three speed levels and two torque levels are present, combining to define four different operating points. An additional insight that can be drawn concerns the efficiency of these points. In particular, three out of the four lie close to the Optimal Operating Line, indicating that the engine operates within a high efficiency region. This confirms the effective performance of the ECMS controller in selecting energetically favorable operating points.

```
% Engine map operating points
engMapWithPF(veh.eng, prof, 'bsfc');
```



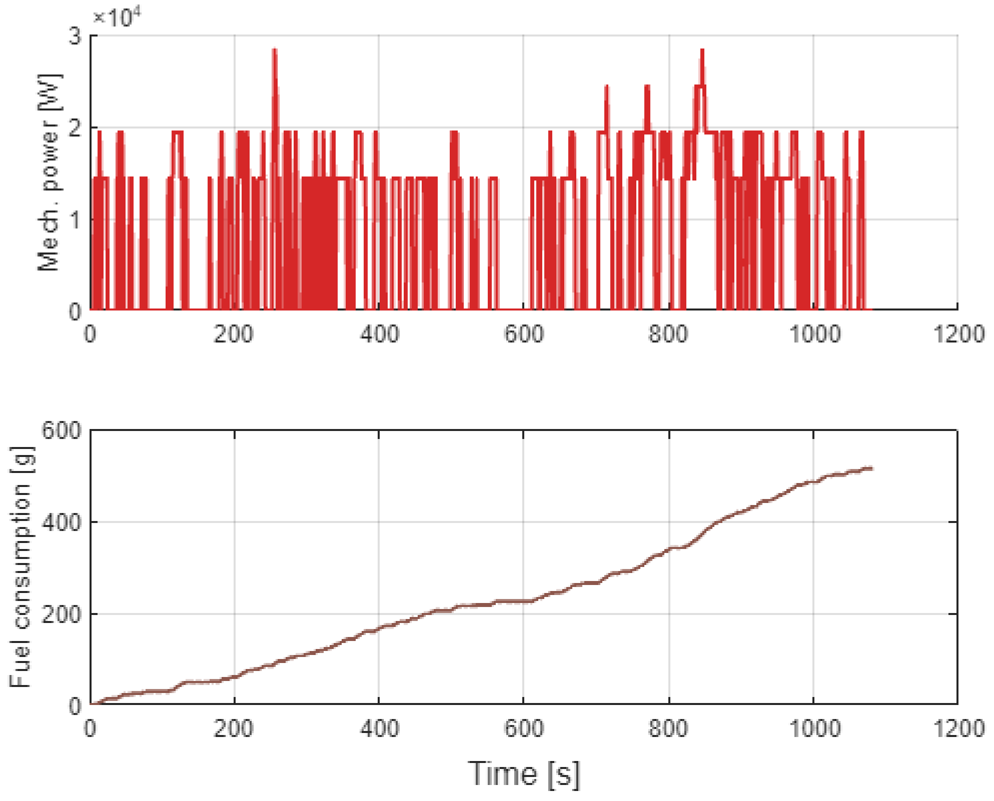
Similarly to the electric motor, the generator map can also be plotted, showing a comparable trend. Since the engine and generator are mechanically connected through an intermediate gear, the generator's speed and torque can be determined by simple mathematical relationships. Specifically, the absolute values of the generator's speed and torque are obtained by dividing and multiplying the engine speed and torque, respectively, by the transmission ratio. Given that the transmission ratio is 0.5, the generator speed is half of the engine speed, while the torque is doubled.

```
% Generator map operating points
emMapWithPF(veh.gen, prof, 'gen', 'all');
```



The final analysis focuses on the evolution of fuel consumption. As illustrated in the following charts, despite the varying power demands throughout the driving cycle, with lower power required in the initial phase and higher power towards the end, the engine power remains relatively evenly distributed across the entire cycle. Therefore, fuel consumption is characterized by an almost linear trend throughout the cycle.

```
% Fuel consumption-engine power relation
[fig5, t5] = fuelConsumpt(prof);
```

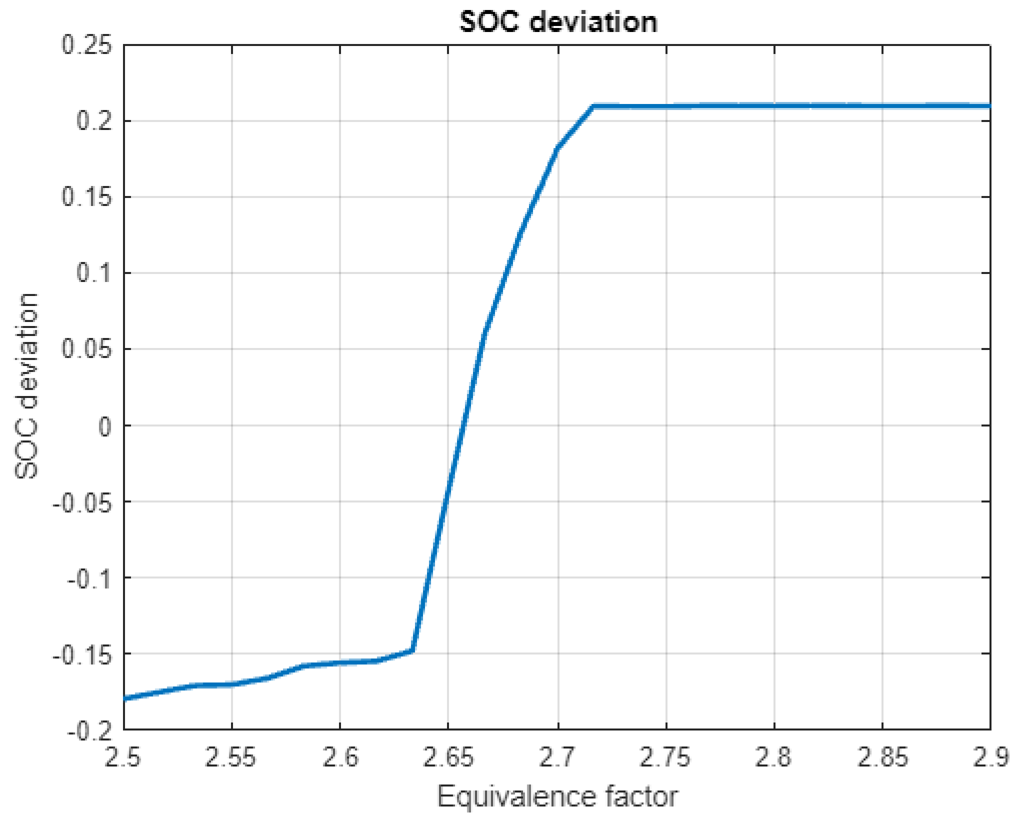


Once the analysis of the ECMS behavior has been completed, it is useful to evaluate the performance of the bisection algorithm. To support this evaluation, two charts are used: the first shows the difference between the final and initial SOC values as a function of the equivalence factor, while the second illustrates the SOC deviation across the iterations.

In the first chart, the necessary conditions for a correct application of the bisection algorithm can be verified. In particular, the SOC deviation function crosses the zero line within the initial chosen interval, confirming that the bisection algorithm can identify a root of the function.

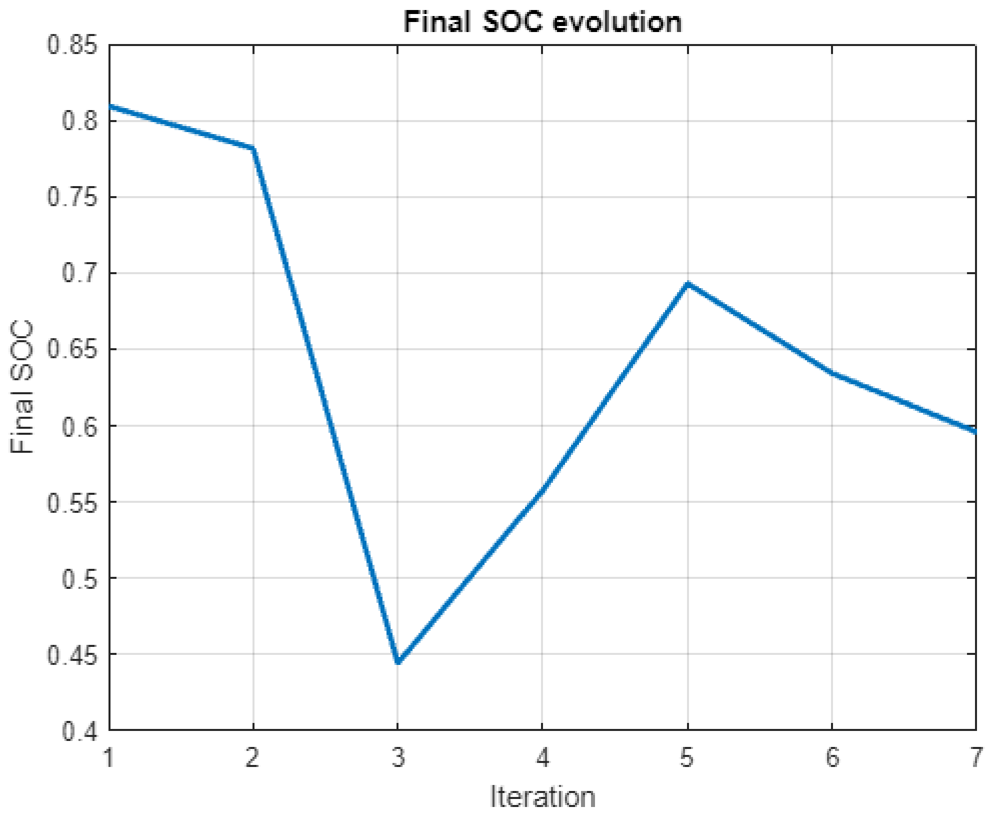
Another important observation concerns the controller's high sensitivity to the equivalence factor. As shown in the central region of the SOC deviation function, where the curve crosses zero, the slope is particularly steep. This indicates that even a small change in the equivalence factor can result in a significant variation in the final SOC value. Such sensitivity may lead to suboptimal or even unfeasible behavior, as discussed in a previous section.

```
figure;
plot(s_vector, psi, Linewidth=2);
grid on;
xlabel('Equivalence factor');
ylabel('SOC deviation');
title('SOC deviation');
```



On the other hand, the second chart is used to illustrates the convergence of the final SOC value toward the initial value as the number of bisection algorithm iterations increases. Initially, the error is quite large, with the first estimation resulting in a final SOC of 80%, which is significantly different from the desired value of 60%, the initial SOC. However, as the iterations progress, the algorithm refines its estimation at each step. After 7 iterations, the final SOC converges to the initial value within the acceptable tolerance, demonstrating the effectiveness of the bisection method.

```
figure;
plot(1:length(finSOC_values), finSOC_values, LineWidth=2);
grid on;
xlabel('Iteration');
ylabel('Final SOC');
title('Final SOC evolution');
```



The ECMS controller

The development of the energy management strategy is based on the ECMS (Equivalent Consumption Minimization Strategy), implemented through the corresponding ECMS control. The main objective of this control strategy is to determine the optimal engine operating point, defined by speed and torque, at each instant of the vehicle’s driving cycle. This operating point is primarily chosen to minimize fuel consumption. However, the strategy must operate within certain constraints imposed by the physical limits of the system. In particular, the battery's State of Charge must remain within a predefined range to prevent damage and extend battery life. Additionally, the power the battery can deliver or absorb is limited by its maximum and minimum allowable values, which must not be exceeded. As a result of these constraints, the selected engine operating point is not necessarily the one characterized by the absolute minimum fuel consumption. Instead, it corresponds to the point that offers the lowest fuel consumption among all those that satisfy the previously mentioned constraints.

The implementation of the control strategy within the controller is carried out in four main steps, as outlined below:

1. Initialization of all combinations of engine speed and torque
2. Evaluation of the fuel flow rate corresponding to each combination
3. Exclusion of unfeasible combinations due to system constraints
4. Selection of the engine speed and torque pair that minimizes the fuel flow rate among the feasible options

```
function [engSpd, engTrq] = ecmsControl(s, SOC, LHV, battEnergy, vehSpd, vehAcc,
veh)
% ECMS Controller - Selects optimal engine speed and torque to minimize equivalent
fuel consumption

% Inputs:
% s                   - Equivalence factor
% SOC                 - Battery state of charge
% LHV                 - Fuel lower heating value [J/kg]
% battEnergy          - Nominal energy stored in the battery [Wh]
% vehSpd              - Vehicle speed at the analyzed time step [m/s]
% vehAcc              - Vehicle acceleration at the analyzed time step [m/s^2]
% veh                 - Vehicle data

% Outputs:
% engSpd              - Optimal engine speed (rad/s)
% engTrq              - Optimal engine torque (Nm)
```

Step 1: initialization of all combinations of engine speed and torque

The first step in the ECMS control strategy involves defining a map that includes all possible combinations of engine speed and torque. To accomplish this, two vectors are initialized: one for engine speed values and one for engine torque values. The engine speed vector spans from idle speed to the maximum allowable speed, while the torque vector ranges from zero to the engine’s maximum torque. Both vectors contain equally spaced values, and their resolution can be arbitrarily defined. A higher vector resolution results in a denser grid of speed–torque combinations, which enhances the accuracy of the control strategy. This is because a finer grid increases the possibility that the control selects an operating point closer to engine's real optimal point, thus leading to lower fuel consumption. However, this increased accuracy leads to a greater computational demand, as the control unit must evaluate a larger number of combinations.

```
% Definition of engine speed and torque limits
% Speed
engSpeedMin = veh.eng.idleSpd;
engSpeedMax = veh.eng.maxSpd;
% Torque
engTorqueMin = 0;
engTorqueMax = max(veh.eng.maxTrq.Values);

% Definition of possible speed torque grid values
k = 10;                                                    % grid dimension
engSpeedSet = linspace(engSpeedMin, engSpeedMax, k-1);
engSpeedSet = [0, engSpeedSet];                            % adding zero speed value
(engine off)
engTorqueSet = linspace(engTorqueMin, engTorqueMax, k);

SOC_next = zeros(length(engSpeedSet), length(engTorqueSet));
fuelFlwRate = zeros(length(engSpeedSet), length(engTorqueSet));
fuelFlwRateEq = zeros(length(engSpeedSet), length(engTorqueSet));
unfeas = zeros(length(engSpeedSet), length(engTorqueSet));
```

```
penalty = 1e6;
penaltySOC = 1e3;
```

Step 2: evaluation of the fuel flow rate corresponding to each combination

Once the grid of possible engine speed and torque combinations has been established, the next step in determining the optimal engine operating point is the evaluation of the fuel consumption for each combination. To accomplish this, the *hev_model* function is employed. This function calculates the fuel flow rate and the battery's SOC based on the selected engine operating point and the drivetrain model. In addition, it also identifies the unfeasiblities, i.e. the combinations that result in conditions that can never occur in practice. These unfeasible points are essential for the following step, as they are excluded from the optimization process.

```
for i = 1:length(engSpeedSet)
    for j = 1:length(engTorqueSet)
        [SOC_next(i,j), fuelFlwRate(i,j), unfeas(i,j), prof] = hev_model(SOC,
[engSpeedSet(i), engTorqueSet(j)], [vehSpd, vehAcc], veh);        % function
evaluating the new SOC and the fuel consumption according to the SOC, engine
operating point and cycle point
```

Step 3: exclusion of unfeasible combinations due to system constraints

As mentioned before, due to the physical limits of the system, not all the combinations of engine speed and torque represent feasible operating points. Therefore, to ensure a proper functioning of the controller, these unfeasible points must be excluded from the set of valid combinations. This is accomplished by exploiting the unfeasibility evaluation provided by the *hev_model* function. In particular, whenever an unfeasibility is detected, the controller adds a penalty to the fuel flow rate value corresponding to the engine speed and torque combination for which the unfeasibility is occurring. This approach ensures that such points are effectively excluded from the optimization process, as the large penalty prevents them from being selected as the minimum fuel consumption solution.

Different types of unfeasibilities can arise. Some are related to the operational limits of the engine, motor, and generator, meaning that certain speed–torque combinations fall outside their respective performance maps. Another type involves the battery's power limits; if these limits are exceeded, the corresponding operating point is set as unfeasible. Moreover, there is a specific unfeasibility related to the battery's State of Charge. Whenever the SOC exceeds predefined limits, the controller must detect and prevent this condition. For this project, the SOC constraints are defined as follows:

- SOCmax = 80%
- SOCmin = 40%

To be noticed that the penalty associated with SOC limit violations is lower, as these do not cause unfeasibility but are instead discouraged due to their negative impact on battery health. In certain cases, the fuel consumption matrix may contain both unfeasible points and points violating SOC constraints. In such situations, it is preferable to select the operating point that exceeds SOC limits rather than one that is unfeasible. For this reason, the two penalties are assigned different weights.

It is also important to note that the *hev_model* function only evaluates the unfeasibilities related to the physical limits of the drivetrain components and does not account for SOC constraints. Therefore, an additional check must be implemented to ensure that the SOC remains within the specified range for each speed–torque combination.

```
        % Exclusion of unfeasible controls by adding a penalty
        if SOC_next(i,j) > 0.8 || SOC_next(i,j) < 0.4             % considering SOC
constraints
           fuelFlwRateEq(i,j) = penaltySOC;
        else
           fuelFlwRateEq(i,j) = fuelFlwRate(i,j) - s * (battEnergy*3600) / (LHV/
1000) * (SOC_next(i,j) - SOC) + unfeas(i,j) * penalty;      % considering unfeas
as penalty
        end
    end
end
```

Step 4: selection of the engine speed and torque pair that minimizes the fuel flow rate among the feasible options

Once the fuel flow rate has been calculated for each possible combination of engine speed and torque, including any penalties applied for unfeasibilities, the optimal engine operating point can be determined. This is done by identifying the minimum fuel consumption among all feasible combinations and selecting the corresponding engine speed and torque values giving this minimum.

```
% Evaluation of minimum equivalent fuel flow rate
[~, minFuelFlwRateEq_index] = min(fuelFlwRateEq(:));
[rowminFuelFlwRateEq, colminFuelFlwRateEq] = ind2sub(size(fuelFlwRateEq),
minFuelFlwRateEq_index);

% Evaluation of engine speed and torque corresponding to min equivalent fuel flow
rate
engSpd = engSpeedSet(1, rowminFuelFlwRateEq);
engTrq = engTorqueSet(1, colminFuelFlwRateEq);

end
```