

Trabajo de Técnicas

El método de Pontryagin es un método para encontrar máximos o mínimos de un funcional. El método busca la trayectoria óptima que maximiza o minimiza el funcional. A ese proceso se le conoce como optimización de un sistema.

En los archivos adjuntos junto a este enunciado se encuentra un programa en MATLAB donde se usa este método, además, ese programa cuenta con la siguiente documentación para entender el código y el algoritmo.

Enlace: [Pontryagin's Minimum Principle \(kaputtengineers.wixsite.com\)](http://kaputtengineers.wixsite.com).

(a) Comenta el programa línea a línea de manera que un lector pueda entender que hace cada trozo del programa y donde se ubica la implementación del método de Pontryagin.

Disponemos de dos archivos, empezamos comentando el primero. Donde preparamos los datos antes de lanzarnos con método de Pontryagin.

```
// PMP_main.m
%% Pontryagin's Minimum Principle

%% Initialization
% Definimos bien los ciclos, donde tomaremos medidas en cada instante de
% tiempo




% Cargamos todos los archivos de la UDDS (datos de los dinamómetros de
% conducción urbana)
load('UDDS_drive_cycle.mat');

% Los siguientes datos son los correspondientes a cada ciclo:
% ts es la tasa de cambio en el tiempo, es decir, el tiempo de muestreo
Cycle.ts = 1;




% Cantidad de veces que realizamos el ciclo, esto se corresponde con el
% tiempo donde realizamos cada una de las mediciones.
% Para ello tenemos el tiempo discretizado de manera equidistante
Cycle.N = length(t);

% Datos recogidos en cada t
Cycle.P_dem = P_dem';
```

Los archivos cargados son los siguientes:

ipert	name	size	bytes	class	
<input checked="" type="checkbox"/>	 P.d...	1370x1	10960	double	1 0
<input checked="" type="checkbox"/>	 t	1370x1	10960	double	2 0
<input checked="" type="checkbox"/>	 v	1370x1	10960	double	3 0
					4 0
					5 0
					6 0
					7 0
					8 0
					9 0
					10 0
					11 0
					12 0
					13 0
					14 0
					15 0
					16 0
					17 0
					18 0
					19 0
					20 0
					21 0
					22 2.7326
					23 5.0319
					24 7.8507
					25 10.1808
					26 11.8510
					27 3.1185
					28 5.2339
					29 15.0695
					30 7.5627
					31 6.0429
					32 2.3083
					33 -0.9854
					34 -2.2795
					35 -2.2351
					36 -1.5748
					37 -2.1500
					38 -14.9070
					39 -9.3551
					40 0.9679
					41 2.2819
					42 2.3322
					43 3.2939
					44 5.2224

Import	Name ^	Size	Bytes	Class	1
<input checked="" type="checkbox"/>	 P_d...	1370x1	10960	double	0
<input checked="" type="checkbox"/>	 t	1370x1	10960	double	1
<input checked="" type="checkbox"/>	 v	1370x1	10960	double	2
					3
					4
					5
					6
					7
					8
					9
					10
					11
					12
					13
					14
					15
					16
					17
					18
					19
					20
					21
					22
					23
					24
					25
					26
					27
					28
					29
					30
					31
					32
					33
					34
					35
					36
					37
					38
					39
					40
					41
					42
					43
					44

Import	Name ^	Size	Bytes	Class		
<input checked="" type="checkbox"/>	 P_d...	1370x1	10960	double	1	0
<input checked="" type="checkbox"/>	 t	1370x1	10960	double	2	0
<input checked="" type="checkbox"/>	 v	1370x1	10960	double	3	0
					4	0
					5	0
					6	0
					7	0
					8	0
					9	0
					10	0
					11	0
					12	0
					13	0
					14	0
					15	0
					16	0
					17	0
					18	0
					19	0
					20	0
					21	0
					22	1.3411
					23	2.6375
					24	3.8445
					25	5.1410
					26	6.3927
					27	7.5550
					28	7.7338
					29	8.0914
					30	9.2537
					31	9.7008
					32	10.0137
					33	10.0584
					34	9.8796
					35	9.6114
					36	9.3431
					37	9.1196
					38	8.8514
					39	7.5997
					40	6.6609
					41	6.6609
					42	6.7950
					43	6.9291
					44	7.1526

```

%% Battery Inputs
% Datos recogidos en la batería

% Potencia máxima
Bat.P_max = 15; % Medido en kW (kilovatios, es decir Julios por segundo)

% Se establecen los límites inferiores y superiores del estado de carga (SOC) de la batería
Bat.lb_SOC = 0.3;      Bat.ub_SOC = 0.8;

% Se establecen los límites inferiores y superiores de la potencia suministrada de la batería
Bat.lb_P = 0.1*Bat.P_max;  Bat.ub_P = 0.9*Bat.P_max;

% Voltaje de circuito abierto de la batería.
Bat.U_oc = 320; % Medido en V (voltios)

% Capacidad de la batería
Bat.Q_bat = 18000; % Medido en Ah (Amperios por hora)

% Resistencia interna de la batería
Bat.R0_B = 0.001; % in Ohm

% Creamos provisinalmente un vector de ceros. Aquí almacenaremos la
% potencia de la batería en cada ciclo
Bat.P_bat = zeros(Cycle.N,1);

% Creamos provisinalmente un vector de ceros. Aquí almacenaremos el SOC(State of Charge)de
% cada ciclo
Bat.SOC = zeros(1,Cycle.N);

% Fijamos el estado de carga inicial establecido previamente en la linea 30
% Asumimos que empezamos el viaje con un 80 %
Bat.SOC(1,1) = Bat.ub_SOC;

%% Engine Modellling
% Modelado del motor

% Valores de potencia de motor
% 1000 puntos equiespaciados entre 1 y 20 kW
Engine.P_engine = linspace(1,20,1000)';

% Para almacenar la potencia óptima en cada ciclo
Engine.P_opt_eng = zeros(Cycle.N,1);

% Flujo de combustible por el motor
Engine.fl_wy_en = 0.001;

% Potencia inicial del motor
Engine.P = Engine.P_engine(1,1);

% Eficiencia del motor
Engine.eff = 0.45;

```

Las siguientes variables son necesarias para calcular la función objetivo y las ecuaciones de estado en cada ciclo, y para encontrar el camino óptimo que satisface la función objetivo y las ecuaciones de estado.

%% PMP Implementation

% Principio mínimo de Pontryagin

% Para almacenar el costate de la función objetivo en cada ciclo.

% Esto es una medida de sensibilidad de la función objetivo

costate_p = zeros(Cycle.N,1);

% Para almacenar el peso de cada ciclo.

% Le da importancia a una variable frente a otra

w_factor = zeros(Cycle.N,1);

Obtenemos resultado con nuestra función Pontryagin.

Probamos con dos condiciones iniciales diferentes.

% Obtenemos el estado de carga de la batería con dos valores iniciales

% diferentes

% Esto lo conseguimos pasando nuestros datos por la función creada en el

% fichero pontryagin

SOC_1 = pontryagin(-5.33,Bat,Cycle,Engine);

SOC_2 = pontryagin(20000,Bat,Cycle,Engine);

// pontryagin.m

%% Principio mínimo de Pontryagin

function soc = pontryagin(p0,Bat,Cycle,Engine)

% Objetivo: Encontrar el camino óptimo que satisface la función objetivo y

% las ecuaciones de estado

% Valor inicial de la ecuación de co-estado

costate_p(1,1) = p0;

% A partir de aquí iniciamos el algoritmo tomando los datos en cada

% instante de tiempo (Cycle.N = 1370)

for i = 1:Cycle.N-1

% Potencia de la batería, esto es la diferencia entre la potencia

% demandada y la potencia del motor

Bat.P_bat = Cycle.P_dem(i) - Engine.P;

% Determinamos si el estado de carga de la batería se sitúa entre los

% límites establecidos, estableciendo los factores de peso

if Bat.SOC(i) <= Bat.lb_SOC

 w_factor(i) = -10^8;

elseif Bat.SOC(i) >= Bat.ub_SOC

 w_factor(i) = +10^8;

else

 w_factor(i) = 0;

end



Añadimos un peso a la función en el caso de que se salga de los límites, eso sigue la siguiente función

$$k = 10^8$$

Additional Weight function

$$w(SOC) = \begin{cases} 0 & \text{if } SOC_{max} < SOC < SOC_{min} \\ K & \text{if } SOC < SOC_{min} \\ -K & \text{if } SOC > SOC_{max} \end{cases}$$

Este peso nos sirve para penalizar posteriormente en el hamiltoniano

% Calculamos la corriente de la batería a partir de la ecuación de estado

$$I_{bat} = (Bat.U_{oc} - \sqrt{Bat.U_{oc}^2 - Bat.R_0 \cdot Bat.P_{bat} \cdot 1000 \cdot 4}) / (Bat.R_0 \cdot 2);$$

Esta $I(t)$ es la misma que la señalada en rojo

% Cambio en el SOC (variación)

$$\Delta SOC = -(Cycle.ts \cdot I_{bat}) / (0.9 \cdot Bat.Q_{bat});$$

Esta variación de la batería la necesitaremos para sacar el hamiltoniano, encontraremos así el óptimo de la función objetivo. En azul

State Equations

$$\dot{x}(t) = f(t, x(t), u(t)) \quad \rightarrow \quad \begin{aligned} \dot{SOC}(t) &= -\frac{1}{\eta_{coul}^{sign(I(t))}} \frac{I(t)}{Q_{nom}} \\ \dot{SOC}(t) &= -\frac{1}{\eta_{coul}^{sign(I(t))} \cdot Q_{nom}} \left[\frac{V_{oc}(x)}{2R_0(x)} - \sqrt{\left(\frac{V_{oc}(x)}{2R_0(x)} \right)^2 - \frac{u(t)}{2R_0(x)}} \right] \end{aligned}$$

% Valor de la función objetivo

$$H = Engine.fl_wy_en \cdot Cycle.ts \cdot Engine.P_{engine} / Engine.eff + (costate_p(i) + w_factor(i)) \cdot \Delta SOC;$$

% Establecemos el H actual como el mínimo y la potencia actual como la óptima en el motor

$$H_{min} = H;$$

$$P_{eng} = Engine.P;$$

Aquí estamos sacando el hamiltoniano

Hamiltonian

$$\mathcal{H}(t, x(t), u(t), \lambda(t)) = F(t, x(t), u(t)) + \lambda^T f(t, x(t), u(t))$$



$$\begin{aligned} \mathcal{H}(P_{batt}(t), x(t), P_{batt}(t), \lambda(t)) \\ = \dot{m}_f(P_{batt}(t), u(t)) + (\lambda(t) + w(SOC)) \cdot \dot{SOC}(t) \end{aligned}$$

```

% Recorremos los valores de la potencia del motor, para encontrar la
% óptima
for j =2:length(Engine.P_engine)

    % Asignamos la potencia del motor a la iteración correspondiente
    Engine.P = Engine.P_engine(j,1);

    % Diferencia entre la energía que se demanda y la que tenemos en el
    % motor
    Bat.P_bat = Cycle.P_dem(i)-Engine.P;

    % Calculamos H al igual que antes
    I_bat = (Bat.U_oc-sqrt(Bat.U_oc^2-Bat.R0_B*Bat.P_bat*1000*4))/(Bat.R0_B*2);
    Del_soc = -(Cycle.ts*I_bat)/(0.9*Bat.Q_bat);
    H = Engine.fl_wy_en*Cycle.ts*Engine.P_engine/Engine.eff + (costate_p(i)+w_factor(i))*Del_soc;

    % Si el resultado es mayor que el que teníamos actualizamos y
    % establecemos la potencia como óptima
    if H_min > H
        H_min = H;
        P_eng = Engine.P;
    end
end

% Nos quedamos con la potencia óptima obtenida
Engine.P_opt_eng(i) = P_eng;

```

A partir del linspace que creamos inicialmente, vemos cual de todas las potencias entre las mil que tenemos es la mínima para nuestra función.

```

% Volvemos a calcular la potencia y la corriente de la batería donde obtuvimos la óptima
Bat.P_bat(i) = Cycle.P_dem(i)-Engine.P_opt_eng(i);
I_bat = (Bat.U_oc-sqrt(Bat.U_oc^2-Bat.R0_B*Bat.P_bat(i)*1000*4))/(Bat.R0_B*2);

% Calculamos el nuevo SOC y la nueva costate.
% La siguiente iteración partirá de estos valores.
Bat.SOC(i+1) = Bat.SOC(i) - (Cycle.ts*I_bat/(0.9*Bat.Q_bat));
del_p = costate_p(i)*1 / Bat.Q_bat * (1/(2*Bat.R0_B) - 1/(4*Bat.R0_B^2) * Bat.U_oc / sqrt(Bat.U_oc^2/(4*Bat.R0_B^2) - Bat.P_bat(i)*1000/Bat.R0_B)) * 1;
costate_p(i+1) = del_p*Cycle.ts + costate_p(i);

end

% Guardamos todos los estados de carga obtenidos durante cada una de las
% iteraciones
soc = Bat.SOC;
end

```

Cuando acabemos todas las iteraciones devolvemos el vector con todos los óptimos obtenidos, para finalmente graficarlo y observar la evolución que hemos tenido con respecto al tiempo.

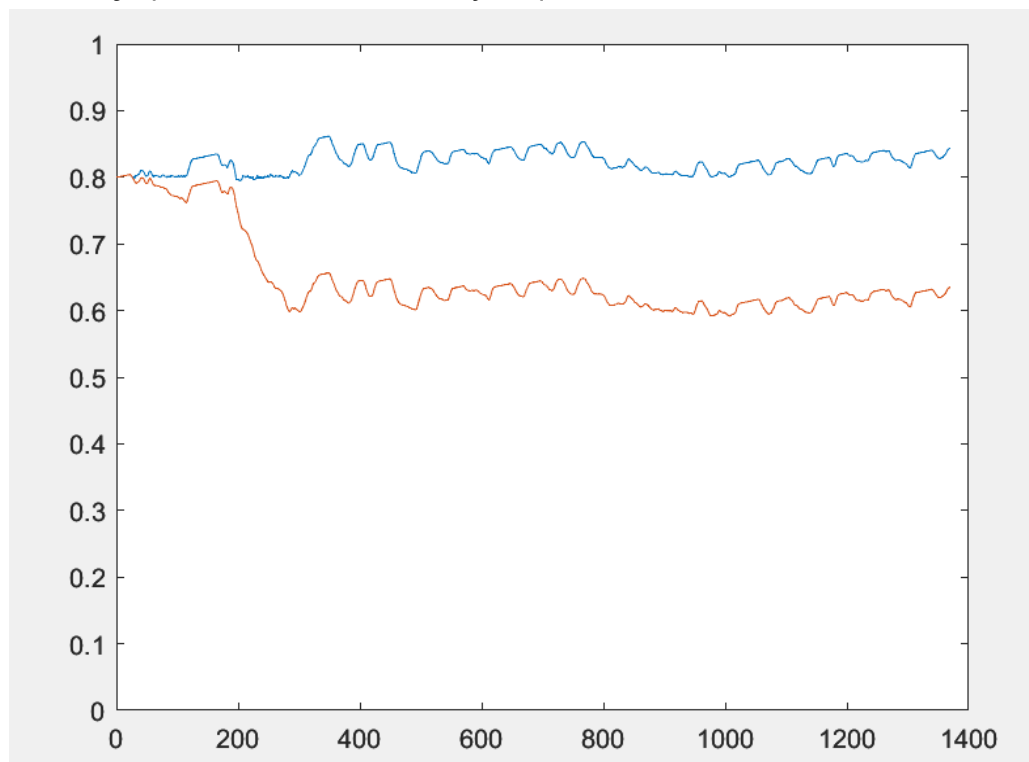
```

%% Plotting
% Graficamos los resultados obtenidos y vemos la comparación entre dos
% estados iniciales diferentes
figure
plot(SOC_1)
hold on
plot(SOC_2)
hold off
ylim([0 1]);

```

RESULTADOS:

En azul nos queda para una condición inicial es menor o igual que 0
El naranja para una condición mayor que 0



Este código es una implementación del principio mínimo de Pontryagin para optimizar la eficiencia energética en un sistema híbrido de batería-motor. Utiliza un enfoque de programación dinámica para calcular la potencia óptima del motor en cada ciclo y el estado de carga de la batería en función de la potencia demandada y los parámetros del sistema.

b) Traza la conexión entre el método de Pontryagin numérico (el implementado en el código) con el teórico.

El método de Pontryagin teórico es una técnica matemática para resolver problemas de optimización dinámica. Se basa en la idea de utilizar funciones costo y restricciones para describir el sistema y luego utilizar ecuaciones diferenciales para encontrar la trayectoria óptima del sistema a lo largo del tiempo. El método de Pontryagin numérico es una forma de aplicar esta técnica a un sistema específico, en este caso, el sistema híbrido de batería-motor.

En el código, se establecen las funciones costo y restricciones a través de las variables de entrada, como la potencia demandada, los límites de SOC y la potencia, y los parámetros del sistema, como la capacidad de la batería y la resistencia interna. Luego, se utilizan las ecuaciones de estado de la batería y las ecuaciones del costate para calcular la trayectoria óptima del sistema en cada ciclo.

En resumen, el código implementa el método de Pontryagin teórico de forma numérica para resolver el problema de optimización dinámica en un sistema híbrido de batería-motor. Utiliza las ecuaciones de estado del sistema y las restricciones para calcular la trayectoria óptima y la potencia óptima del motor.