Manual for PMP Code

Written by: Chonghyuk Song
Data: 2017/07/06

Problem Formulation:

$$\min J = \alpha(SOC_f - SOC_0)^2 + m_{fuel} = \phi(SOC_f) + \int_0^{t_f} \dot{m}_{fuel}(t, \omega_{eng}, T_{eng})\, dt$$

subject to

1. $S\dot{O}C = f(SOC, \omega_{eng}, T_{eng})$ (system equations)

2. t_f given by driving cycle

3. $SOC(0)$ given

4. (*speed* operating limits)

$$\begin{cases} \omega_{eng,\min} \le \omega_{eng} \le \omega_{eng,\max} \\ \omega_{MG1,\min} \le \omega_{MG1} \le \omega_{MG1,\max} \\ \omega_{MG2,\min} \le \omega_{MG2} \le \omega_{MG2,\max} \end{cases}$$

5. (*torque* operating limits)

$$\begin{cases} T_{eng,\min}(\omega_{eng}) \le T_{eng} \le T_{eng,\max}(\omega_{eng}) \\ T_{MG1,\min}(\omega_{MG1}) \le T_{MG1} \le T_{MG1,\max}(\omega_{MG1}) \\ T_{MG2,\min}(\omega_{MG2}) \le T_{MG2} \le T_{MG2,\max}(\omega_{MG2}) \end{cases}$$

6. (*current* operating limits)

$$I_{\min}(SOC) \le I_{batt} \le I_{\max}(SOC)$$

Approach: PMP + Shooting Method:

$$J = \alpha(SOC_f - SOC_0)^2 + \int_0^{t_f} \dot{m}_{fuel}(\omega_{eng}, T_{eng})\, dt$$

$$H \triangleq \dot{m}_{fuel}(\omega_{eng}, T_{eng}) + \lambda f(\omega_{eng}, T_{eng}, SOC)$$

state eqn: $\quad S\dot{O}C = \dfrac{\partial H}{\partial \lambda} = f(\omega_{eng}, T_{eng}, SOC)$

costate eqn: $\quad \dot{\lambda} = -\dfrac{\partial H}{\partial (SOC)} = -\lambda \dfrac{\partial f}{\partial (SOC)}$

control condition: $\quad H(\omega_{eng}^*, T_{eng}^*, \lambda^*, SOC^*, t) \le H(\omega_{eng}, T_{eng}, \lambda^*, SOC^*, t) \quad (\forall u = [\omega_{eng}, T_{eng}] \in U_{feasible})$

natural boundary: $\quad \lambda(t_f) = \dfrac{\partial \phi}{\partial SOC}(SOC(t_f)) = 2\alpha(SOC_f - SOC_0)$

boundary conditions: $\quad t_f$ *given*, SOC_0 *given*

%% 1. Loading All Variables

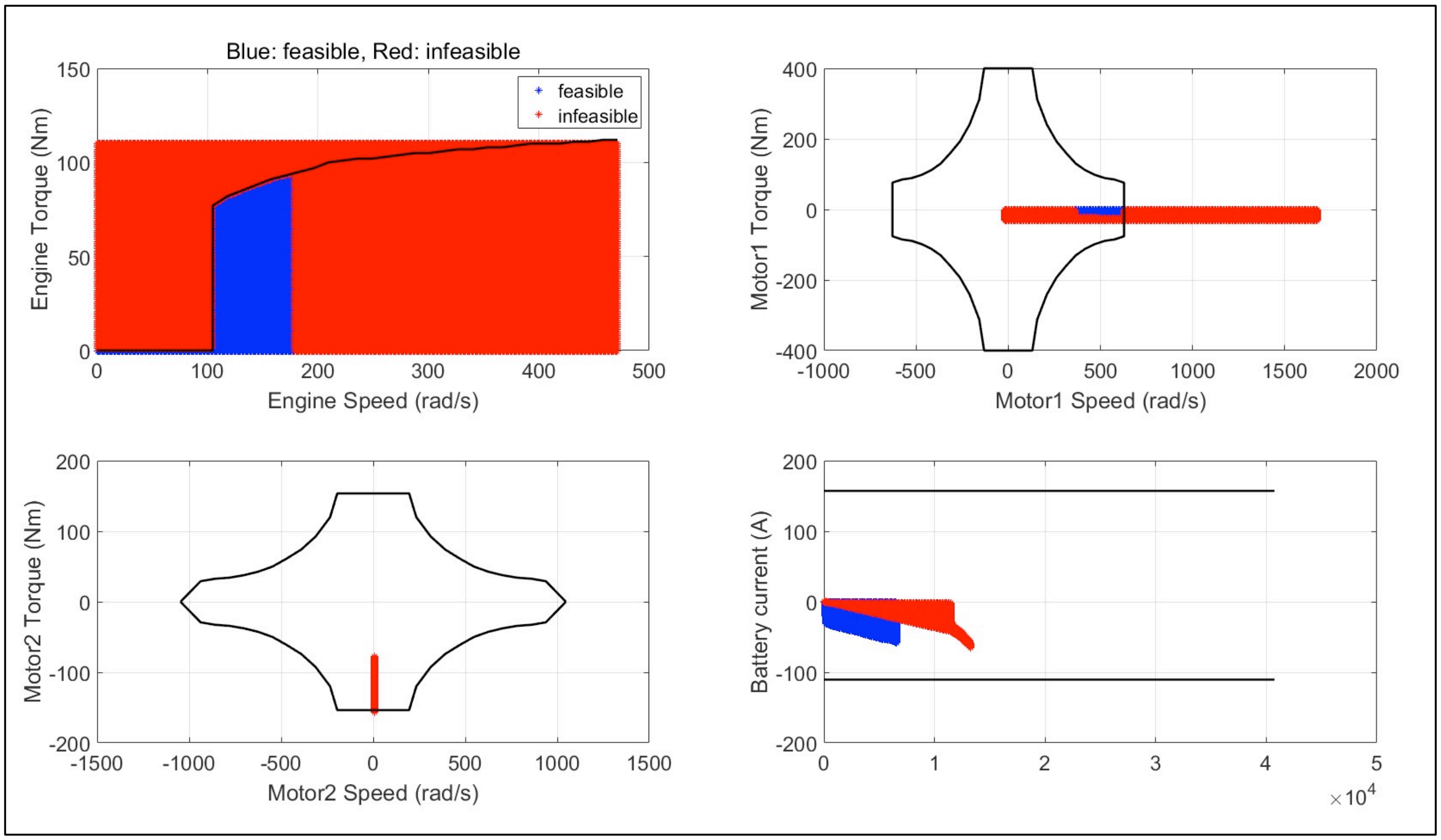
The necessary vehicle specifications and other parameters are loaded from the mat files included with the "PMP_final" script. These variables include those regarding the engine, the two electric motors, the battery, the vehicle itself, and the driving cycle.

%% 2. Setting time-invariant variables ex) calculating d(m_fc)/dt for points in search space

As part of the shooting method, we need to find the optimal control at each time step of the driving cycle. The optimal control is found at each time step via Pontryagin's Minimum Principle (PMP):

$$\boxed{\mathrm{u}^*(t) = \left[\omega_{eng}^*(t),\, T_{eng}^*(t)\right] = \underset{\mathrm{u}\in U_{feasible}}{\arg\min}\, H(t)}$$

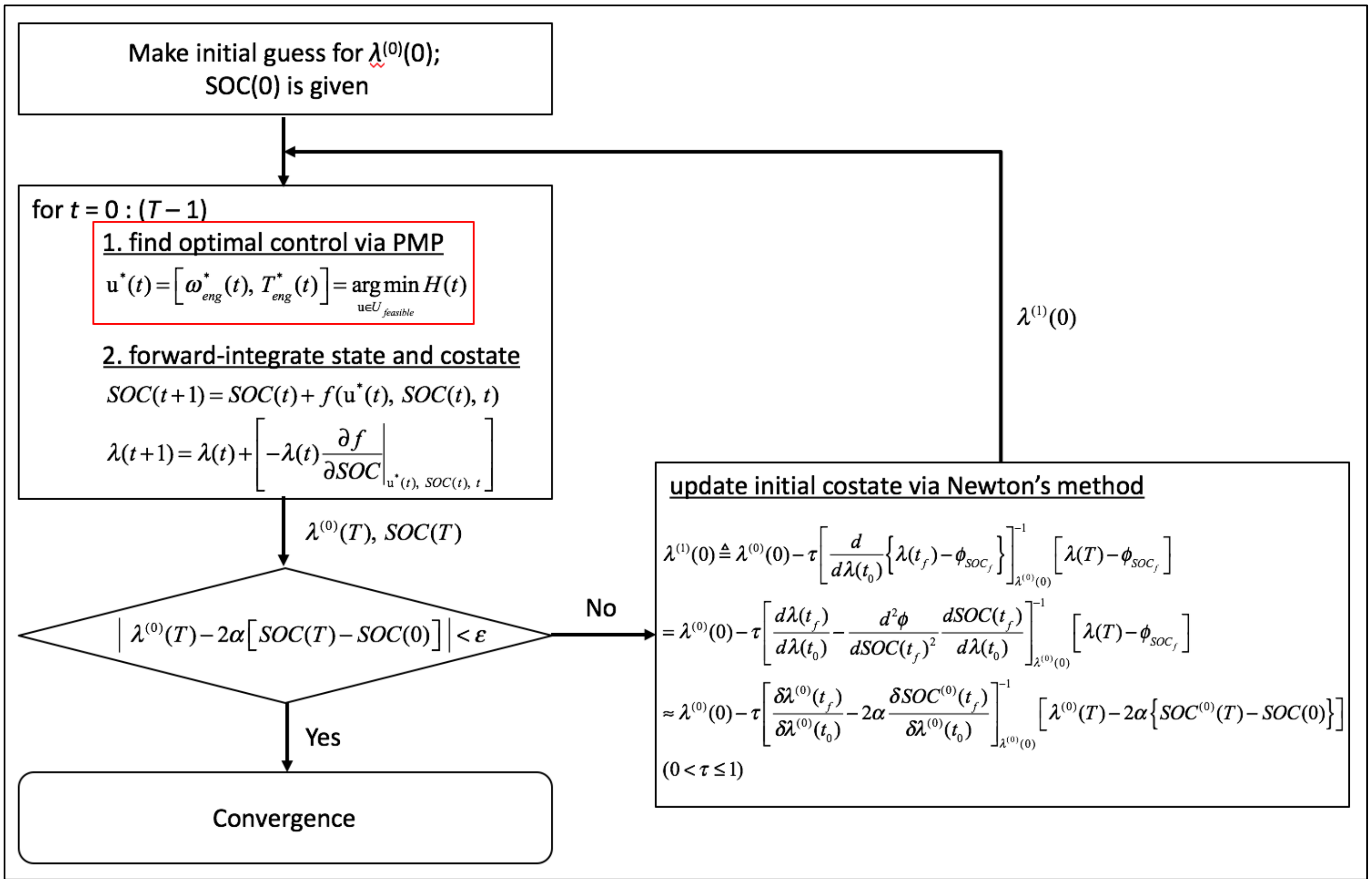
In order to find the control that minimizes the Hamiltonian, we discretize the feasible region on the engine speed-torque plane (control plane) into a mesh grid of resolution 1 (Nm) and 1 (rad/s) for engine torque and speed, respectively. We then check the feasibility of each point in the mesh grid and compute the Hamiltonian for each feasible point. The feasible point which the smallest Hamiltonian corresponds to the optimal control.



When evaluating the Hamiltonian for each point however, the first term (fuel consumption rate term) will be the same for each point in the mesh grid over all time steps. **Hence to avoid redundant computations we calculate the fuel consumption rate for every point in the mesh grid *before* we enter the iterative loop of the shooting method.**

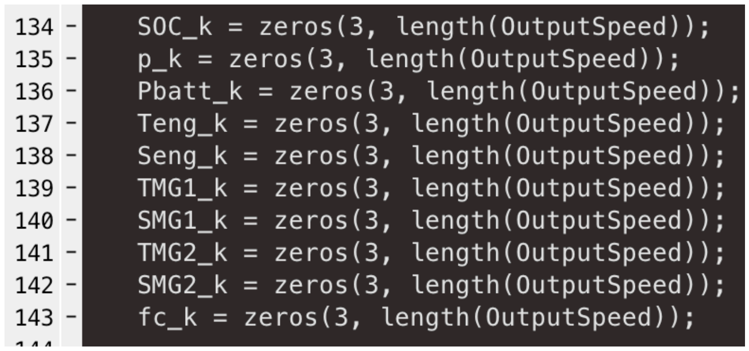
```
%% 3. Shooting Method
```

This part of the script will serve as the outer-loop portion of the shooting method. An initial guess for the initial costate is made and from the inner loop (PMP and forward integration), we obtain values for the final costate and state. If these terminal values do not satisfy the natural boundary condition, the initial costate is updated and the outer-loop is repeated until convergence.



Make initial guess for $\lambda^{(0)}(0)$; SOC(0) is given
for $t = 0 : (T-1)$
1. find optimal control via PMP
$\mathrm{u}^*(t) = \left[\omega^*_{eng}(t),\, T^*_{eng}(t)\right] = \underset{\mathrm{u}\in U_{feasible}}{\arg\min}\, H(t)$
2. forward-integrate state and costate
$SOC(t+1) = SOC(t) + f(\mathrm{u}^*(t),\, SOC(t),\, t)$
$\lambda(t+1) = \lambda(t) + \left[-\lambda(t)\left.\frac{\partial f}{\partial SOC}\right|_{\mathrm{u}^*(t),\, SOC(t),\, t}\right]$
$\lambda^{(0)}(T),\, SOC(T)$
$\left|\lambda^{(0)}(T) - 2\alpha\left[SOC(T) - SOC(0)\right]\right| < \varepsilon$
No
Yes
Convergence
$\lambda^{(1)}(0)$
update initial costate via Newton's method
$\lambda^{(1)}(0) \triangleq \lambda^{(0)}(0) - \tau\left[\frac{d}{d\lambda(t_0)}\left\{\lambda(t_f) - \phi_{SOC_f}\right\}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda(T) - \phi_{SOC_f}\right]$
$= \lambda^{(0)}(0) - \tau\left[\frac{d\lambda(t_f)}{d\lambda(t_0)} - \frac{d^2\phi}{dSOC(t_f)^2}\frac{dSOC(t_f)}{d\lambda(t_0)}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda(T) - \phi_{SOC_f}\right]$
$\approx \lambda^{(0)}(0) - \tau\left[\frac{\delta\lambda^{(0)}(t_f)}{\delta\lambda^{(0)}(t_0)} - 2\alpha\frac{\delta SOC^{(0)}(t_f)}{\delta\lambda^{(0)}(t_0)}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda^{(0)}(T) - 2\alpha\left\{SOC^{(0)}(T) - SOC(0)\right\}\right]$
$(0 < \tau \le 1)$

In this section, we initialize placeholder arrays to store the SOC, costate lambda, battery power, engine torque, engine speed, MG1 torque, MG1 speed, MG2 torque, MG2 speed, and fuel consumption at each time step for the latest iteration of the outer-loop:



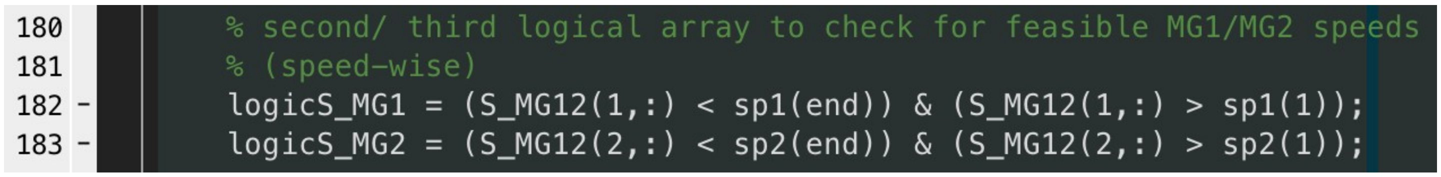
```
SOC_k = zeros(3, length(OutputSpeed));
p_k = zeros(3, length(OutputSpeed));
Pbatt_k = zeros(3, length(OutputSpeed));
Teng_k = zeros(3, length(OutputSpeed));
Seng_k = zeros(3, length(OutputSpeed));
TMG1_k = zeros(3, length(OutputSpeed));
SMG1_k = zeros(3, length(OutputSpeed));
TMG2_k = zeros(3, length(OutputSpeed));
SMG2_k = zeros(3, length(OutputSpeed));
fc_k = zeros(3, length(OutputSpeed));
```

The reason there are three rows in each placeholder is because we also need the *perturbed* values of the final costate to calculate the derivative of the final costate with respect to the initial costate, for the Newton update:

$$\boxed{\begin{aligned}\frac{d\lambda(t_f)}{d\lambda(t_0)} &\approx \frac{\delta\lambda(t_f)}{\delta\lambda(t_0)}\\ &= \frac{\lambda(t_f)\left\{\lambda(t_0)+\varepsilon\right\} - \lambda(t_f)\left\{\lambda(t_0)-\varepsilon\right\}}{2\varepsilon}\end{aligned}}$$

%% 4. PMP step: finding optimal control

In order to find the optimal control, we need to first find the feasible points within the predefined mesh grid on the engine speed – torque plane (control plane). A feasible point is a point that not only stays within the engine operating line, but also whose corresponding points on other graphs stay within their operating limits too!
In the script, this is implemented in the form of 6 logical arrays, each of which is an array of Boolean values that indicate whether each point doesn't violate the limits of a certain variable.



```
    % second/ third logical array to check for feasible MG1/MG2 speeds
    % (speed-wise)
    logicS_MG1 = (S_MG12(1,:) < sp1(end)) & (S_MG12(1,:) > sp1(1));
    logicS_MG2 = (S_MG12(2,:) < sp2(end)) & (S_MG12(2,:) > sp2(1));
```

Only points (*i.e.* indices) whose corresponding entries across all 6 logical arrays are true (1) are deemed feasible, and the evaluation of the Hamiltonian will be done only on these feasible points. We have already evaluated the first term of the Hamiltonian for all points in the search space in section 2; in this section, we now evaluate the second term of the Hamiltonian for all feasible points, given the current SOC value. The point with the smallest Hamiltonian corresponds to the optimal control.

%% 5. forward integration from step t to step t+1

The second step in the inner-loop is forward integration. Having obtained the optimal control from the previous section, we substitute the control terms in the state and costate equations and forward integrate them to obtain the costate and state values of the next time step:

$$\boxed{\begin{aligned} &SOC(t+1) = SOC(t) + f(\mathrm{u}^*(t),\ SOC(t),\ t) \\ &\lambda(t+1) = \lambda(t) + \left[-\lambda(t) \left. \frac{\partial f}{\partial SOC} \right|_{\mathrm{u}^*(t),\ SOC(t),\ t} \right] \end{aligned}}$$

In order to compute the partial derivative of *f* with respect to SOC, we use the quotient rule for differentiation. The partial derivatives of open-circuit voltage and equivalent resistance are interpolated from the lookup table imported from the mat-file "battery.mat" in section 1.

$$\boxed{\begin{aligned} &f = \frac{\sqrt{V_{OC}^2 - 4P_{batt}R_{OC}} - V_{OC}}{2Q_{\max}R_{OC}} = \frac{N}{D} \\ &\frac{\partial f}{\partial SOC} = \frac{\frac{\partial N}{\partial SOC} D - \frac{\partial D}{\partial SOC} N}{D^2} \\ &\frac{\partial N}{\partial SOC} = \frac{1}{2\sqrt{V_{OC}^2 - 4P_{batt}R_{OC}}} \left[2V_{OC}\frac{dV_{OC}}{dSOC} - 4P_{batt}\frac{dR_{OC}}{dSOC} \right] - \frac{dV_{OC}}{dSOC} \\ &\frac{\partial D}{\partial SOC} = 2Q_{\max}\frac{dR_{OC}}{dSOC} \end{aligned}}$$

%% 6. Update Rule

In this section, we check whether natural boundary condition is satisfied by the final state and costate values calculated from the inner loop. The natural boundary condition is satisfied when the final costate value is equal to the partial derivative of the quadratic penalty function (a function of the final state). In practice, we assume convergence when the difference between the two falls within a small tolerance. Here, we define that tolerance to be 0.5, which is a small number compared to the range of costate values dealt with in this problem (~ -300):

$$\left| \lambda^{(0)}(T) - 2\alpha\left[SOC(T) - SOC(0)\right]\right| < \varepsilon$$

If the above inequality is false, then the outer-loop proceeds to the update of the initial costate based on the Newton method, which was outlined in section three. For detailed information on the update rule refer to (1970, Kirk p. 345). Note that the writer made a slight modification to the original update rule, namely the presence of a step size adjustment factor tau, which takes on values between 0 and 1, and helps prevent divergence; The adjustment factor is set as 0.5 in this script.

$$\boxed{\begin{aligned}
&\lambda^{(1)}(0) \triangleq \lambda^{(0)}(0) - \tau\left[\frac{d}{d\lambda(t_0)}\left\{\lambda(t_f) - \phi_{SOC_f}\right\}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda(T) - \phi_{SOC_f}\right]\\
&= \lambda^{(0)}(0) - \tau\left[\frac{d\lambda(t_f)}{d\lambda(t_0)} - \frac{d^2\phi}{dSOC(t_f)^2}\frac{dSOC(t_f)}{d\lambda(t_0)}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda(T) - \phi_{SOC_f}\right]\\
&\approx \lambda^{(0)}(0) - \tau\left[\frac{\delta\lambda^{(0)}(t_f)}{\delta\lambda^{(0)}(t_0)} - 2\alpha\frac{\delta SOC^{(0)}(t_f)}{\delta\lambda^{(0)}(t_0)}\right]^{-1}_{\lambda^{(0)}(0)}\left[\lambda^{(0)}(T) - 2\alpha\left\{SOC^{(0)}(T) - SOC(0)\right\}\right]\\
&(0 < \tau \le 1)
\end{aligned}}$$