

# Assignment #3: A-ECMS

## Table of Contents

Group information.....	1
Load the cycle and vehicle data.....	1
Test the A-ECMS on the test cycles.....	5
Results Analysis.....	7
AMDC.....	8
AUDC.....	11
WLTP.....	14
Turin Test.....	16
The A-ECMS controller.....	21
Step 1: adaptation of the equivalence factor.....	22
Step 2: initialization of all combinations of engine speed and torque.....	22
Step 3: evaluation of the fuel flow rate corresponding to each combination.....	23
Step 4: exclusion of unfeasible combinations due to system constraints.....	23
Step 5: selection of the engine speed and torque pair that minimizes the fuel flow rate among the feasible options.....	24

## Group information

Group number: 33

Students:

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

## Load the cycle and vehicle data

The simulation conducted in this project focuses on implementing an Adaptive-ECMS and testing it across four distinct driving cycles: WLTP, AMDC, AUDC, and the Turin Test. These cycles are loaded and initialized in the following section, with their respective variables for mission, time, vehicle speed, and acceleration. To visually inspect and compare the nature of each cycle, both the vehicle speed and acceleration are plotted over time for each cycle. These plots help to understand the dynamic behavior of each test case and anticipate how challenging each driving pattern might be for the Adaptive-ECMS controller.

```
clear
close all
clc

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

```

mission_WLTP = load("data\WLTP.mat");
time_WLTP = mission_WLTP.time_s; % [s]
vehSpd_WLTP = mission_WLTP.speed_km_h ./ 3.6; % [m/s]
vehAcc_WLTP = mission_WLTP.acceleration_m_s2; % [m/s^2]

mission_AMDC = load("data\AMDC.mat");
time_AMDC = mission_AMDC.time_s; % [s]
vehSpd_AMDC = mission_AMDC.speed_km_h ./ 3.6; % [m/s]
vehAcc_AMDC = mission_AMDC.acceleration_m_s2; % [m/s^2]

mission_AUDC = load("data\AUDC.mat");
time_singleAUDC = mission_AUDC.time_s; % [s]
vehSpd_singleAUDC = mission_AUDC.speed_km_h ./ 3.6; % [m/s]
vehAcc_singleAUDC = mission_AUDC.acceleration_m_s2; % [m/s^2]
time_AUDC = 0:length(time_singleAUDC)*5-1;
vehSpd_AUDC = repmat(vehSpd_singleAUDC, 1, 5); % Row vector with five
times the speed profile
vehAcc_AUDC = repmat(vehAcc_singleAUDC, 1, 5); % Row vector with five
times the acceleration profile

mission_TurinTest = load("data\TurinTest.mat");
time_TurinTest = mission_TurinTest.time_s; % [s]
vehSpd_TurinTest = mission_TurinTest.speed_km_h ./ 3.6; % [m/s]
vehAcc_TurinTest = mission_TurinTest.acceleration_m_s2; % [m/s^2]

% PLOTS OF CYCLE'S SPEED AND ACCELERATION
tiledlayout(5, 2);

% WLTP
% Vehicle Speed plot WLTP
nexttile(3);
plot(time_WLTP, vehSpd_WLTP, Color='r', LineWidth = 1);
xlabel('Time [s]');
ylabel('Speed [m/s]');
title('WLTP CYCLE');
grid on;

% Vehicle Acceleration plot WLTP
nexttile(4);
plot(time_WLTP, vehAcc_WLTP);
xlabel('Time [s]');
ylabel('Acceleration [m/s^2]');
title('WLTP CYCLE');
grid on;

% AMDC
% Vehicle Speed plot AMDC
nexttile(5);

```

```

plot(time_AMDC, vehSpd_AMDC, Color='r', LineWidth = 1);
xlabel('Time [s]');
ylabel('Speed [m/s]');
title('AMDC CYCLE');
grid on;

% Vehicle Acceleration plot AMDC
nexttile(6);
plot(time_AMDC, vehAcc_AMDC);
xlabel('Time [s]');
ylabel('Acceleration [m/s^2]');
title('AMDC CYCLE');
grid on;

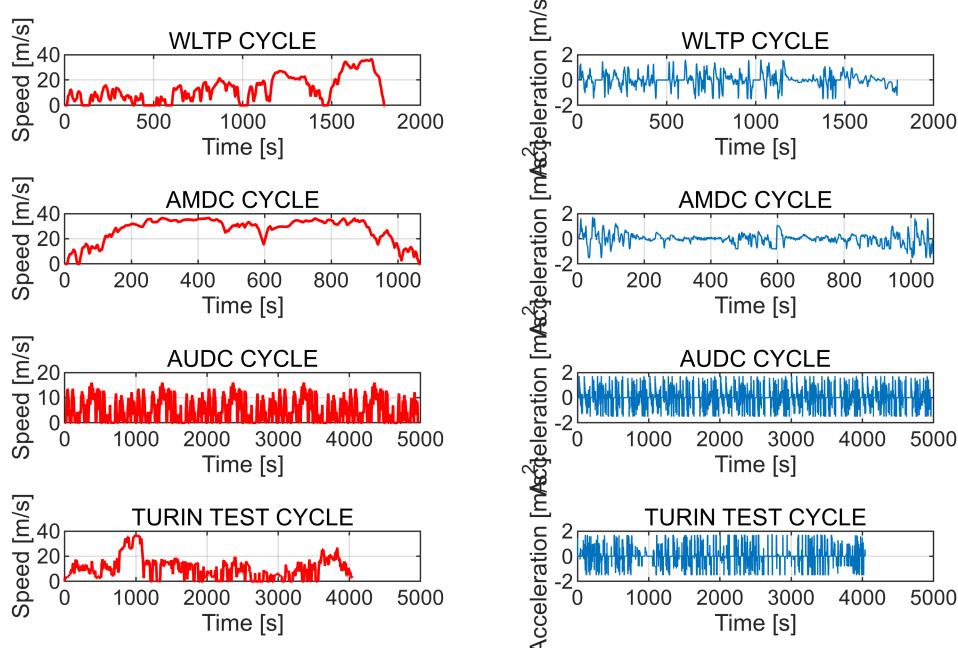
% AUDC
% Vehicle Speed plot AUDC
nexttile(7);
plot(time_AUDC, vehSpd_AUDC, Color='r', LineWidth = 1);
xlabel('Time [s]');
ylabel('Speed [m/s]');
title('AUDC CYCLE');
grid on;

% Vehicle Acceleration plot AUDC
nexttile(8);
plot(time_AUDC, vehAcc_AUDC);
xlabel('Time [s]');
ylabel('Acceleration [m/s^2]');
title('AUDC CYCLE');
grid on;

% TurinTest
% Vehicle Speed plot TurinTest
nexttile(9);
plot(time_TurinTest, vehSpd_TurinTest, Color='r', LineWidth = 1);
xlabel('Time [s]');
ylabel('Speed [m/s]');
title('TURIN TEST CYCLE');
grid on;

% Vehicle Acceleration plot TurinTest
nexttile(10);
plot(time_TurinTest, vehAcc_TurinTest);
xlabel('Time [s]');
ylabel('Acceleration [m/s^2]');
title('TURIN TEST CYCLE');
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
LHV = veh.eng.fuelLHV;
battEnergy = veh.batt.nomEnergy;

% Initialization of the state variable (SOC) and pre-allocation of variables for
% WLTP
SOC_WLTP = zeros(1, length(time_WLTP));
engSpd_WLTP = zeros(1, length(time_WLTP));
engTrq_WLTP = zeros(1, length(time_WLTP));
fuelFlwRate_WLTP = zeros(1, length(time_WLTP));
unfeas_WLTP = zeros(1, length(time_WLTP));

% Initialization of the state variable (SOC) and pre-allocation of variables for
% AMDC
SOC_AMDC = zeros(1, length(time_AMDC));
engSpd_AMDC = zeros(1, length(time_AMDC));
engTrq_AMDC = zeros(1, length(time_AMDC));
fuelFlwRate_AMDC = zeros(1, length(time_AMDC));
unfeas_AMDC = zeros(1, length(time_AMDC));
```

```

% Initialization of the state variable (SOC) and pre-allocation of variables for
AUDC
SOC_AUDC = zeros(1, length(time_AUDC));
engSpd_AUDC = zeros(1, length(time_AUDC));
engTrq_AUDC = zeros(1, length(time_AUDC));
fuelFlwRate_AUDC = zeros(1, length(time_AUDC));
unfeas_AUDC = zeros(1, length(time_AUDC));

% Initialization of the state variable (SOC) and pre-allocation of variables for
TurinTest
SOC_TurinTest = zeros(1, length(time_TurinTest));
engSpd_TurinTest = zeros(1, length(time_TurinTest));
engTrq_TurinTest = zeros(1, length(time_TurinTest));
fuelFlwRate_TurinTest = zeros(1, length(time_TurinTest));
unfeas_TurinTest = zeros(1, length(time_TurinTest));

Tupdate = 60;           % update time of the equivalence factor [s]

kp = 0.331575;         % kp value set equal to the average between the optimal ones of
the analyzed cycles

```

## Test the A-ECMS on the test cycles

In this section, as previously mentioned, the Adaptive-ECMS is tested on four different driving cycles: AMDC, AUDC, WLTP, and the Turin Test. The choice of these specific cycles is motivated by their different characteristics. Indeed, to properly evaluate the controller's performance and functionality, it is essential to test it under a variety of conditions in terms of speed and acceleration. This ensures a comprehensive assessment across the full range of operating scenarios that the HEV may encounter in real world applications. Each cycle represents a different driving context: the AMDC simulates highway conditions, the AUDC focuses exclusively on urban driving, while the WLTP combines both urban and extra-urban segments. Lastly, the Turin Test is representative of real-world city driving, characterized predominantly by low speeds and frequent high acceleration phases, with a smaller portion involving also higher speed periods. These distinguishing features of the driving cycles are clearly illustrated in the speed and acceleration profiles presented in the previous section.

```

s0 = 2.65625;           % the initial value of the equivalence factor is considered
to be equal to the one obtained by tuning the ECMS on the ARDC for all the
simulated cycles

% WLTP
SOC_WLTP(1) = 0.6;      % the initial value of the battery SOC is assumed to be 60%
s_WLTP(1) = s0;          %s_n-1
s_WLTP(2) = s0;          %s_n
tc = 0;                  % setting the counter
for k = 1:length(time_WLTP)
    [engSpd_WLTP(k), engTrq_WLTP(k), sn, tc] = adaptiveEcmsControl(SOC_WLTP(k),
    LHV, battEnergy, vehSpd_WLTP(k), vehAcc_WLTP(k), veh, s_WLTP(end), s_WLTP(max(1,

```

```

end-Tupdate)), tc, Tupdate, kp, SOC_WLTP(1)); % ecms controller function evaluating
the engine speed and torque according to SOC, engine state and cycle point
    [SOC_WLTP(k+1), fuelFlwRate_WLTP(k), unfeas_WLTP(k), prof_WLTP(k)] =
hev_model(SOC_WLTP(k), [engSpd_WLTP(k), engTrq_WLTP(k)], [vehSpd_WLTP(k),
vehAcc_WLTP(k)], veh); % function evaluating the new SOC and the fuel
consumption according to the SOC, engine operating point and cycle point
    prof_WLTP(k) = structArray2struct(prof_WLTP(k)); % conversion to scalar
structures containing arrays
    s_WLTP(k) = sn;
    tc = tc + 1;
end
finalSOC_WLTP = SOC_WLTP(end);

% AMDC
SOC_AMDC(1) = 0.6; % the initial value of the battery SOC is assumed to be 60%
s_AMDC(1) = s0; %s_n-1
s_AMDC(2) = s0; %s_n
tc = 0; % resetting the counter
for k = 1:length(time_AMDC)
    [engSpd_AMDC(k), engTrq_AMDC(k), sn, tc] = adaptiveEcmsControl(SOC_AMDC(k),
LHV, battEnergy, vehSpd_AMDC(k), vehAcc_AMDC(k), veh, s_AMDC(end), s_AMDC(max(1,
end-Tupdate)), tc, Tupdate, kp, SOC_AMDC(1)); % ecms controller function evaluating
the engine speed and torque according to SOC, engine state and cycle point
    [SOC_AMDC(k+1), fuelFlwRate_AMDC(k), unfeas_AMDC(k), prof_AMDC(k)] =
hev_model(SOC_AMDC(k), [engSpd_AMDC(k), engTrq_AMDC(k)], [vehSpd_AMDC(k),
vehAcc_AMDC(k)], veh); % function evaluating the new SOC and the fuel
consumption according to the SOC, engine operating point and cycle point
    prof_AMDC(k) = structArray2struct(prof_AMDC(k)); % conversion to scalar
structures containing arrays
    s_AMDC(k) = sn;
    tc = tc + 1;
end
finalSOC_AMDC = SOC_AMDC(end);

% AUDC
SOC_AUDC(1) = 0.6; % the initial value of the battery SOC is assumed to be 60%
s_AUDC(1) = s0; %s_n-1
s_AUDC(2) = s0; %s_n
tc = 0; % resetting the counter
for k = 1:length(time_AUDC)
    [engSpd_AUDC(k), engTrq_AUDC(k), sn, tc] = adaptiveEcmsControl(SOC_AUDC(k),
LHV, battEnergy, vehSpd_AUDC(k), vehAcc_AUDC(k), veh, s_AUDC(end), s_AUDC(max(1,
end-Tupdate)), tc, Tupdate, kp, SOC_AUDC(1)); % ecms controller function evaluating
the engine speed and torque according to SOC, engine state and cycle point
    [SOC_AUDC(k+1), fuelFlwRate_AUDC(k), unfeas_AUDC(k), prof_AUDC(k)] =
hev_model(SOC_AUDC(k), [engSpd_AUDC(k), engTrq_AUDC(k)], [vehSpd_AUDC(k),
vehAcc_AUDC(k)], veh); % function evaluating the new SOC and the fuel
consumption according to the SOC, engine operating point and cycle point
    prof_AUDC(k) = structArray2struct(prof_AUDC(k)); % conversion to scalar
structures containing arrays

```

```

s_AUDC(k) = sn;
tc = tc + 1;
end
finalSOC_AUDC = SOC_AUDC(end);

% TurinTest
SOC_TurinTest(1) = 0.6;      % the initial value of the battery SOC is assumed to be
60%
s_TurinTest(1) = s0;          %s_n-1
s_TurinTest(2) = s0;          %s_n
tc = 0;                      % resetting the counter
for k = 1:length(time_TurinTest)
    [engSpd_TurinTest(k), engTrq_TurinTest(k), sn, tc] =
adaptiveEcmsControl(SOC_TurinTest(k), LHV, battEnergy, vehSpd_TurinTest(k),
vehAcc_TurinTest(k), veh, s_TurinTest(end), s_TurinTest(max(1, end-Tupdate)), tc,
Tupdate, kp, SOC_TurinTest(1)); % ecms controller function evaluating the engine
speed and torque according to SOC, engine state and cycle point
    [SOC_TurinTest(k+1), fuelFlwRate_TurinTest(k), unfeas_TurinTest(k),
prof_TurinTest(k)] = hev_model(SOC_TurinTest(k), [engSpd_TurinTest(k),
engTrq_TurinTest(k)], [vehSpd_TurinTest(k), vehAcc_TurinTest(k)], veh);      %
function evaluating the new SOC and the fuel consumption according to the SOC,
engine operating point and cycle point
    prof_TurinTest(k) = structArray2struct(prof_TurinTest(k)); % conversion to
scalar structures containing arrays
    s_TurinTest(k) = sn;
    tc = tc + 1;
end
finalSOC_TurinTest = SOC_TurinTest(end);

```

## Results Analysis

The main profiles obtained for each cycle are summarized in the charts 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.

The first plot of the main profiles shows the different operating modes of the hybrid powertrain. 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.

In addition to the main profiles, a second group of charts is presented. These plots show the variation of the equivalence factor and its relationship with the SOC over time, offering further insight into the control strategy and energy management throughout each driving cycle.

It is highlighted that prior to this simulation, a series of analyses were conducted, each aimed at determining the optimal kp for a specific driving cycle. The goal was to identify the kp value that ensures charge-sustaining

operation, i.e., maintaining the SOC at its initial value of 60% by the end of the cycle, which is the typical operating condition for a HEV. A trial-and-error procedure was applied: for each driving cycle, multiple simulations were run, adjusting the kp value until the one that returned a final SOC equal to the initial one was found. Once the optimal kp values for all four cycles were identified, their average was calculated and used as the proportional gain in the final simulation in order to better understand the influence of kp on powertrain behavior. The results of this simulation are presented in the following section.

The following bullet list reports the optimal tuned values of kp for the different analyzed cycles, as well as the average value:

- AMDC: kp\_optimal = 0.0978
- AUDC: kp\_optimal = 0.3
- WLTP: kp\_optimal = 0.796
- Turin Test: kp\_optimal = 0.1325
- Average: kp\_avg = 0.331575

## AMDC

In this section the results of the AMDC are analyzed. The State Of Charge plot clearly shows a value of the final SOC largely distant from the desired one. This behavior implies that the equivalence factor calibration is not able to properly react to the SOC variations throughout the cycle. The reason of this behavior is related to the nature of the AMDC cycle. As can be seen by the vehicle speed profile, the vehicle operates most of the time in highway, therefore at high speed. This condition leads to a large power demand for almost the whole duration of the cycle. By better analyzing the main profiles charts, an almost perfect charge sustaining phase is visible at the beginning of the mission, when the speed is small yet and consequently also the power demand. Being this operating condition so close to the desired one, the equivalence factor remains almost constant in this phase, as shown by the first three values of the equivalence factor plot.

However, as the simulation progresses, the power demand increases, and charge-sustaining operation can no longer be maintained. This condition becomes evident between t=110 seconds and t=250 seconds, where the powertrain operates predominantly in charge-depleting mode, as can be observed by the blue portion of the vehicle speed profile. This condition corresponds to a significant drop in the battery SOC, which reaches the lower limit of 40%. To counteract this rapid decrease in SOC, the A-ECMS suddenly increases the equivalence factor. By doing so, a higher cost is assigned to the use of electrical energy, thereby discouraging battery usage and prioritizing power from the internal combustion engine. This condition of increasing the equivalence factor persists throughout the most of the remaining part of the mission. Nevertheless, the increase in the equivalence factor is not fast enough to allow the SOC to return to its initial value of 60% before the end of the cycle. As a result, during the entire high-speed phase, the battery SOC remains at its lower limit.

In contrast, the final phase of the simulation presents an opposite behavior. This segment is characterized by strong deceleration, leading to a rapid drop in speed over a short period. Due to the continuous rise of the equivalence factor during the previous phase, it reaches a very high value by the time this deceleration phase begins. However, such driving conditions offer a high potential for regenerative braking. Under these circumstances, a low equivalence factor would normally be preferable to encourage the use of electrical energy which can be recovered through the regenerative braking. Instead, the high equivalence factor makes the use of electrical energy less favorable, which in turn causes the SOC to rise rapidly. This is a result of the large

amount of energy recovered through regenerative braking combined with the fact that the controller prefers to minimize battery usage because it is considered not beneficial. Moreover, since this final deceleration phase is very short in duration and the controller updates the equivalence factor only every 60 seconds, it does not have sufficient time to adjust the equivalence factor to match the new operating condition. In fact, as shown in the plot of the equivalence factor evolution, the controller begins to decrease the equivalence factor, but only once, right before the end of the mission.

Considering the characteristics of the cycle, to accomplish with these conditions, two possible approaches could be adopted. One option is to use a very small proportional gain  $k_p$ ; the other is to use a very large one. A low value of  $k_p$  results in smaller changes in the equivalence factor between successive updates. This restrained increase allows the equivalence factor at the start of the final deceleration phase to remain closer to its initial value, which slows down the rise in SOC in the final part of the mission and helps to achieve a final SOC close to the initial 60%, thereby ensuring charge-sustaining behavior. On the other hand, using a high value of  $k_p$  enables the controller to respond more aggressively, allowing for quicker adjustments to the equivalence factor when power demand changes suddenly or when strong deceleration occurs. This may help the system reach the appropriate equivalence factor more quickly and potentially maintain charge sustaining operation. However, large changes in the equivalence factor may lead to instability in the controller's behavior, making this approach less desirable. For these reasons, the controller with a lower  $k_p$  is preferred and, in fact, the optimal  $k_p$  for this specific cycle is the smallest among all of those analyzed.

Noteably, alternative solutions may be adopted. For example, rather than modifying the controller's aggressiveness through  $k_p$ , it may be sufficient to start with an initial equivalence factor closer to the optimal value for the AMDC cycle. The current initial value,  $s_0$ , was calibrated on an ARDC cycle, which differs significantly from the AMDC. Therefore, using an initial value of the equivalence factor tuned on a cycle more similar to the AMDC can realistically increase the performance of the controller.

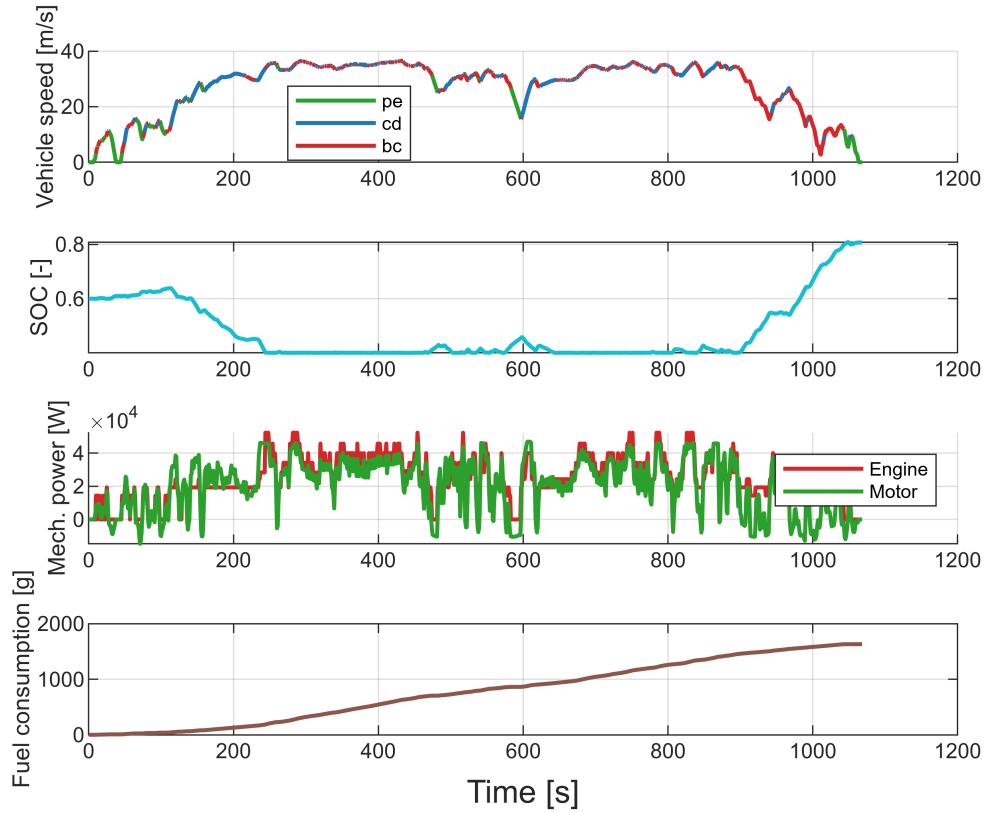
Another potential improvement could involve reducing the update interval of the equivalence factor, allowing the controller to adapt more frequently. This would help the system to converge more rapidly to the desired value. However, if the update frequency is too high, the controller may no longer be able to detect the trend of the SOC effectively, which could result in an inappropriate adaptation of the equivalence factor and potential instability.

Another insight concerns the fact that, at the end of the cycle, the SOC constraints are not respected. This situation indicates that, at those time instants, all combinations of engine speed and torque result in an unfeasible operating point, either due to the violation of the system's physical limits or because of SOC constraint violations. Hence, the controller decides to violate the SOC constraints rather than select an operating point that would violate the physical limitations of the system. This decision is justified by the nature of the SOC constraints: unlike physical limits, which are absolute and must not be exceeded, SOC constraints are constraints introduced to preserve battery health and extend its lifespan. In this case, the controller prioritizes the feasibility of powertrain operation, even at the cost of slightly exceeding the SOC boundaries, in order to maintain system integrity.

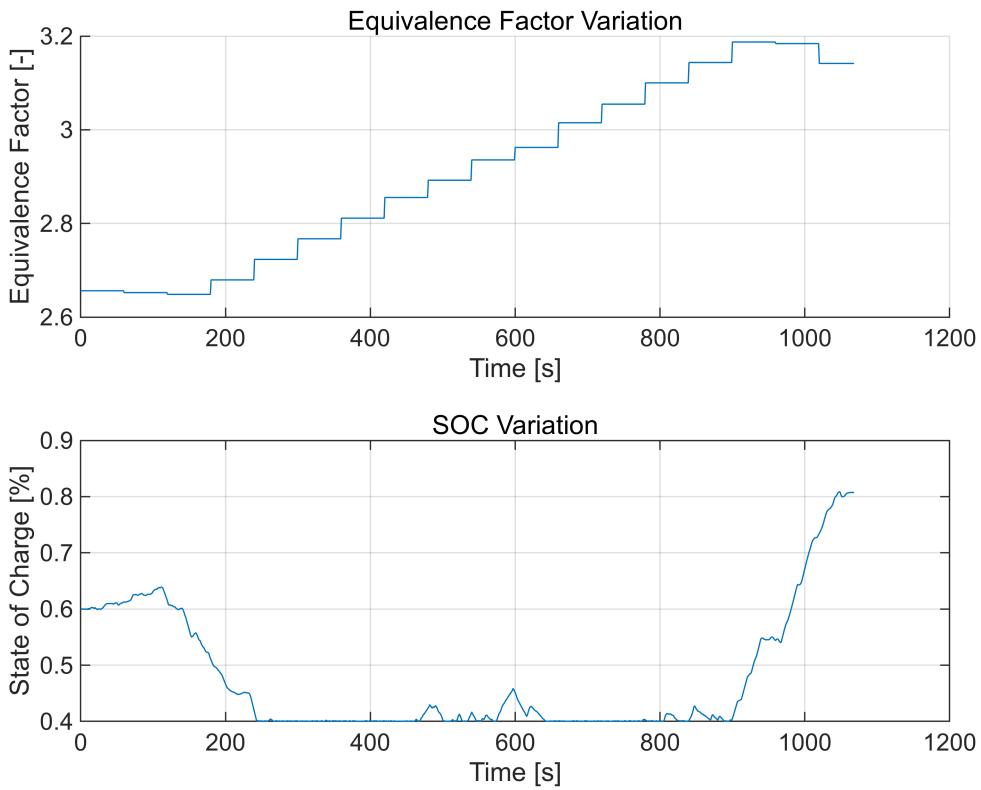
The final observation concerns the evolution of fuel consumption. As illustrated in the fourth chart, fuel consumption follows a nearly linear trend throughout the cycle. This behavior reflects the nature of the driving cycle, which maintains a relatively constant power demand for most of its duration. Slightly lower rates of increase in fuel consumption are observed at the beginning and end of the cycle, corresponding to the vehicle's entry onto and exit from the highway, phases that require less power. This near linear growth in

fuel consumption is also consistent with the engine power profile. As shown in the corresponding chart, engine power remains at a high level for the majority of the cycle, resulting in a steady increase in fuel consumption. Additionally, the engine rarely turns off, indicating that the powertrain seldom operates in pure electric mode. This is further supported by the vehicle speed profile, which is predominantly composed of red and blue segments, representing engine-on operation. In contrast, green segments, which denote pure electric driving, appear only during a few deceleration phases where regenerative braking is used to recharge the battery.

```
[fig, t] = mainProfiles(prof_AMDC);
```



```
[fig2, t2] = eqFactorSOCPlot(time_AMDC, s_AMDC, SOC_AMDC);
```



## AUDC

The following charts illustrate the results obtained from the AUDC cycle simulation. This mission is representative of an urban driving scenario, which, in this case, is repeated five times. Due to its urban nature, the AUDC presents significantly different operating conditions for the HEV compared to the AMDC case, which instead reflects motorway conditions. Specifically, the AUDC is characterized by lower average speeds and higher acceleration values.

A key observation is the periodic behavior exhibited by both the equivalence factor and the State Of Charge. This trend is expected, given that the AUDC consists of multiple repetitions of the same driving cycle. As a result, the time evolution of the SOC and equivalence factor naturally follows a periodic pattern.

Another important result concerns the final SOC. As shown in the SOC variation chart, the battery's State Of Charge at the end of the AUDC cycle is very close to its initial value, even though the  $k_p$  value used in the simulation is not optimized specifically for the AUDC. This indicates that the controller is nearly able to maintain charge-sustaining operation. This outcome is attributed to the fact that the average  $k_p$  value, calculated as the mean of the optimal  $k_p$  values across different cycles, is very close to the optimal value for the AUDC. As a result, the slight deviation in the proportional gain only causes a minor SOC error, ensuring effective controller performance in terms of final SOC consistency.

By analyzing the variations in the equivalence factor, the SOC evolution, and the speed profile in more detail, additional insights into the controller's behavior can be drawn. At the beginning of the mission, the battery SOC shows a slow increase over time, which can be attributed to an excessively high initial value of the equivalence factor. This gradual increase continues until approximately  $t = 900$  seconds, corresponding to the end of the first of the five repeated cycles. This observation suggests that the initial guess of the equivalence factor

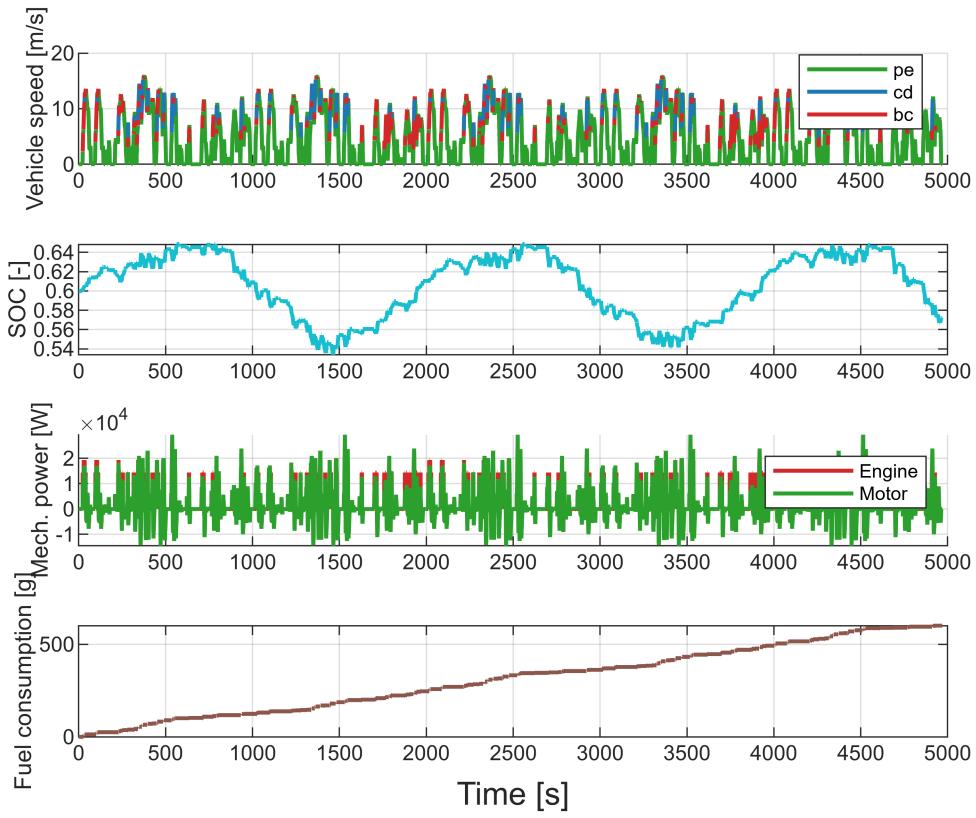
overestimates the optimal value for the single cycle of the repeated sequence. Indeed, as previously mentioned, the  $s_0$  value was tuned based on the ARDC cycle, which is characterized by higher speeds and greater power demands. As a result, the tuning process produced an equivalence factor that penalizes battery usage more heavily than what would be ideal for the AUDC. Nevertheless, since the differences in power demand between the ARDC and AUDC cycles are not so large, the error in the equivalence factor is relatively small. This leads to only a slow deviation in the SOC trajectory from the desired value. This contrasts with the AMDC case, where a significant error in the equivalence factor caused a much steeper deviation in the SOC evolution. To counteract this deviation, the controller progressively decreases the equivalence factor, thereby promoting greater use of electrical energy. This adjustment allows the battery to discharge gradually and restore the initial value of the SOC.

After this initial phase of decrease of the equivalence factor, the system enters an increasing phase, during which the equivalence factor rises almost back to its initial value. This behavior results from the excessively low equivalence factor value reached during the first phase of the mission, which causes an overuse of battery energy in the subsequent phase. To counteract this, the controller begins to increase the equivalence factor, eventually restoring it to its original value, so that the State Of Charge can return to the desired level. This repeated rise and fall of the equivalence factor results in the previously mentioned periodic oscillation in both the SOC and the equivalence factor. Such harmonic and periodic evolution is enabled by the nature of the cycle itself. Specifically, the AUDC cycle is a strictly urban driving cycle that maintains nearly constant conditions throughout the mission. As a result, there are no phases with significantly higher power demand compared to others, and the overall power demand remains relatively steady. This consistency prevents rapid fluctuations in the battery SOC, unlike in the AMDC cycle, allowing the controller to adapt the equivalence factor more efficiently and avoiding large, abrupt changes in SOC.

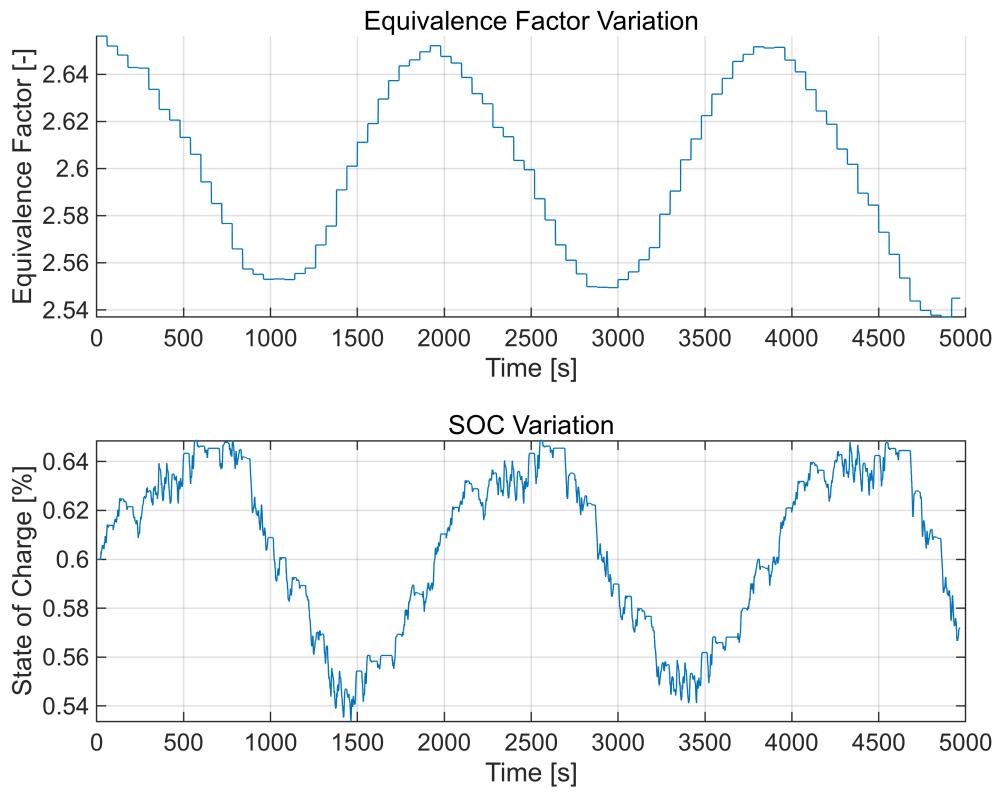
Another insight that emerges from analyzing the harmonic behavior of the SOC and the equivalence factor is related to the delay introduced by the controller. Observing their similar sinusoidal patterns, it is evident that the equivalence factor leads the SOC variation by approximately 400 seconds. This indicates that the A-ECMS controller does not have an immediate impact on the powertrain behavior; instead, a delay is required before its influence becomes effective in the SOC. Again, to reduce the delay, but also to mitigate the amplitude of the oscillations, one possible solution is to decrease the update interval of the equivalence factor. This would allow the controller to respond more rapidly to changes in SOC, leading to a faster response of the controller on the SOC variation.

Similar to the AMDC cycle, the AUDC cycle also exhibits a nearly linear increase in fuel consumption. This behavior is primarily due to the relatively constant power demand throughout the entire mission. However, a significant difference emerges between the two cycles: although the AUDC lasts approximately five times longer than the AMDC, its total fuel consumption is considerably lower. This outcome is mainly due to the lower average power demand of the AUDC, visible in the engine power plot. However, it may also be attributed to a less effective control strategy by the A-ECMS during the AMDC cycle. In particular, the proportional gain  $k_p$  appears to be significantly more suboptimal for highway driving rather than the AUDC cycle, leading to less efficient energy management.

```
[fig3, t3] = mainProfiles(prof_AUDC);
```



```
[fig4, t4] = eqFactorSOCplot(time_AUDC, s_AUDC, SOC_AUDC);
```



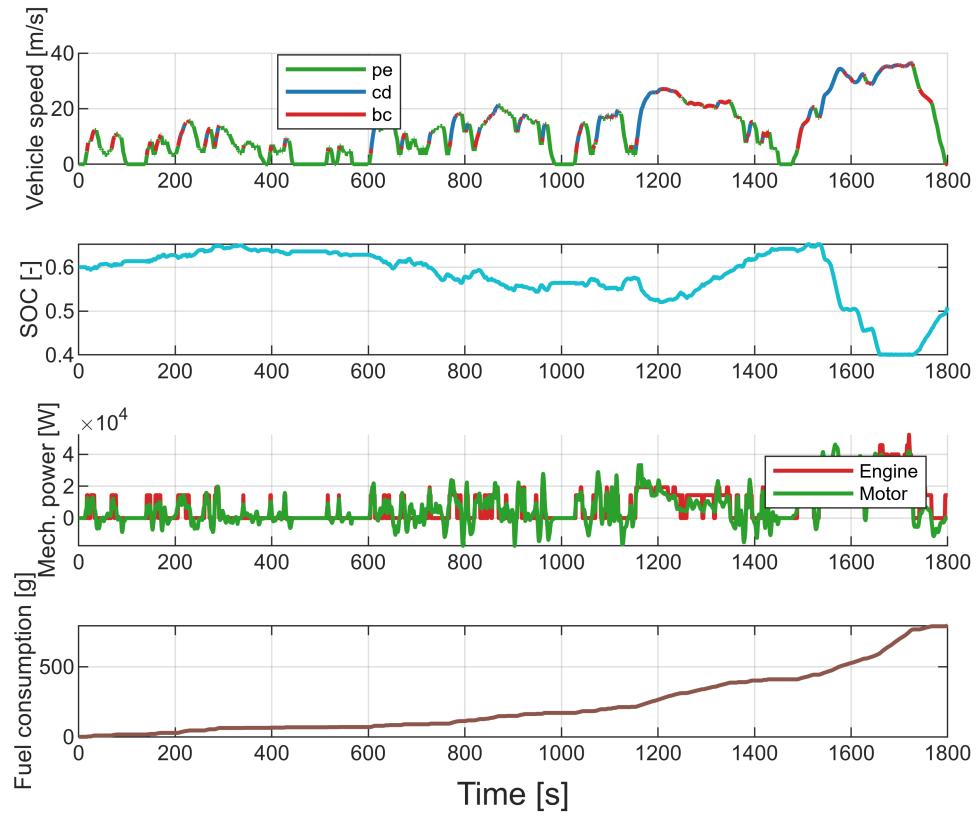
## WLTP

This section is dedicated to the analysis of the A-ECMS controller applied to the WLTP cycle, which is designed to more accurately reflect real-world and modern driving conditions, hence combining both low-speed, low-acceleration phases with high-speed, high-acceleration ones.

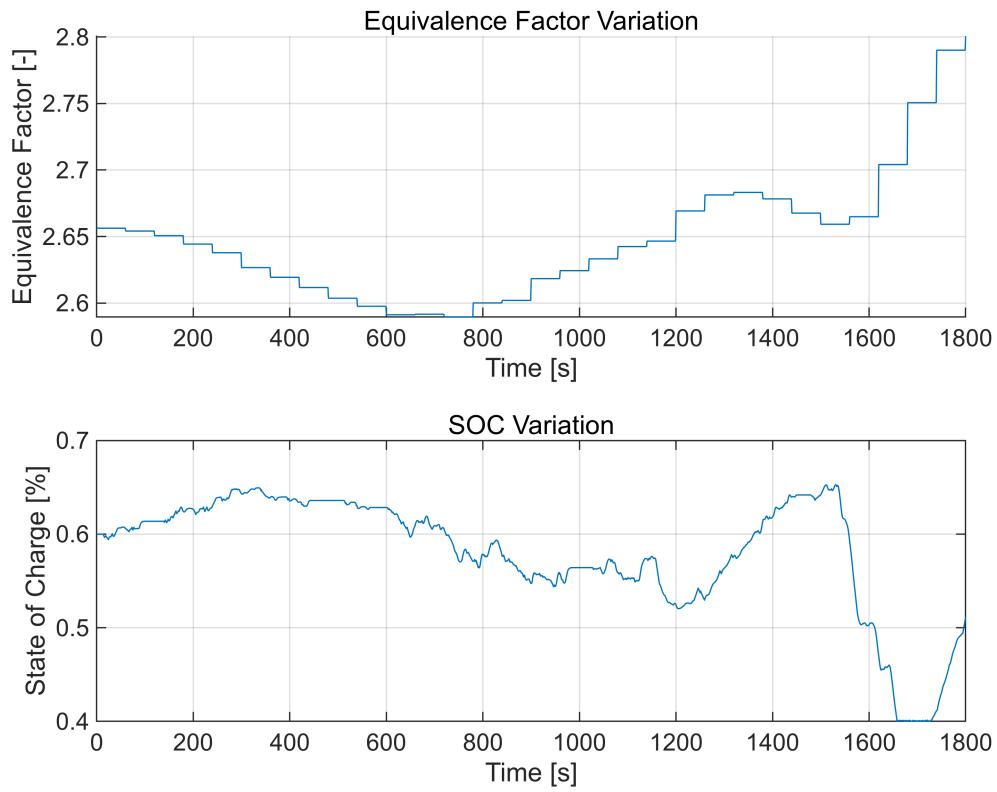
The first notable characteristic of the results is the non-periodic evolution of both the State Of Charge and the equivalence factor, quite different from the previously analyzed AUDC cycle. More specifically, both trends exhibit very limited deviations from their initial values for most of the cycle duration. However, toward the end of the mission, the SOC suddenly drops to the lower limit imposed by the SOC constraints - i.e., 40%. This behavior can be attributed to the structure of the WLTP cycle, which consists of a low-speed phase at the beginning, a medium-speed phase in the middle, and a high-speed phase at the end. This is in contrast to the AUDC, where power demand remained relatively constant throughout the cycle. If only the first two phases are considered, low and medium speed, the WLTP closely resembles the ARDC cycle, which was used to tune the initial value of the equivalence factor. As a result, the initial guess for the equivalence factor is already quite close to the optimal value for these phases. Consequently, the controller manages the SOC effectively during this portion of the cycle. Indeed, when the medium-speed phase ends and the high-speed phase begins, around  $t = 1450$  seconds, the SOC remains at approximately 64%, a relatively good value given that no specific tuning was performed for the WLTP cycle. However, because the high-speed phase introduces significantly different power demands compared to the ARDC, which lacks such a phase, the controller can no longer maintain the stable SOC deviation. Indeed, at the start of the high-speed phase, the equivalence factor is still optimized for low-to-medium speed driving. This value becomes suboptimal for the final phase, where power demand increases sharply. A higher equivalence factor is needed in this context to reduce battery usage and prevent rapid depletion. Therefore, the controller activates its adaptation mechanism, gradually increasing the equivalence factor to discourage battery consumption and avoid excessive discharge. This dynamic is clearly visible in the equivalence factor variation plot: it shows relatively small deviations during the first two phases, followed by a sharp increase during the final high-speed segment of the cycle.

A key difference compared to the previously analyzed cycles lies in the evolution of fuel consumption, which no longer follows a linear trend. Instead, it exhibits an exponential rise, reflecting the continuous increase in the power required from the powertrain over time.

```
[fig5, t5] = mainProfiles(prof_WLTP);
```



```
[fig6, t6] = eqFactorSOCplot(time_WLTP, s_WLTP, SOC_WLTP);
```



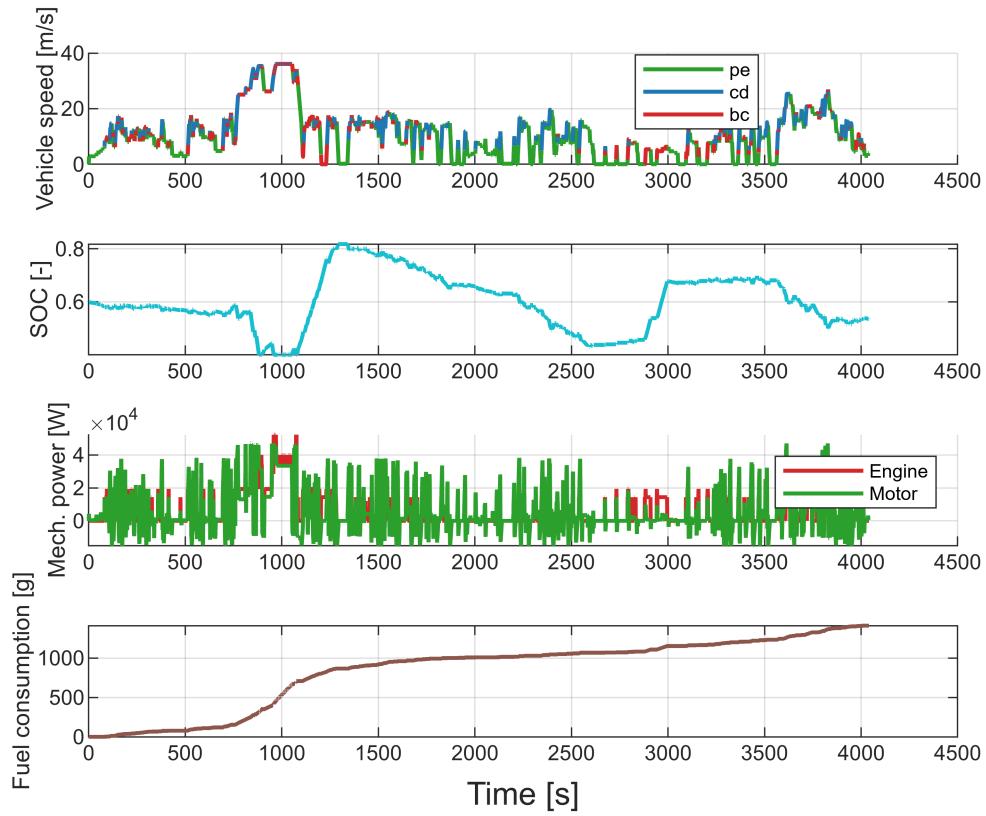
## Turin Test

The results from the Turin Test show SOC and equivalence factor variations that are consistent with the behaviors observed in the previously analyzed cycles. Therefore, this cycle can be effectively used to summarize the overall behavior of the controller across different driving conditions.

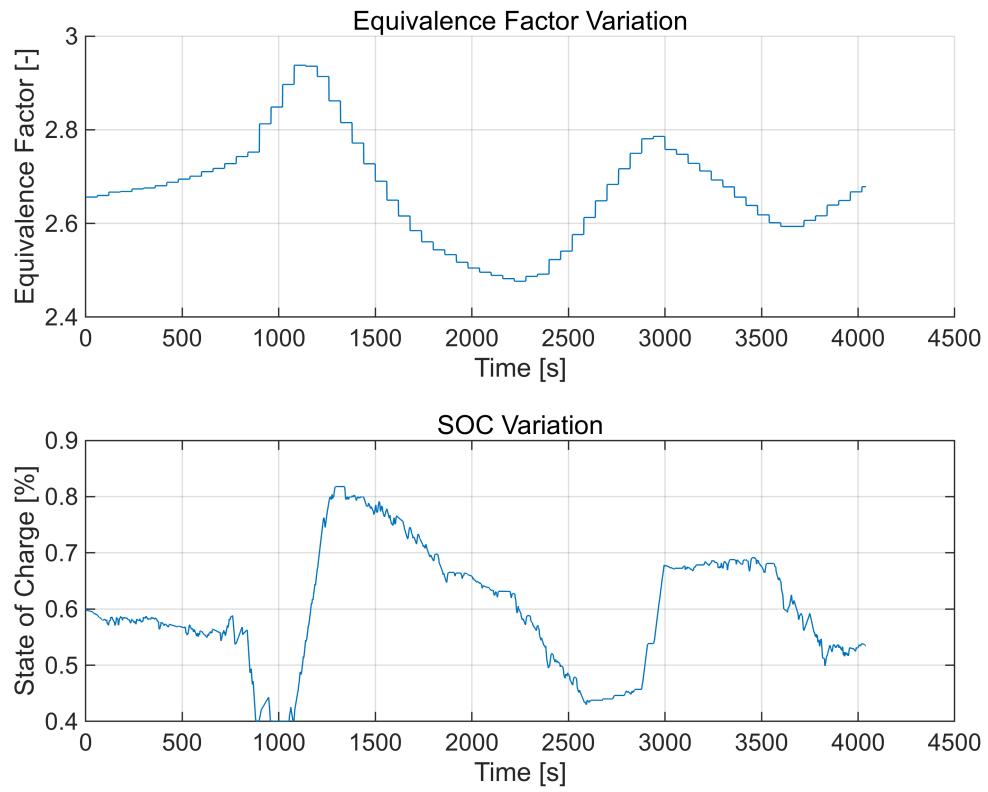
As in the first part of the WLTP cycle, the initial phase of the Turin Test exhibits minimal variation in both SOC and the equivalence factor. This is once again due to the similarity between this portion of the cycle and the ARDC cycle, on which the initial equivalence factor was tuned. Subsequently, a high-speed phase occurs, leading to a rapid decrease in SOC. To counteract this drop, the controller significantly increases the equivalence factor, similar to the behavior observed in the AMDC cycle. Following this, a lower power demand phase takes place. In this phase, the controller begins to reduce the equivalence factor to encourage the use of electric energy. This is necessary because, during the early part of this low-demand phase, the elevated equivalence factor, previously adapted for the high-speed segment, causes an excessive battery charging. This overcharging is attributed to the adaptation delay previously described in the analysis of the AUDC cycle. Toward the end of the Turin Test, the equivalence factor undergoes another increase followed by a decrease, driven by the alternation between low-speed and high-speed phases. This pattern once again highlights the controller's adaptive response to varying power demands, aiming to maintain SOC within acceptable limits while optimizing energy usage.

Regarding fuel consumption, a noticeable change in the slope of its time evolution occurs around  $t = 800$  s, indicating a significant increase in fuel usage during that portion of the mission. This behavior aligns with expectations, as the steep rise in fuel consumption corresponds to the high-speed phase of the cycle. During this phase, the powertrain experiences a high power demand, which results in the engine operating under heavier load conditions. Consequently, this leads to a steeper increase in fuel consumption compared to other segments of the cycle, where the power demand is lower and the fuel consumption rises more gradually. This high load operating condition of the engine is also evident in the chart comparing engine and electric motor power. Specifically, during the high speed interval, the motor reaches its peak power output across the entire cycle, indicating the highest power demand. Accordingly, the engine maintains a high power output throughout this phase, close to its maximum rated power of 55 kW.

```
[fig7, t7] = mainProfiles(prof_TurinTest);
```



```
[fig8, t8] = eqFactorSOCplot(time_TurinTest, s_TurinTest, SOC_TurinTest);
```



After analyzing all the driving cycles, several additional insights can be drawn regarding the behavior and effectiveness of the A-ECMS controller. The first key conclusion is that the controller consistently implements a correct energy management strategy across all driving cycles. While the final SOC values may not strictly satisfy charge-sustaining objectives in every case, this discrepancy is not a consequence of a flaw in the control logic but rather a result of the selected gain value  $k_p$ . Therefore, the control strategy itself remains valid. The core of the controller's adaptive logic is based on feedback from the SOC. Specifically, the input to the controller at each update step is the difference between the current SOC and the initial SOC. A lower SOC compared to the initial value yields a positive error, prompting an increase in the equivalence factor to reduce battery usage. Conversely, when the SOC is higher than the initial value, a negative error is produced, leading to a decrease in the equivalence factor and greater reliance on electric energy. This feedback behavior is clearly observable in the plots from the Turin Test. From the beginning of the cycle up to  $t = 1165$  s, the SOC remains below the target value, which causes the equivalence factor to steadily increase. From  $t = 1165$  s to  $t = 2230$  s, the SOC rises above the desired threshold, producing a negative error and a corresponding decrease in the equivalence factor.

This relationship between the SOC and the equivalence factor resembles the mathematical relationship between a function and its derivative. Specifically, the SOC curve appears to mimic the derivative of the equivalence factor curve, with a notable inversion of maxima and minima. For example, when the SOC crosses the desired value, taken as 60%, the equivalence factor exhibits a peak, either a maximum or a minimum. At  $t = 1165$  s, the SOC crosses the 60% threshold from below, and the equivalence factor reaches a local maximum. Similarly, at  $t = 2230$  s, the SOC crosses the 60% level from above, coinciding with a minimum in the equivalence factor curve. This parallelism is even more apparent in the AUDC cycle, where both the SOC and equivalence factor follow quasi-harmonic trends. Assuming the equivalence factor behaves like a sinusoidal function, its "derivative", represented conceptually by the SOC variation, would resemble a cosine function. A cosine is phase-shifted in time relative to a sine wave, analogous to the observed delay between the controller's action, i.e. adjusting the equivalence factor, and the resulting change in SOC. However, it's important to note that this analogy is not completely exact: in the actual system, the SOC curve appears to be shifted forward, differently from the ideal mathematical derivative which is phase shifted rearward.

Another notable observation relates to the magnitude of the step changes in the equivalence factor at each update. Since the state feedback is based on the deviation of the SOC from the target value (60%), the farther the SOC is from this reference, the larger the adjustment in the equivalence factor. This behavior is also evident in the Turin Test results. In the early stages of the cycle, when the SOC remains close to 60%, the equivalence factor shows only minor changes at each update. However, when the SOC deviates significantly, such as at  $t = 1000$  s or  $t = 1300$  s, the corresponding step changes in the equivalence factor are substantially larger.

To gain a deeper understanding of the influence of the proportional gain on the battery State Of Charge, multiple simulations were conducted, each with a different  $k_p$  value. The final SOC from each simulation was evaluated and stored. These simulations were run in a separate script and the results were saved in a .mat file to ensure faster execution of the main code. Below, the results of these simulations are plotted to analyze the behavior of the adaptive ECMS function.

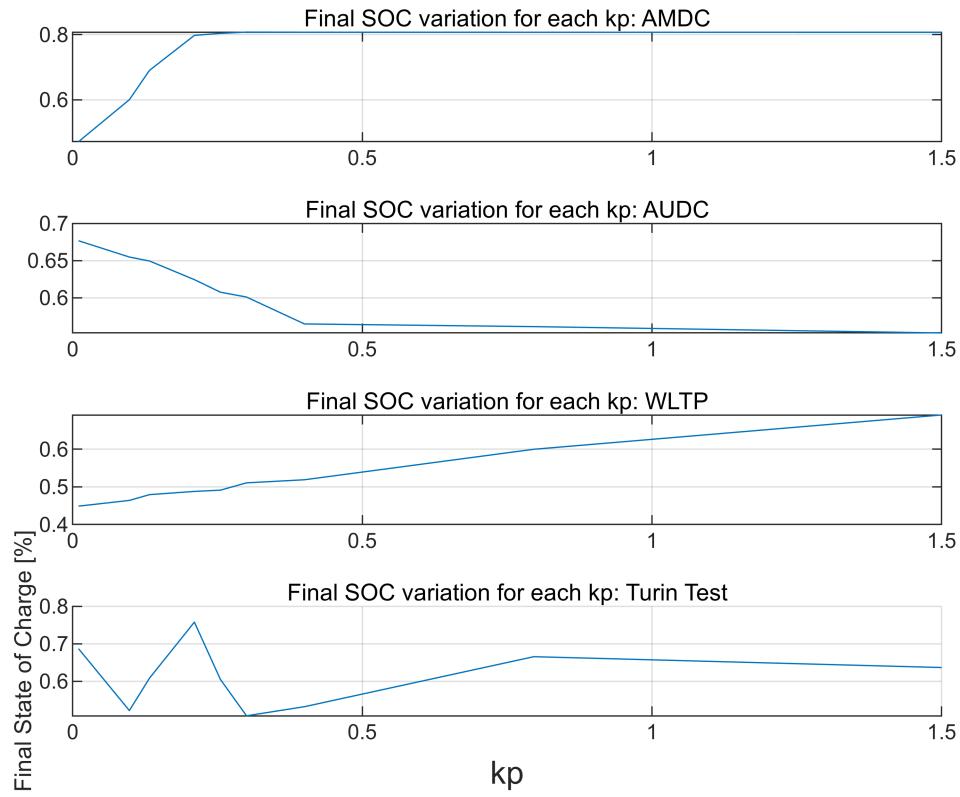
From the plots, different trends in final SOC can be observed across the different driving cycles. The AMDC and AUDC cycles exhibit an initial linear trend: an increasing SOC for AMDC and a decreasing SOC for AUDC.

This is followed by a saturation phase in both cases. For AMDC, the SOC reaches an upper saturation limit, which coincides with the upper SOC constraint. On the other hand, the AUDC cycle shows a lower saturation limit, occurring around 55% SOC. Differently, the WLTP cycle displays a consistently increasing linear trend throughout the entire kp range. Finally, the Turin Test shows the most irregular behavior, characterized by oscillations in the final SOC that gradually stabilize and decrease in amplitude as kp increases. A notable consequence of this non-monotonic trend is that, unlike the other cycles, the Turin Test allows for multiple kp values that result in a final SOC of 60%, thus ensuring charge-sustaining operation. This behavior increases the probability of selecting an effective kp value for charge-sustaining performance, even when using a random or non-optimized guess, making the Turin Test more tolerant to kp variations in this context.

The obtained results clearly indicate that it is not possible to identify a single value of kp that performs well across all four test cases, as its optimal tuning strongly depends on the specific characteristics of each driving cycle. However, it can be observed that for all four analyzed cycles, the kp values that lead to a final SOC of approximately 60% generally fall within the range [0.1; 0.35]. This limited interval represents the best trade-off in terms of controller performance. Indeed, as previously discussed, setting kp too high results in a more aggressive controller, characterized by abrupt variations in the equivalence factor and wider fluctuations, behavior that is generally undesirable. Conversely, a kp value that is too low may cause the controller to respond too slowly, reducing its ability to effectively adjust the equivalence factor in response to power demand fluctuations throughout the cycles. Therefore, if kp must be selected without prior knowledge of the driving cycle, a reasonable approach is to choose a value within the aforementioned range.

To achieve better performance, the controller could be enhanced by incorporating additional state feedback, such as driving cycle prediction. This prediction provides foresight into future power demands, allowing the controller to proactively adjust the equivalence factor, guaranteeing improved battery SOC management. Alternatively, if only real-time state feedback is available, an adaptive mechanism for tuning kp itself could be introduced. One possibility is to adjust kp dynamically based on the SOC error, i.e., the magnitude of the deviation from the target SOC. A larger error implies a greater discrepancy between the current and optimal equivalence factor, hence a higher kp could be used to accelerate adaptation. Conversely, when the error is small, only a minor adjustment is needed, and a lower kp can be used to avoid excessive oscillations. To implement this, a predefined lookup table could be designed, mapping SOC error values to corresponding kp values. This would allow the controller to apply a more optimal kp value whenever adaptation is needed, thereby improving the overall performance of the system.

```
% Load results
kpFinalSOC = load("FinalSoc_KP.mat");
[fig9, t9] = kpFinalSOCEvolution(kpFinalSOC);
```

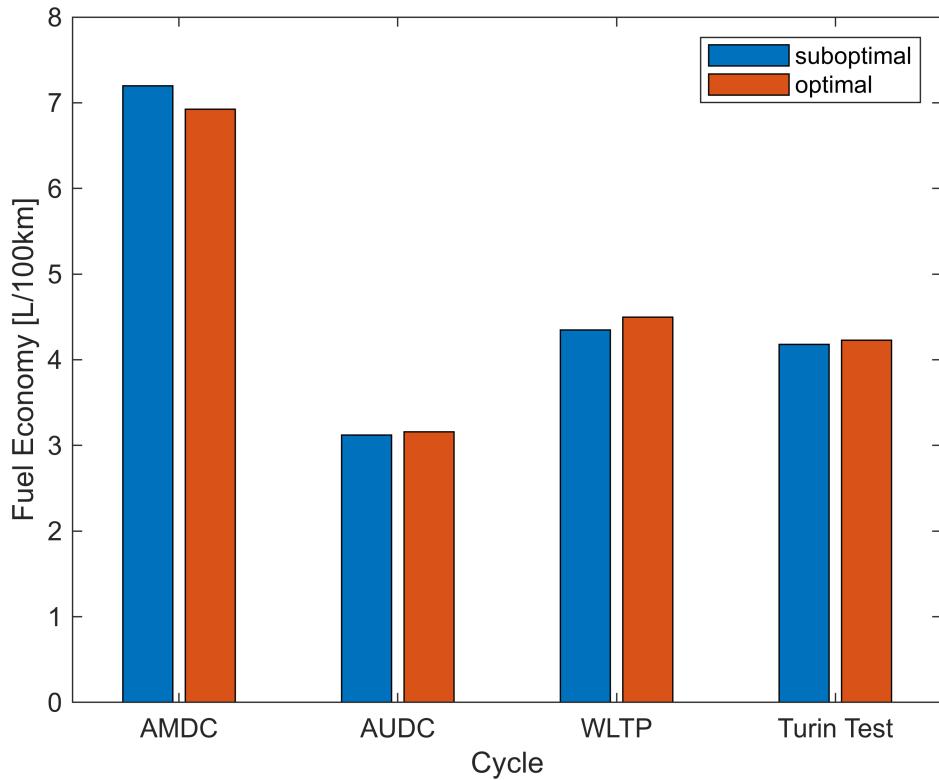


Another important consideration concerns the update time. In this project, it is set to 60 seconds, but this value is not fixed. Instead, it should be chosen mainly in relation to the battery capacity. For instance, in vehicles with large battery capacities, such as Plug-in Hybrid, the State Of Charge varies more slowly. As a result, a longer update interval should be used, so to allow the system to get meaningful data about SOC changes. Moreover reducing  $T_{update}$  leads to lower computational effort. On the other hand, in vehicles with smaller batteries, such as Mild Hybrid, SOC can change much more rapidly. In these cases, a faster update rate is necessary to ensure that the equivalence factor is adjusted in time, preventing excessively fast battery discharge or charging.

The following analysis focuses on the effect of the proportional gain on fuel consumption. The diagram below compares fuel economy across the different driving cycles, using both the kp value applied in the current simulation and the optimal one. As before, these evaluations are carried out asynchronously in a separate script. The results are then stored in a .mat file, which is subsequently loaded into the current simulation.

The results show that the AMDC cycle exhibits the highest fuel consumption, while the AUDC cycle shows the lowest. This is consistent with expectations, as AMDC has the highest power demand and AUDC the lowest. It is also evident that using a suboptimal kp value leads to either an overestimation or underestimation of fuel economy across all cycles. This outcome is linked to the fact that the battery SOC at the end of the mission does not match its initial value, resulting in either excessive or insufficient fuel usage.

```
fuelEconomy = load("FuelEconomy.mat");
fig10 = fuelEconomyhistogram(fuelEconomy);
```



## The A-ECMS controller

The development of the energy management strategy is based on the A-ECMS (Adaptive - Equivalent Consumption Minimization Strategy), implemented through the corresponding Adaptive ECMS control. This strategy operates on the same fundamental principles as the standard ECMS but introduces the key improvement of dynamically adapting the equivalence factor throughout the driving cycle. The primary objective of the A-ECMS 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.

As previously mentioned, differently from the traditional ECMS, the adaptive version does not keep a fixed equivalence factor for the entire cycle. Instead, it dynamically adjusts this factor in order to contrast the SOC variation, aiming to maintain the SOC around the target value, typically the initial SOC. This dynamic behavior eliminates the need for prior tuning of the equivalence factor, which in the conventional ECMS requires knowledge of the driving cycle in advance. Consequently, the A-ECMS removes the need for such calibration and significantly reduces computational effort, offering a more flexible solution for real world applications, where the driving cycle is not known *a priori*.

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

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

### **Step 1: adaptation of the equivalence factor**

The first step in the A-ECMS control strategy involves the update of the equivalence factor if necessary. Indeed, the adaptation does not occur continuously at every time step, but rather at fixed intervals of duration  $T_{update}$ , which, in the implemented controller, is set to 60 seconds. Thus, every 60 seconds the equivalence factor is updated with a new tuned value. For this adaptation, the SOC deviation from the target value is exploited as feedback, ensuring that the controller can respond proportionally to the SOC deviation maintaining it near the desired level. In particular, a large deviation indicates that the current equivalence factor is far from optimal, meaning that a significant adjustment is needed. Conversely, a small deviation results in a minor update of the equivalence factor.

In addition to the SOC variation, two other terms contribute to the adaptation strategy. One is the average of the last two equivalence factor values, which introduces a stabilizing effect and smooths out abrupt changes, the other one is  $k_p$ . This last parameter is a proportional gain of the feedback controller which is used as a tuning parameter. Higher values of  $k_p$  result in more aggressive adaptations, producing larger shifts in the equivalence factor between successive update intervals. This leads to a more aggressive controller which impose more abrupt variations of the SOC.

The algorithm implementing the equivalence factor adaptation starts from an initial guess  $s_0$ , which in this project is set to the equivalence factor previously calculated in Project 2 using the bisection method. If the equivalence factor is too low, the battery tends to discharge excessively, risking depletion over time if the equivalence factor were kept at the same level. To counteract this, the controller increases the equivalence factor during an update, so that higher cost is assigned to the electrical energy usage. This leads to a less intensive usage of the battery in the following time steps. By continuously applying this adaptive mechanism throughout the driving cycle, the strategy effectively reverses any undesirable SOC trends, trying to achieve charge-sustaining operation.

### **Step 2: initialization of all combinations of engine speed and torque**

The second step 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.

### **Step 3: 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 unfeasibilities, 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.

### **Step 4: 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:

- SOC<sub>max</sub> = 80%
- SOC<sub>min</sub> = 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.

## Step 5: 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.

```
function [engSpd, engTrq, sn, tc] = adaptiveEcmsControl(SOC, LHV, battEnergy,
vehSpd, vehAcc, veh, sn, s_nm1, tc, Tupdate, kp, SOC_0)
% ECMS Controller - Selects optimal engine speed and torque to minimize equivalent
fuel consumption

% Inputs:
% SOC           - Battery actual 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 [-]
% sn            - Actual Equivalence factor [-]
% s_nm1          - Previous Equivalence factor [-]
% tc             - Cycle counter [s]
% Tupdate        - Update equivalence factor time [s]
% kp             - Proportional gain for SOC feedback [-]
% SOC_0          - Battery initial state of charge [-]

% Outputs:
% engSpd         - Optimal engine speed [rad/s]
% engTrq         - Optimal engine torque [Nm]
% sn             - Actual equivalence factor [-]
% tc             - Cycle counter [s]

% Adaptation of the equivalence factor if required
if tc == Tupdate
    sn = (sn + s_nm1) / 2 + kp * (SOC_0 - SOC);      % new value of the equivalence
factor to be used in the next iterations
    tc = 0;                                         % reset the counter
end

% 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);

% Pre-allocation of the variables
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));

% Definition of the penalties for unfeasibilities and SOC constrains
penalty = 1e6;
penaltySOC = 1e3;
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
        % Exclusion of unfeasible controls by adding a penalty
        if unfeas(i,j) % considering unfeas
            fuelFlwRateEq(i,j) = penalty;
        elseif 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) - sn * (battEnergy*3600) / (LHV/1000)
* (SOC_next(i,j) - SOC); % definition of the equivalent fuel flow rate using a
proper formula
        end
    end
end

% 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

```