Controllo della temperatura di una casa

Indice

Descrizione del problema

L'obiettivo del progetto è regolare la temperatura di una casa utilizzando un controllore MPC. Si ipotizza che la casa non condivida alcun muro esterno con altre abitazioni, che i flussi d'aria tra le stanze della casa siano trascurabili e che il riscaldamento della casa sia affidato a tre termosifoni, uno per ogni stanza. La pianta della casa in oggetto è mostrata nella figura seguente:

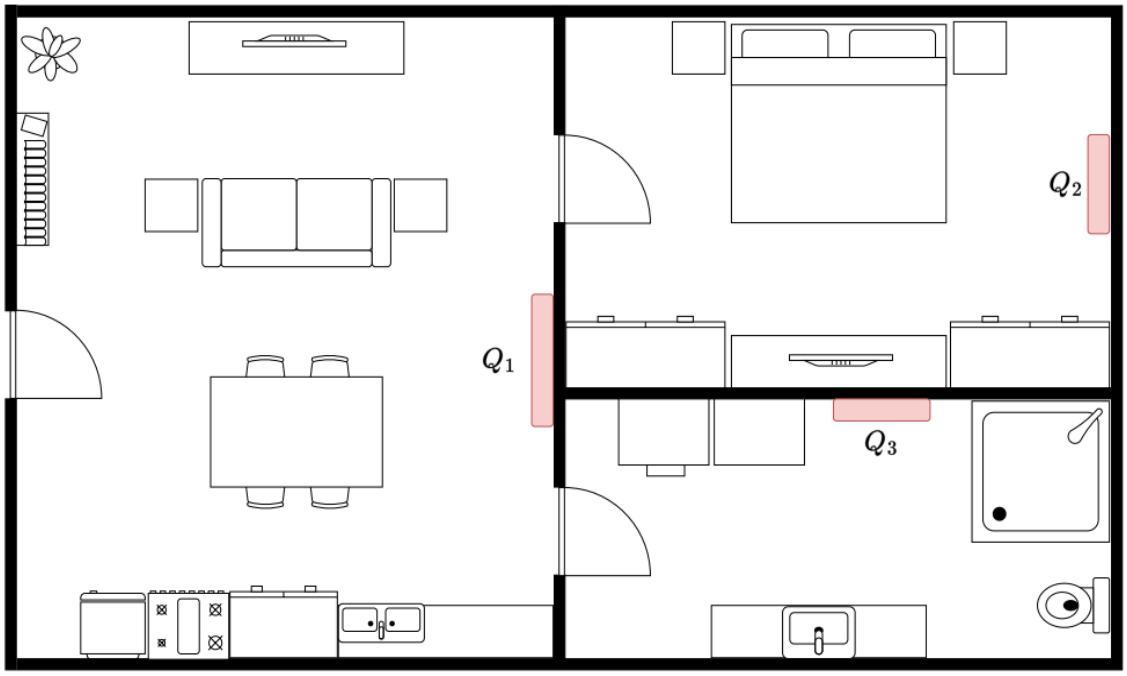


Figura 1: Planimetria della casa.

La dinamica della temperatura all'interno della casa è definita dalle equazioni seguenti:

$$\begin{cases} C_1\dot{T}_1(t) = Q_1(t) - k_{1,2}(t)(T_1(t) - T_2(t)) - k_{1,3}(t)(T_1(t) - T_3(t)) - k_{ext}(T_1(t) - T_{ext}) \\ C_2\dot{T}_2(t) = Q_2(t) + k_{1,2}(t)(T_1(t) - T_2(t)) - k_{2,3}(t)(T_2(t) - T_3(t)) - k_{ext}(T_2(t) - T_{ext}) \\ C_3\dot{T}_3(t) = Q_3(t) + k_{1,3}(t)(T_1(t) - T_3(t)) + k_{2,3}(t)(T_2(t) - T_3(t)) - k_{ext}(T_3(t) - T_{ext}) \\ \dot{Q}_1 = (Q_{1,r}(t) - Q_1(t))/\tau_1 \\ \dot{Q}_2 = (Q_{2,r}(t) - Q_2(t))/\tau_2 \\ \dot{Q}_3 = (Q_{3,r}(t) - Q_3(t))/\tau_3 \end{cases} \quad \text{con } k_{i,j} = \overline{k}_{i,j} + \frac{4}{1 + e^{-0.5\|T_i(t) - T_j(t)\|_2}}$$

Il significato delle grandezze presenti nelle precedenti equazioni è il seguente:

- $T_i\,[K]$ è la temperatura della i-esima stanza;
- $T_{ext}\,[K]$ è la temperatura esterna, ipotizzata costante.
- $C_i\,[J/s]$ è la capacità termica della i-esima stanza;
- $k_{i,j}\,[W/K]$ è un parametro che regola lo scambio di calore tra l'i-esima e la j-esima stanza;
- $k_{ext}\,[W/K]$ è un parametro che regola lo scambio di calore tra le stanze della casa e l'esterno;
- $Q_i\,[W]$ è la potenza termica dell'i-esimo termosifone;
- $Q_{i,r}\,[W]$ è la potenza termica di riferimento dell'i-esimo termosifone;
- $\tau_i\,[s]$ è la costante di tempo dell'i-esimo termosifone.

I parametri noti sono riportati nella tabella seguente:

Parametro	**Valore**	**Unità**
C_1	6800	J/s
C_2	5600	J/s
C_3	5100	J/s
$\overline{k}_{1,2}$	15	W/K
$\overline{k}_{1,3}$	20	W/K
$\overline{k}_{2,3}$	15	W/K
k_{ext}	8	W/K
T_{ext}	278	K
τ_1	400	s
τ_2	450	s
τ_3	450	s

Tabella 1: Parametri noti del sistema in esame.

La potenza erogata dai tre termosifoni è non-negativa e mai superiore ai 150W, mentre la temperatura all'interno delle tre stanze non deve mai scendere al di sotto dei 286K.

Gli ingressi controllati del sistema sono le potenze di riferimento dei tre termosifoni, $Q_{i,r}$, $i \in \{1,2,3\}$.

L'obiettivo del lavoro è progettare un controllore MPC in grado di portare il sistema dalla condizione iniziale $(T_1, T_2, T_3, Q_1, Q_2, Q_3) = (288, 288, 288, 0, 0, 0)$ all'equilibrio $(293, 293, 293, 120, 120, 120)$, rispettando sempre i vincoli, e simulare il funzionamento del sistema in anello chiuso. Il tempo minimo di campionamento, T_s, è pari ad $1\ s$.

Parametri costanti del modello della casa

Inizialmente, si creano delle variabili per salvare tutti i parametri noti del modello.

```
clc
close all
clear
T_ext = 278; % [K], temperatura esterna ipotizzata costante
C = [6800 5600 5100]; % [J/s], capacità termica della i-esima stanza
k_bar = [15 20 15]; % [W/K], scambio di calore fra l'i-esima e la j-esima stanza
k_ext = 8; % [W/K], scambio di calore fra le stanze e l'esterno
Q_ref = [150 150 150]; % [W], potenza termica di riferimento dell'i-esimo termosifone
tau = [400 450 450]; % [s], costante di tempo dell'i-esimo termosifone
x_eq = [293 293 293 120 120 120]';
x_ci = [288 288 288 0 0 0]';
Ts = 1; % [s], tempo di campionamento minimo
```

Equazioni del sistema in variabili di stato

Prima di procedere ad analizzare il sistema, è necessario riscrivere le equazioni della sua dinamica nello spazio degli stati. Si definiscono il vettore degli stati, $\mathbf{x}$, ed il vettore degli ingressi, $\mathbf{u}$:

$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]' := [T_1, T_2, T_3, Q_1, Q_2, Q_3]'$, $\mathbf{x} \in \mathbb{R}^6$

$\mathbf{u} = [u_1, u_2, u_3]' := [Q_{1,r}, Q_{2,r}, Q_{3,r}]'$, $\mathbf{u} \in \mathbb{R}^3$

Il sistema diventa quindi:

$$\begin{cases} \dot{x}_1(t) = (x_4(t) - k_{1,2}(t)(x_1(t) - x_2(t)) - k_{1,3}(t)(x_1(t) - x_3(t)) - k_{ext}(x_1(t) - T_{ext}))/C_1 \\ \dot{x}_2(t) = (x_5(t) + k_{1,2}(t)(x_1(t) - x_2(t)) - k_{2,3}(t)(x_2(t) - x_3(t)) - k_{ext}(x_2(t) - T_{ext})/C_2 \\ \dot{x}_3(t) = (x_6(t) + k_{1,3}(t)(x_1(t) - x_3(t)) + k_{2,3}(t)(x_2(t) - x_3(t)) - k_{ext}(x_3(t) - T_{ext})/C_3 \\ \dot{x}_4 = (u_1(t) - x_4(t))/\tau_1 \\ \dot{x}_5 = (u_2(t) - x_5(t))/\tau_2 \\ \dot{x}_6 = (u_3(t) - x_6(t))/\tau_3 \end{cases} \quad \text{con } k_{i,j} = \bar{k}_{i,j} + \frac{4}{1 + e^{-0.5\|x_i(t) - x_j(t)\|_2}}$$

Le equazioni vengono scritte in Matlab utilizzando la notazione simbolica.

```
syms x1 x2 x3 x4 x5 x6 x1d x2d x3d x4d x5d x6d u1 u2 u3;
% Coefficienti k(i,j)
% Definizione dei vettori per poter calcolare correttamente le norme
```

```
x1_vect = [x1;0;0];
x2_vect = [0;x2;0];
x3_vect = [0;0;x3];
k12 = k_bar(1)+(4/(1+exp(-0.5*norm(x1_vect-x2_vect,2))^2));
k13 = k_bar(2)+(4/(1+exp(-0.5*norm(x1_vect-x3_vect,2))^2));
k23 = k_bar(3)+(4/(1+exp(-0.5*norm(x2_vect-x3_vect,2))^2));
% Equazioni della dinamica del sistema
eq1 = x1d == (x4 -k12*(x1-x2) -k13*(x1-x3) -k_ext*(x1-T_ext))/C(1);
eq2 = x2d == (x5 +k12*(x1-x2) -k23*(x2-x3) -k_ext*(x2-T_ext))/C(2);
eq3 = x3d == (x6 +k13*(x1-x3) +k23*(x2-x3) -k_ext*(x3-T_ext))/C(3);
eq4 = x4d == (u1-x4)/tau(1);
eq5 = x5d == (u2-x5)/tau(2);
eq6 = x6d == (u3-x6)/tau(3);
equazioni = [eq1;eq2;eq3;eq4;eq5;eq6]
```

```
equazioni =
```

$$\begin{pmatrix} \mathrm{x1d} = \frac{x_4}{6800} - \frac{x_1}{850} - \frac{\sigma_3}{6800} - \frac{\sigma_1}{6800} + \frac{139}{425} \\ \mathrm{x2d} = \frac{x_5}{5600} - \frac{x_2}{700} + \frac{\sigma_3}{5600} - \frac{\sigma_2}{5600} + \frac{139}{350} \\ \mathrm{x3d} = \frac{x_6}{5100} - \frac{2\,x_3}{1275} + \frac{\sigma_2}{5100} + \frac{\sigma_1}{5100} + \frac{556}{1275} \\ \mathrm{x4d} = \frac{u_1}{400} - \frac{x_4}{400} \\ \mathrm{x5d} = \frac{u_2}{450} - \frac{x_5}{450} \\ \mathrm{x6d} = \frac{u_3}{450} - \frac{x_6}{450} \end{pmatrix}$$

where

$$\sigma_1 = \left(\frac{4}{\mathrm{e}^{-\sqrt{|x_1|^2+|x_3|^2}} + 1} + 20 \right) (x_1 - x_3)$$

$$\sigma_2 = \left(\frac{4}{\mathrm{e}^{-\sqrt{|x_2|^2+|x_3|^2}} + 1} + 15 \right) (x_2 - x_3)$$

$$\sigma_3 = \left(\frac{4}{\mathrm{e}^{-\sqrt{|x_1|^2+|x_2|^2}} + 1} + 15 \right) (x_1 - x_2)$$

Vincoli su stato e ingresso

Successivamente, si definiscono i vincoli esistenti su stato e ingresso del sistema.

```
x_min_temp = 286.*ones(3,1); % Vincolo di minimo sulla temperatura dell'i-esima stanza
x_min_power = zeros(3,1); % Vincolo di minimo sulla potenza dell'i-esimo termosifone
x_min = [x_min_temp;x_min_power]; % Vincolo di minimo sullo stato
```

```
x_max_temp = inf.*ones(3,1); % Vincolo di massimo sulla temperatura dell'i-esima stanza
x_max_power = 150.*ones(3,1); % Vincolo di massimo sulla potenza dell'i-esimo termosifone
x_max = [x_max_temp;x_max_power]; % Vincolo di massimo sullo stato
u_min = zeros(3,1); % Vincolo di minimo sulla potenza dell'i-esimo termosifone
u_max = 150.*ones(3,1); % Vincolo di minimo sulla potenza dell'i-esimo termosifone
```

Punti di equilibrio

Dalle equazioni sappiamo che il sistema non è lineare, in quanto alcune variabili di stato compaiono nell'esponenziale. Sarà perciò necessario linearizzare il sistema attorno ad un punto di equilibrio. Per calcolare gli ingressi di equilibrio, si azzera la dinamica del sistema, ponendo a zero le derivate degli stati, e si calcolano le soluzioni del sistema di equazioni sostituendo il punto di equilibrio.

```
% Azzeramento della dinamica
x1d = 0;
x2d = 0;
x3d = 0;
x4d = 0;
x5d = 0;
x6d = 0;

% Sostituzione dei valori noti, ovvero il punto di equilibrio
x1 = x_eq(1);
x2 = x_eq(2);
x3 = x_eq(3);
x4 = x_eq(4);
x5 = x_eq(5);
x6 = x_eq(6);
equazioni_equilibrio = subs(equazioni);

% Ottenimento degli ingressi di equilibrio, u_eq
res = solve(equazioni_equilibrio)
```

```
res = struct with fields:
    u1: 120
    u2: 120
    u3: 120
```

```
u_eq = [double(res.u1) double(res.u2) double(res.u3)]';
```

Sistema linearizzato

Prima di progettare il regolatore MPC, è necessario linearizzare il sistema attorno al punto di equilibrio calcolato nella sezione precedente, così da calcolare le matrici A e B del sistema linearizzato.

```
syms x1 x2 x3 x4 x5 x6 u1 u2 u3
f1 = (x4 -k12*(x1-x2) -k13*(x1-x3) -k_ext*(x1-T_ext))/C(1);
f2 = (x5 +k12*(x1-x2) -k23*(x2-x3) -k_ext*(x2-T_ext))/C(2);
f3 = (x6 +k13*(x1-x3) +k23*(x2-x3) -k_ext*(x3-T_ext))/C(3);
f4 = (u1-x4)/tau(1);
f5 = (u2-x5)/tau(2);
f6 = (u3-x6)/tau(3);
sistema = [f1;f2;f3;f4;f5;f6];
```

```
% Calcolo delle derivate parziali rispetto ad ogni stato
sistema_lin(:,1) = diff(sistema,x1);
sistema_lin(:,2) = diff(sistema,x2);
sistema_lin(:,3) = diff(sistema,x3);
sistema_lin(:,4) = diff(sistema,x4);
sistema_lin(:,5) = diff(sistema,x5);
sistema_lin(:,6) = diff(sistema,x6);
sistema_lin;

% Calcolo delle derivate parziali rispetto ad ogni ingresso
ingresso_lin(:,1) = diff(sistema,u1);
ingresso_lin(:,2) = diff(sistema,u2);
ingresso_lin(:,3) = diff(sistema,u3);
ingresso_lin;

% Sostituzione del punto di equilibrio
x1 = x_eq(1);
x2 = x_eq(2);
x3 = x_eq(3);
x4 = x_eq(4);
x5 = x_eq(5);
x6 = x_eq(6);
u1 = u_eq(1);
u2 = u_eq(2);
u3 = u_eq(3);
A_ct = subs(sistema_lin)
```

```
A_ct =
```

$$\begin{pmatrix} -\frac{1}{850\,\sigma_2}-\frac{43}{6800} & \frac{1}{1700\,\sigma_2}+\frac{3}{1360} & \frac{1}{1700\,\sigma_2}+\frac{1}{340} & \frac{1}{6800} & 0 & 0 \\ \sigma_1 & -\frac{1}{700\,\sigma_2}-\frac{19}{2800} & \sigma_1 & 0 & \frac{1}{5600} & 0 \\ \frac{1}{1275\,\sigma_2}+\frac{1}{255} & \frac{1}{1275\,\sigma_2}+\frac{1}{340} & -\frac{2}{1275\,\sigma_2}-\frac{43}{5100} & 0 & 0 & \frac{1}{5100} \\ 0 & 0 & 0 & -\frac{1}{400} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{450} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{450} \end{pmatrix}$$

where

$$\sigma_1 = \frac{1}{1400\,\sigma_2}+\frac{3}{1120}$$

$$\sigma_2 = e^{-\sqrt{171698}}+1$$

```
B_ct = subs(ingresso_lin)
```

B_ct =

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{400} & 0 & 0 \\ 0 & \frac{1}{450} & 0 \\ 0 & 0 & \frac{1}{450} \end{pmatrix}$$

Autovalori del sistema linearizzato a tempo continuo

Prima di andare a discretizzare il sistema, si analizzano gli autovalori della matrice di stato, A, per studiare la natura del sistema in esame. Questo passaggio è utile sia per capire il comportamento del sistema, che per scegliere il metodo di discretizzazione più adatto.

```
lambda_i = eig(A_ct);
real_lambda_i = double(real(lambda_i))
```

```
real_lambda_i = 6×1
   -0.0025
   -0.0022
   -0.0022
   -0.0014
   -0.0110
   -0.0134
```

```
imag_lambda_i = double(imag(lambda_i([5,6])))
```

```
imag_lambda_i = 2×1
10^-73 ×
   -0.1327
    0.0995
```

Il sistema ha quattro autovalori reali negativi e una coppia di autovalori complessi coniugati con parte reale negativa, di conseguenza è asintoticamente stabile. Tuttavia, questi valori sono molto prossimi all'asse immaginario, quindi significa che sono molto lenti. La parte immaginaria degli autovalori complessi coniugati è praticamente nulla, ciò significa che la dinamica del sistema non è soggetta ad oscillazioni.

Per progettare il controllore, il sistema deve essere completamente raggiungibile. Verifichiamo quest'ipotesi calcolando il rango della matrice di raggiungibilità del sistema.

```
Mr_ct = ctrb(A_ct,B_ct);
rk_Mr_ct = rank(Mr_ct)
```

```
rk_Mr_ct = 6
```

Dato che il rango della matrice di raggiungibilità M_r è pari alla dimensione dello stato, n, ovvero 6, si può concludere che il sistema è completamente raggiungibile.

Matrici del sistema linearizzato a tempo discreto

Dato che i controllori lavorano a tempo discreto, è necessario discretizzare le matrici del sistema. Le matrici A e B sono già note a tempo continuo, mentre C e D no. Visto che per il corretto funzionamento del controllore MPC è necessario retroazionare lo stato, si ipotizza che questa misura sia disponibile, ovvero che lo stato sia accessibile, quindi la matrice C è una matrice identità di dimensioni 6x6. Per quanto riguarda la matrice D, sarà una matrice di zeri di dimensioni 6x3, perché si ipotizza che le misure non dipendano dagli ingressi. Una volta ottenuto il sistema a tempo continuo, si ricaverà il sistema a tempo discreto con l'apposito comando di Matlab. Si sceglie di utilizzare il metodo di Tustin, così che il semipiano reale negativo venga mappato nel cerchio di raggio 1 centrato nell'origine. Dato che il sistema è asintoticamente stabile, si poteva utilizzare anche il metodo di Eulero in avanti.

```
C_ct = eye(6);
D_ct = zeros(6,3);
continuous_time_ss = ss(double(A_ct),double(B_ct),C_ct,D_ct);
discrete_time_ss = c2d(continuous_time_ss,Ts,'tustin');
A = discrete_time_ss.A;
B = discrete_time_ss.B;
```

Per verificare la correttezza dell'operazione di discretizzazione, si possono analizzare gli autovalori della matrice A e il rango della matrice di raggiungibilità a tempo discreto.

```
lambda_i_dt = eig(A)
```

```
lambda_i_dt = 6×1
    0.9986
    0.9891
    0.9867
    0.9975
    0.9978
    0.9978
```

```
Mr = ctrb(A,B);
rk_Mr = rank(Mr)
```

```
rk_Mr = 6
```

I risultati ottenuti a tempo discreto sono coerenti con quelli ottenuti a tempo continuo: gli autovalori hanno valore molto prossimo all'unità, ciò significa che sono lenti, inoltre il rango della matrice di raggiungibilità rimane pari alla dimensione dello stato, che 6, perciò il sistema è completamente raggiungibile.

Espressione dei vincoli nelle coordinate del sistema linearizzato

Il progetto del controllore MPC è legato al punto di equilibrio scelto, perché questo viene visto come se fosse l'origine del sistema. Per questo motivo, si devono riesprimere i vincoli e la condizione iniziale nelle coordinate del sistema linearizzato.

```
x_ci_mpc = x_ci-x_eq;
x_eq_mpc = zeros(6,1);
u_eq_mpc = zeros(3,1);
x_max_mpc = x_max-x_eq;
x_min_mpc = x_min-x_eq;
u_max_mpc = u_max-u_eq;
u_min_mpc = u_min-u_eq;
```

Calcolo del Control Invariant Set

Un ingredente necessario per progettare il controllore MPC è il Control Invariant Set (CIS). Le dimensioni di questo insieme dipendono dalla scelta delle matrici Q ed R del costo quadratico. Il CIS è calcolato iterativamente, ma prima bisogna esprimere i vincoli del sistema nella forma $H_x\, x \leq hx, H_u\, u \leq hu$.

N.B.: se il punto di equilibrio non appartiene al CIS, l'esecuzione del codice viene interrotta.

```
% Vincoli su stato e ingresso, espressi nella forma di <=
Hx = [eye(6); -eye(6)];
hx = [x_max_mpc;-x_min_mpc];
Hu = [eye(3); -eye(3)];
hu = [u_max_mpc;-u_min_mpc];

% 1. Controllo veloce

% Valori sulla diagonale delle matrici quadrate Q ed R
Q_val_hard = 100;
R_val_hard = 1;
Q_hard = Q_val_hard.*eye(6);
R_hard = R_val_hard.*eye(3);
% Chiamata alla funzione che calcola il CIS
[G_hard,g_hard] = cis(A,B,x_eq_mpc,u_eq_mpc,Hx,hx,Hu,hu,Q_hard,R_hard);
```

```
ans =
"iterazione 1"
ans =
"iterazione 2"
ans =
"iterazione 3"
ans =
"iterazione 4"
ans =
"iterazione 5"
ans =
"iterazione 6"
ans =
"iterazione 7"
ans =
"iterazione 8"
ans =
"iterazione 9"
ans =
"iterazione 10"
ans =
"iterazione 11"
ans =
"iterazione 12"
ans =
"iterazione 13"
ans =
"iterazione 14"
ans =
"iterazione 15"
ans =
"iterazione 16"
ans =
"iterazione 17"
```

```
ans =
"iterazione 18"
ans =
"iterazione 19"
ans =
"iterazione 20"
ans =
"iterazione 21"
ans =
"iterazione 22"
ans =
"iterazione 23"
ans =
"iterazione 24"
ans =
"iterazione 25"
ans =
"iterazione 26"
ans =
"iterazione 27"
ans =
"iterazione 28"
ans =
"iterato 28 volte"
```

```
CIS_hard = Polyhedron(G_hard,g_hard);
% Stampa di alcune informazioni riguardanti il CIS appena calcolato
CIS_hard.display()
```

```
Polyhedron in R^6 with representations:
    H-rep (redundant)   : Inequalities  48 | Equalities   0
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
CIS_hard.isBounded()
```

```
ans = logical
   1
```

```
CIS_hard.isEmptySet()
```

```
ans = logical
   0
```

```
CIS_hard.isFullDim()
```

```
ans = logical
   1
```

```
CIS_hard.isInside(x_eq_mpc)
```

```
ans = logical
   1
```

```
if ~(CIS_hard.isInside(x_eq_mpc))
    error('Il punto di equilibrio non è contenuto nel Control Invariant Set!')
end

%{
% 2. Controllo economico
%% SERVONO PIù DI 120 ITERAZIONI
```

```
% Valori sulla diagonale delle matrici quadrate Q ed R
Q_val_soft = 1;
R_val_soft = 10;
Q_soft = Q_val_soft.*eye(6);
R_soft = R_val_soft.*eye(3);
% Chiamata alla funzione che calcola il CIS
[G_soft,g_soft] = cis(A,B,x_eq_mpc,u_eq_mpc,Hx,hx,Hu,hu,Q_soft,R_soft);
CIS_soft = Polyhedron(G_soft,g_soft);
% Stampa di alcune informazioni riguardanti il CIS appena calcolato
CIS_soft.display()
CIS_soft.isBounded()
CIS_soft.isEmptySet()
CIS_soft.isFullDim()
CIS_soft.isInside(x_ci_mpc)
%}
```

Dato che non è possibile visualizzare il grafico del poliedro, in quanto appartenente allo spazio 6D, si utilizzano delle funzioni di MPT3 per ricavare alcune proprietà di questo poliedro. Alcuni vincoli sono ridondanti, la regione individuata non è vuota ed è limitata, inoltre ha dimensione pari alla dimensione dello stato, che è 6. La condizione iniziale del problema non appartiene al CIS (questo è coerente, altrimenti avremmo poturo risolvere il problema del controllo con un semplice LQR).

Visualizzazione del CIS

Un modo per poter visualizzare il CIS è quello di dividere la regione in base alle stanze, in questo modo si avranno tre grafici 2D che mostrano la regione ammissibile per ogni stanza.

```
% 1. Controllo veloce

% Stanza 1
G1_hard = G_hard(:,[1,4]);
CIS1_hard = Polyhedron(G1_hard,g_hard)
```

```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  48 | Equalities   0
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
% Stanza 2
G2_hard = G_hard(:,[2,5]);
CIS2_hard = Polyhedron(G2_hard,g_hard)
```

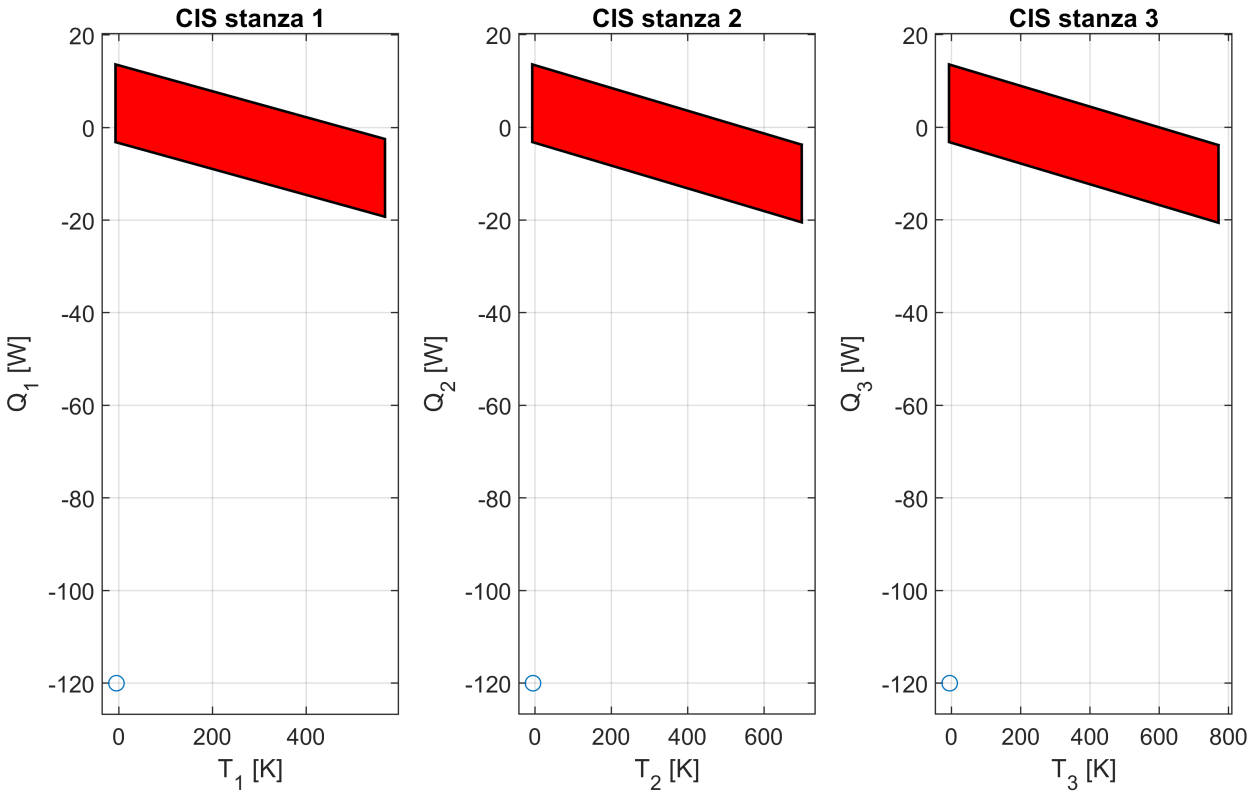
```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  48 | Equalities   0
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
% Stanza 3
G3_hard = G_hard(:,[3,6]);
CIS3_hard = Polyhedron(G3_hard,g_hard)
```

```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  48 | Equalities   0
```

```
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
h = figure();
subplot(1,3,1)
plot(x_ci_mpc(1),x_ci_mpc(4),'o')
hold on
CIS1_hard.plot()
title('CIS stanza 1');
xlabel('T_1 [K]');
ylabel('Q_1 [W]');
subplot(1,3,2)
plot(x_ci_mpc(2),x_ci_mpc(5),'o')
hold on
CIS2_hard.plot()
title('CIS stanza 2');
xlabel('T_2 [K]');
ylabel('Q_2 [W]');
subplot(1,3,3)
plot(x_ci_mpc(3),x_ci_mpc(6),'o')
hold on
CIS3_hard.plot()
title('CIS stanza 3');
xlabel('T_3 [K]');
ylabel('Q_3 [W]');
set(h,'Units','normalized','Position',[0 0 0.5 0.5]);
```




```
%{
```

```
% 2. Controllo economico

% Stanza 1
G1_soft = G_soft(:,[1,4]);
CIS1_soft = Polyhedron(G1_soft,g_soft)
% Stanza 2
G2_soft = G_soft(:,[2,5]);
CIS2_soft = Polyhedron(G2_soft,g_soft)
% Stanza 3
G3_soft = G_soft(:,[3,6]);
CIS3_soft = Polyhedron(G3_soft,g_soft)
figure
subplot(1,3,1)
CIS1_soft.plot()
hold on
plot(x_ci_mpc(1),x_ci_mpc(4),'o')
xlim([-20,800]);
ylim([-140,15]);
subplot(1,3,2)
CIS2_soft.plot()
hold on
plot(x_ci_mpc(2),x_ci_mpc(5),'o')
xlim([-20,800]);
ylim([-140,15]);
subplot(1,3,3)
CIS3_soft.plot()
hold on
plot(x_ci_mpc(3),x_ci_mpc(6),'o')
xlim([-20,800]);
ylim([-140,15]);
%}
```

Calcolo del N-step Controllable Set del Control Invariant Set

...

N.B.: se il punto che rappresenta la condizione iniziale non appartiene all'N-step set, l'esecuzione del codice viene interrotta (per la proprietà di recursive feasibility: se la condizione iniziale del problema è fattibile, allora il controllore MPC risulta fattibile; al contrario se la condizione iniziale non è fattibile, non lo sarà nemmeno il controllore MPC).

```
% Orizzonte di predizione
N = 80;

% 1. Controllo veloce

[N_steps_H_hard, N_steps_h_hard] = controllable_set(Hx,hx,Hu,hu,G_hard,g_hard,A,B,N);
```

```
ans =
"iterazione 1"
ans =
"iterazione 2"
ans =
"iterazione 3"
```

```
ans =
"iterazione 4"
ans =
"iterazione 5"
ans =
"iterazione 6"
ans =
"iterazione 7"
ans =
"iterazione 8"
ans =
"iterazione 9"
ans =
"iterazione 10"
ans =
"iterazione 11"
ans =
"iterazione 12"
ans =
"iterazione 13"
ans =
"iterazione 14"
ans =
"iterazione 15"
ans =
"iterazione 16"
ans =
"iterazione 17"
ans =
"iterazione 18"
ans =
"iterazione 19"
ans =
"iterazione 20"
ans =
"iterazione 21"
ans =
"iterazione 22"
ans =
"iterazione 23"
ans =
"iterazione 24"
ans =
"iterazione 25"
ans =
"iterazione 26"
ans =
"iterazione 27"
ans =
"iterazione 28"
ans =
"iterazione 29"
ans =
"iterazione 30"
ans =
"iterazione 31"
ans =
"iterazione 32"
ans =
"iterazione 33"
ans =
"iterazione 34"
ans =
"iterazione 35"
```

```
ans =
"iterazione 36"
ans =
"iterazione 37"
ans =
"iterazione 38"
ans =
"iterazione 39"
ans =
"iterazione 40"
ans =
"iterazione 41"
ans =
"iterazione 42"
ans =
"iterazione 43"
ans =
"iterazione 44"
ans =
"iterazione 45"
ans =
"iterazione 46"
ans =
"iterazione 47"
ans =
"iterazione 48"
ans =
"iterazione 49"
ans =
"iterazione 50"
ans =
"iterazione 51"
ans =
"iterazione 52"
ans =
"iterazione 53"
ans =
"iterazione 54"
ans =
"iterazione 55"
ans =
"iterazione 56"
ans =
"iterazione 57"
ans =
"iterazione 58"
ans =
"iterazione 59"
ans =
"iterazione 60"
ans =
"iterazione 61"
ans =
"iterazione 62"
ans =
"iterazione 63"
ans =
"iterazione 64"
ans =
"iterazione 65"
ans =
"iterazione 66"
ans =
"iterazione 67"
```

```
ans =
"iterazione 68"
ans =
"iterazione 69"
ans =
"iterazione 70"
ans =
"iterazione 71"
ans =
"iterazione 72"
ans =
"iterazione 73"
ans =
"iterazione 74"
ans =
"iterazione 75"
ans =
"iterazione 76"
ans =
"iterazione 77"
ans =
"iterazione 78"
ans =
"iterazione 79"
ans =
"iterazione 80"
```

```
N_steps_set_hard = Polyhedron('A', N_steps_H_hard, 'b', N_steps_h_hard);
N_steps_set_hard.isBounded()
```

```
ans = logical
   1
```

```
N_steps_set_hard.isEmptySet()
```

```
ans = logical
   0
```

```
N_steps_set_hard.isFullDim()
```

```
ans = logical
   1
```

```
N_steps_set_hard.isInside(x_ci_mpc)
```

```
ans = logical
   0
```

```
if ~(N_steps_set_hard.isInside(x_ci_mpc))
    %error('N-step set non contiene il punto iniziale, controllo non fattibile!')
    warning('N-step set non contiene il punto iniziale, controllo non fattibile!')
end
```

```
Warning: N-step set non contiene il punto iniziale, controllo non fattibile!
```

```
% Stanza 1
N_steps_H1_hard = N_steps_H_hard(:,[1,4]);
NsH1_hard = Polyhedron(N_steps_H1_hard,N_steps_h_hard)
```

```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  49 | Equalities   0
```

```
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

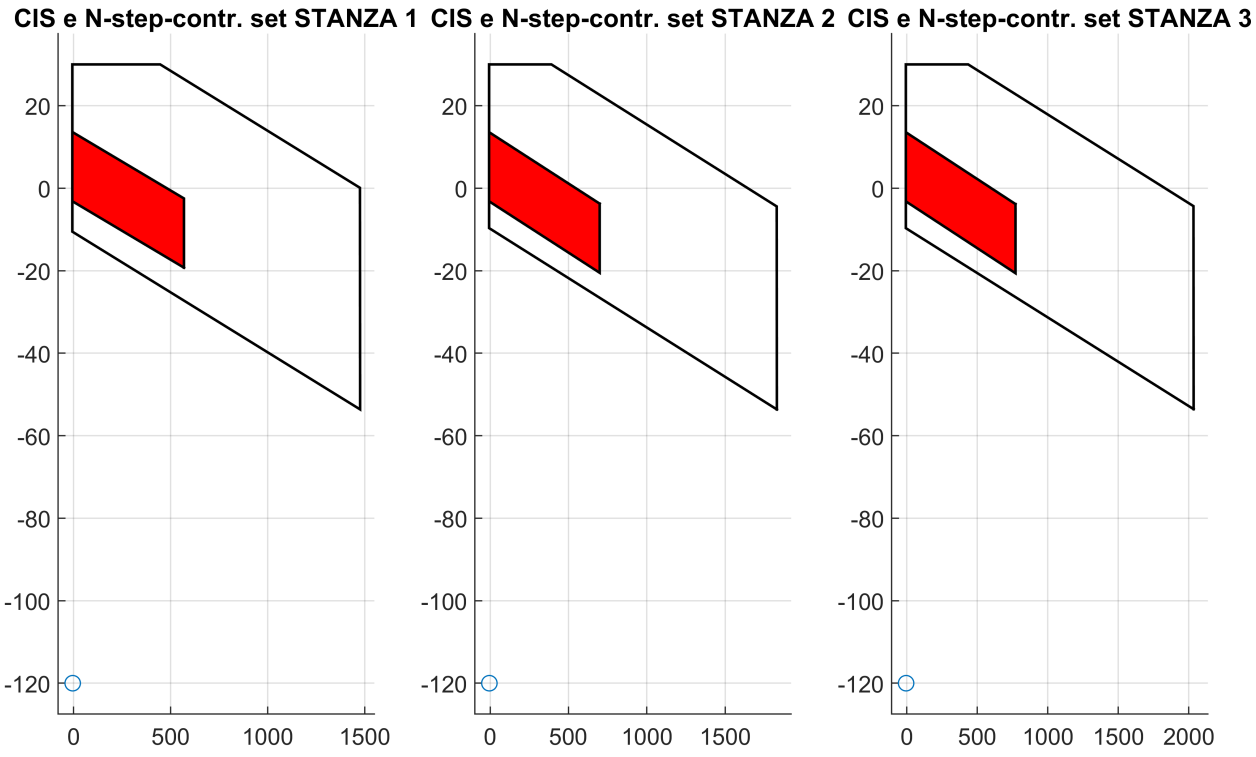
```
% Stanza 2
N_steps_H2_hard = N_steps_H_hard(:,[2,5]);
NsH2_hard = Polyhedron(N_steps_H2_hard,N_steps_h_hard)
```

```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  49 | Equalities   0
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
% Stanza 3
N_steps_H3_hard = N_steps_H_hard(:,[3,6]);
NsH3_hard = Polyhedron(N_steps_H3_hard,N_steps_h_hard)
```

```
Polyhedron in R^2 with representations:
    H-rep (redundant)   : Inequalities  49 | Equalities   0
    V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
h = figure();
subplot(1,3,1)
NsH1_hard.plot('Alpha',0)
hold on
plot(x_ci_mpc(1),x_ci_mpc(4),'o')
CIS1_hard.plot()
title('CIS e N-step-contr. set STANZA 1');
subplot(1,3,2)
NsH2_hard.plot('Alpha',0)
hold on
plot(x_ci_mpc(2),x_ci_mpc(5),'o')
CIS2_hard.plot()
title('CIS e N-step-contr. set STANZA 2');
subplot(1,3,3)
NsH3_hard.plot('Alpha',0)
hold on
plot(x_ci_mpc(3),x_ci_mpc(6),'o')
CIS3_hard.plot()
title('CIS e N-step-contr. set STANZA 3');
set(h,'Units','normalized','Position',[0 0 0.5 0.5]);
```




```
%{
% 2. Controllo economico

[N_steps_H_soft, N_steps_h_soft] = controllable_set(Hx, hx, Hu, hu, G_soft, g_soft, A, B, N);
N_steps_set_soft = Polyhedron('A', N_steps_H_soft, 'b', N_steps_h_soft);
N_steps_set_soft.isBounded()
N_steps_set_soft.isEmptySet()
N_steps_set_soft.isFullDim()
N_steps_set_soft.isInside(x_ci_mpc)
% Stanza 1
N_steps_H1_soft = N_steps_H_soft(:,[1,4]);
NsH1_soft = Polyhedron(N_steps_H1_soft,N_steps_h_soft)
% Stanza 2
N_steps_H2_soft = N_steps_H_soft(:,[2,5]);
NsH2_soft = Polyhedron(N_steps_H2_soft,N_steps_h_soft)
% Stanza 3
N_steps_H3_soft = N_steps_H_soft(:,[3,6]);
NsH3_soft = Polyhedron(N_steps_H3_soft,N_steps_h_soft)
figure
subplot(1,3,1)
NsH1_soft.plot('Alpha',0)
hold on
CIS1_soft.plot()
plot(x_ci_mpc(1),x_ci_mpc(4),'o')
xlim([-200,1000]);
ylim([-500,150]);
subplot(1,3,2)
```

```
NsH2_soft.plot('Alpha',0)
hold on
CIS2_soft.plot()
plot(x_ci_mpc(2),x_ci_mpc(5),'o')
xlim([-200,1000]);
ylim([-500,150]);
subplot(1,3,3)
NsH3_soft.plot('Alpha',0)
hold on
CIS3_soft.plot()
plot(x_ci_mpc(3),x_ci_mpc(6),'o')
xlim([-200,1000]);
ylim([-500,150]);
%}
```

Design del controllore MPC e simulazione

...

```
% Dimensioni matrici
nx = size(A,1);
nu = size(B,2);

%   Numero di step simulati
T_sim = 60;

% Log stati e ingresso sistema
x_log = zeros(nx,T_sim+1);
u_log = zeros(nu,T_sim);
flags = zeros(1,T_sim);
x_log(:,1) = x_ci_mpc;

% 1. Controllo veloce
% Calcolo delle matrici necessarie per quadprog
[H_hard,f_hard,Aineq_hard,bineq_hard] = quadprogmatrix(N,nx,nu,A,B,Q_hard,R_hard,x_ci_mpc,Hx,hx
% TODO: controllare perchè nella versione originale viene posizionato fuori dal for e come inpu
% si passa x_ref_lin e u_ref_lin al posto di x_ci_mpc. Forse per questo ottiene cose diverse?

for tt = 1:T_sim
    %   Stato sistema linearizzato
    x_lin = x_log(:,tt) - x_eq;
    x_lin_shifted = x_lin - x_eq_mpc; % TODO: va messo x_ci_mpc oppure (x_eq_mpc) <==

    %   Impostazioni MPC relative alla condizione iniziale
    % f = f_hard * x_lin_shifted;
    % b_ineq = bineq_hard ;%- mpc.b_ineq_x0_factor * x_lin_shifted; % TODO

    %   Risoluzione problema di ottimizzazione
    % [delta_u_seq,~,exitflag] = quadprog(H_hard,f,Aineq_hard,b_ineq);
    [delta_u_seq,~,exitflag] = quadprog(H_hard,f_hard,Aineq_hard,bineq_hard);

    flags(tt) = exitflag;

    %   Risposta del sistema
```

```
    % u_log(:,tt) = u_ref + delta_u_seq;
    u_ottim = zeros(3,1);
    u_ottim(1) = delta_u_seq(1);
    u_ottim(2) = delta_u_seq(N+1);
    u_ottim(3) = delta_u_seq(2*N+1);
    u_log(:,tt) = u_eq + u_ottim; % TODO: controllare se va bene sostituzione con ==>(u_eq) and

    dxdt = @(t,x) uploadsystem(t,x,u_ottim,k_bar,k_ext,T_ext,C,tau);
    [~,xx] = ode45(dxdt,[0 Ts],x_log(:,tt));
    x_log(:,tt+1) = xx(end,:)';
end
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.
```

```
Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in 
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.
```

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

```
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.
```

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

```
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.
```

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

```
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
%{
VERSIONE PRECEDENTE DEL CODICE

% [H_hard,f_hard,Aineq_hard,bineq_hard] = quadprogmatrix(N,nx,nu,A,B,Q_hard,R_hard,x_ci_mpc,Hx,
% u_hard = quadprog(H_hard,f_hard,Aineq_hard,bineq_hard)
%
% u_shifted_hard = zeros(nu*N,1);
% for i = 0:1:N-1
%     u_shifted_hard(i*nu+1) = u_hard(i*nu+1)+u_eq(1);
%     u_shifted_hard(i*nu+2) = u_hard(i*nu+2)+u_eq(2);
%     u_shifted_hard(i*nu+3) = u_hard(i*nu+3)+u_eq(3);
% end
% u_shifted_hard
%
```

```
% u_hard_matr = zeros(nu,N);
% for i = 0:1:N-1
%     u_hard_matr(1,i+1) = u_hard(i*nu+1);
%     u_hard_matr(2,i+1) = u_hard(i*nu+2);
%     u_hard_matr(3,i+1) = u_hard(i*nu+3);
% end
% u_hard_matr

for i=1:1:N
    sprintf("Iterazione %d dell'ottimizzazione del problema MPC",i)
    if i==1
        x_ci_mpc_hard =  x_ci_mpc;
    end
    [H_hard,f_hard,Aineq_hard,bineq_hard] = quadprogmatrix(N,nx,nu,A,B,Q_hard,R_hard,x_ci_mpc_h
    u_hard = quadprog(H_hard,f_hard,Aineq_hard,bineq_hard);
    u_hard_matr = zeros(nu,N);
    for j = 0:1:N-1
        u_hard_matr(1,j+1) = u_hard(j*nu+1);
        u_hard_matr(2,j+1) = u_hard(j*nu+2);
        u_hard_matr(3,j+1) = u_hard(j*nu+3);
    end
    u0_hard = u_hard_matr(:,1);
    % Aggiornamento di x_ci_mpc per l'iterazione i+1
    x_ci_mpc_hard = A*x_ci_mpc_hard + B*u0_hard;

    dxdt = @(t,x) uploadsystem(t,x,u0_hard,k_bar,k_ext,T_ext,C,tau);
    % [tilde,xx] = ode45(dxdt,[0 Ts],x_log(:,tt));
    % x_log(:,tt+1) = xx(end,:)';
end


u_shifted_hard = zeros(nu,N);
for i = 0:1:N-1
    u_shifted_hard(1,i+1) = u_hard_matr(1,i+1)+u_eq(1);
    u_shifted_hard(2,i+1) = u_hard_matr(2,i+1)+u_eq(2);
    u_shifted_hard(3,i+1) = u_hard_matr(3,i+1)+u_eq(3);
end
u_shifted_hard
%}

%{
% 2. Controllo economico

% [H_soft,f_soft,Aineq_soft,bineq_soft] = quadprogmatrix(N,nx,nu,A,B,Q_soft,R_soft,x_ci_mpc,Hx_

% u_soft = quadprog(H_soft,f_soft,Aineq_soft,bineq_soft)

% u_shifted_soft = zeros(nu*N,1);
% for i = 0:1:N-1
%     u_shifted_soft(i*nu+1) = u_soft(i*nu+1)+u_eq(1);
%     u_shifted_soft(i*nu+2) = u_soft(i*nu+2)+u_eq(2);
%     u_shifted_soft(i*nu+3) = u_soft(i*nu+3)+u_eq(3);
% end
% u_shifted_soft
```

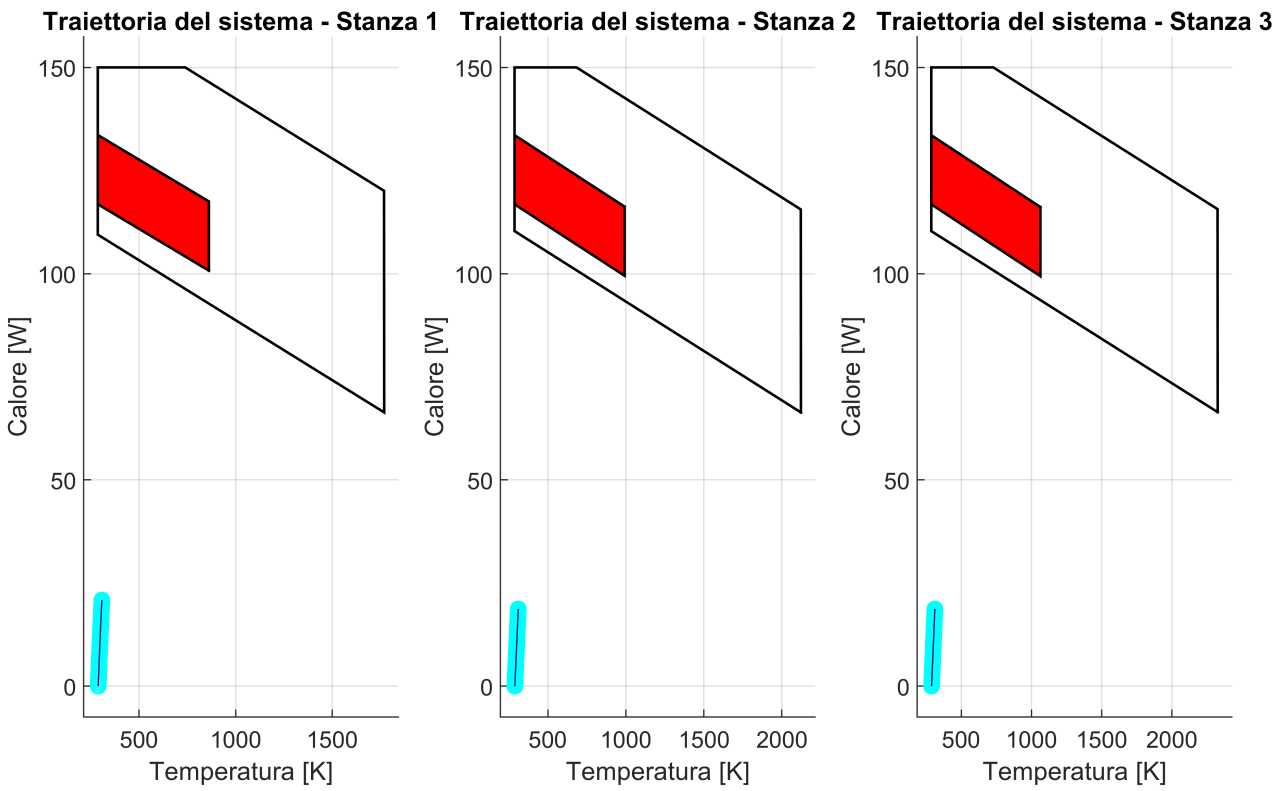
```
% u_soft_matr = zeros(nu,N);
% for i = 0:1:N-1
%     u_soft_matr(1,i+1) = u_soft(i*nu+1);
%     u_soft_matr(2,i+1) = u_soft(i*nu+2);
%     u_soft_matr(3,i+1) = u_soft(i*nu+3);
% end
% u_soft_matr

for i=1:1:N
    sprintf("Iterazione %d dell'ottimizzazione del problema MPC",i)
    if i==1
        x_ci_mpc_soft =  x_ci_mpc;
    end
    [H_soft,f_soft,Aineq_soft,bineq_soft] = quadprogmatrix(N,nx,nu,A,B,Q_soft,R_soft,x_ci_mpc_s
    u_soft = quadprog(H_soft,f_soft,Aineq_soft,bineq_soft);
    u_soft_matr = zeros(nu,N);
    for j = 0:1:N-1
        u_soft_matr(1,j+1) = u_soft(j*nu+1);
        u_soft_matr(2,j+1) = u_soft(j*nu+2);
        u_soft_matr(3,j+1) = u_soft(j*nu+3);
    end
    u0_soft = u_soft_matr(:,1)
    % Aggiornamento di x_ci_mpc per l'iterazione i+1
    x_ci_mpc_soft = A*x_ci_mpc_soft + B*u0_soft;
end

u_shifted_soft = zeros(nu,N);
for i = 0:1:N-1
    u_shifted_soft(1,i+1) = u_soft_matr(1,i+1)+u_eq(1);
    u_shifted_soft(2,i+1) = u_soft_matr(2,i+1)+u_eq(2);
    u_shifted_soft(3,i+1) = u_soft_matr(3,i+1)+u_eq(3);
end
u_shifted_soft
%}
```

Analisi delle prestazioni del controllore MPC progettato

...[prova plot]

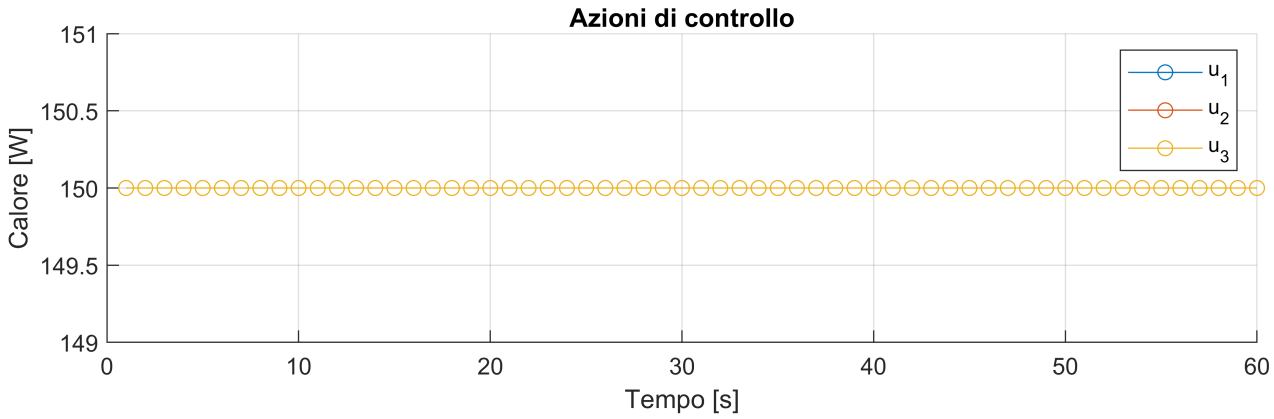
```
% Shift dei poliedri
CIS_shifted_1 = CIS1_hard + x_eq([1,4]);
Np_step_set_shifted_1 = NsH1_hard + x_eq([1,4]);
CIS_shifted_2 = CIS2_hard + x_eq([2,5]);
Np_step_set_shifted_2 = NsH2_hard + x_eq([2,5]);
CIS_shifted_3 = CIS3_hard + x_eq([3,6]);
Np_step_set_shifted_3 = NsH3_hard + x_eq([3,6]);
x_log_p = x_log + x_eq; % TODO aggiunta, non so se è corretta

h = figure();
subplot(1,3,1)
hold on
scatter(x_log_p(1,:),x_log_p(4,:),'cyan');
Np_step_set_shifted_1.plot('Alpha',0);
```

```
title('Traiettoria del sistema - Stanza 1');
xlabel('Temperatura [K]');
ylabel('Calore [W]');
CIS_shifted_1.plot()
plot(x_log_p(1,:),x_log_p(4,:),'Color',[0 0 0.5])

subplot(1,3,2)
hold on
scatter(x_log_p(2,:),x_log_p(5,:),'cyan');
Np_step_set_shifted_2.plot('Alpha',0);
title('Traiettoria del sistema - Stanza 2');
xlabel('Temperatura [K]');
ylabel('Calore [W]');
CIS_shifted_2.plot()
plot(x_log_p(2,:),x_log_p(5,:),'Color',[0 0 0.5])

subplot(1,3,3)
hold on
scatter(x_log_p(3,:),x_log_p(6,:),'cyan');
Np_step_set_shifted_3.plot('Alpha',0);
title('Traiettoria del sistema - Stanza 3');
xlabel('Temperatura [K]');
ylabel('Calore [W]');
CIS_shifted_3.plot()
plot(x_log_p(3,:),x_log_p(6,:),'Color',[0 0 0.5])
set(h,'Units','normalized','Position',[0 0 0.5 0.5]);
```

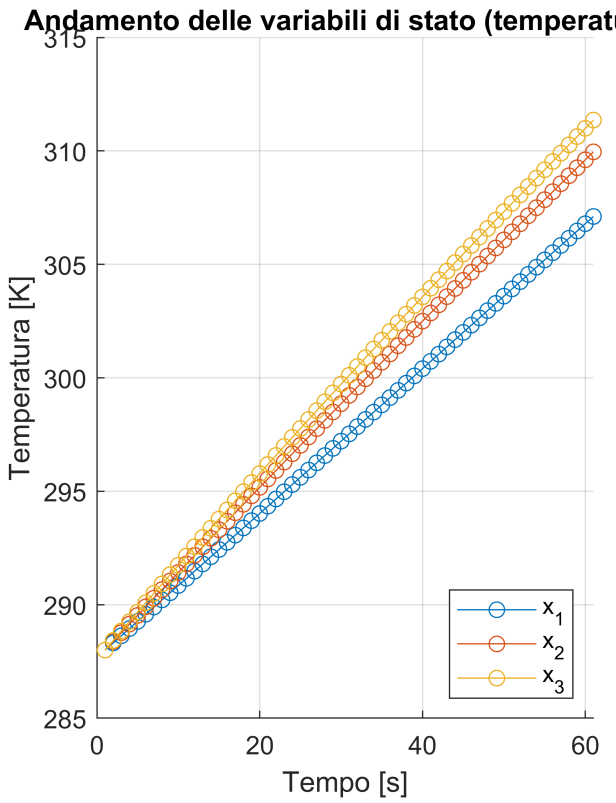


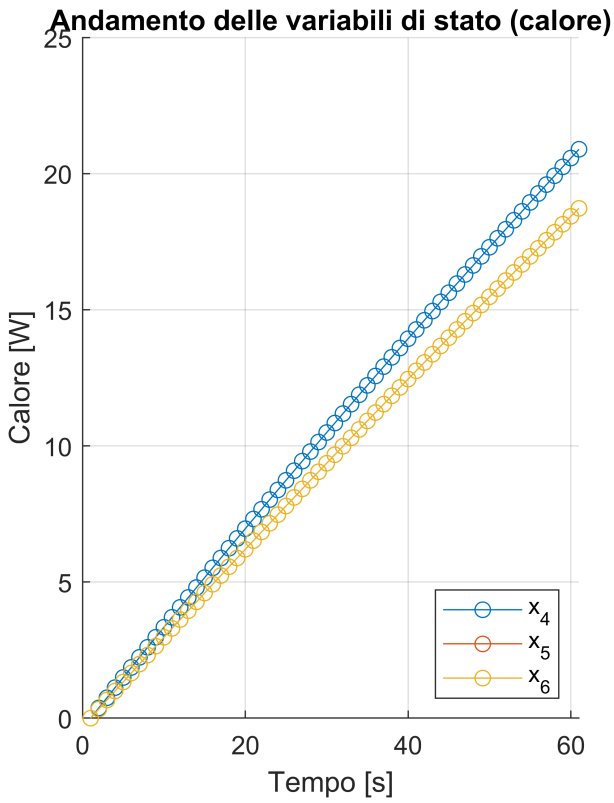

```
h = figure();
```

```
hold on
grid on
title('Azioni di controllo')
plot (1:1:T_sim,u_log(1,:),'-o')
plot (1:1:T_sim,u_log(2,:),'-o')
plot (1:1:T_sim,u_log(3,:),'-o')
xlabel('Tempo [s]');
ylabel('Calore [W]');
legend('u_1','u_2','u_3')
set(h,'Units','normalized','Position',[0 0 1 .5]);
```




```
h=figure();
subplot(1,2,1)
title('Andamento delle variabili di stato (temperatura)')
hold on
grid on
plot(1:1:T_sim+1,x_log_p(1,:),'-o')
plot(1:1:T_sim+1,x_log_p(2,:),'-o')
plot(1:1:T_sim+1,x_log_p(3,:),'-o')
xlabel('Tempo [s]');
ylabel('Temperatura [K]');
legend('x_1','x_2','x_3','Location','southeast')
subplot(1,2,2)
title('Andamento delle variabili di stato (calore)')
hold on
grid on
plot(1:1:T_sim+1,x_log_p(4,:),'-o')
plot(1:1:T_sim+1,x_log_p(5,:),'-o')
plot(1:1:T_sim+1,x_log_p(6,:),'-o')
xlabel('Tempo [s]');
ylabel('Calore [W]');
legend('x_4','x_5','x_6','Location','southeast')
set(h,'Units','normalized','Position',[0 0 0.5 0.5]);
```





Funzioni

Di seguito, si riporta la definizione delle funzioni utilizzate nelle sezioni precedenti.

1. Calcolo del Control Invariant Set

```
function [G,g] = cis(A,B,x_ref,u_ref,Fx,fx,Fu,fu,Q,R)
% Controllore LQR
K = -dlqr(A,B,Q,R);
% Matrice A del sistema controllato con LQR
A_lqr = A+B*K;
% Vincoli su stato e ingresso
F = [Fx; Fu*K];
f = [fx; fu + Fu*(K*x_ref - u_ref)];
% Calcolo CIS (G*x<=g)
% Inizializzazione
CIS_poly_prev = Polyhedron(); % Poliedro vuoto
CIS_poly_curr = Polyhedron(F,f); % Poliedro fornito dai vincoli appena trovati
i=0; %%% DEBUG
while CIS_poly_prev.isEmptySet || CIS_poly_prev ~= CIS_poly_curr
    % Memorizzo il vecchio set candidato
    CIS_poly_prev = CIS_poly_curr;
    % Calcolo il nuovo set candidato (G_hat*x<=g_hat)
    G_hat = [CIS_poly_curr.A * A_lqr; F]; % tutti i vincoli stato
    g_hat = [CIS_poly_prev.b + CIS_poly_curr.A*B*(K*x_ref - u_ref); f];
    CIS_poly_curr = Polyhedron(G_hat,g_hat); % Poliedro con nuovi vincoli
    %%% DEBUG
    i=i+1;
```

```
    sprintf("iterazione %d",i)
    %{
    if i == 100
        break
    end
    %}
    %%%
end
sprintf("iterato %d volte",i)
G = CIS_poly_curr.A;
g = CIS_poly_curr.b;
end
```

2. Calcolo del N-step Controllable Set del Control Invariant Set

```
function [H_nsteps, h_nsteps] = controllable_set(Hx, hx, Hu, hu, H_target, h_target, A, B, N)
n = size(A,2);
m = size(B,2);

% Candidato iniziale
H_ii_steps = H_target;
h_ii_steps = h_target;

for li = 1:N
    % Computazione in R^(n+m)
    sprintf("iterazione %d", li) %%% DEBUG
    temp = Polyhedron('A', [H_ii_steps*A H_ii_steps*B; ...
        zeros(size(Hu,1),n) Hu], 'b', [h_ii_steps; hu]);

    % Proiezione in R^n
    temp = projection(temp, 1:n);
    temp.minHRep();

    % Intersezione con X := {x | Hx*x <= hx}
    H_ii_steps = [temp.A; Hx];
    h_ii_steps = [temp.b; hx];
end

H_nsteps = H_ii_steps;
h_nsteps = h_ii_steps;
end
```

3. Calcolo delle matrici richieste da quadrog

```
function [H,f,Aineq,bineq] = quadprogmatrix(N,nx,nu,A,B,Q,R,x_ci,Hx,hx,Hu,hu)
% Definizione della dinamica del sistema
A_italic = zeros(nx*(N+1),nx);
B_italic = zeros(nx*(N+1),nu*N);
B_italic_col = zeros(nx*N,nu);
for i = 0:1:N
    if(i==0)
        A_italic = eye(nx);
        B_italic_col(1:nx,:) = zeros(nx,nu);
    elseif(i==1)
```

```
        A_italic = [A_italic;A.^i];
        B_italic_col((i*nx)+1:(i+1)*nx,:) = B;
    else
        A_italic = [A_italic;A.^i];
        B_italic_col((i*nx)+1:(i+1)*nx,:) = (A.^(i-1))*B;
    end
end
for i = 0:1:N-1
    B_italic(:,(i*nu)+1:(i+1)*nu) = B_italic_col;
    B_italic_col = [zeros(nx,nu); B_italic_col(1:size(B_italic_col,1)-nx,:)];
end
% Trasformazione del costo
[K,P] = dlqr(A,B,Q,R);
Q_italic = zeros(nx*(N+1),nx*(N+1));
R_italic = zeros(nu*N,nu*N);
for i = 0:1:N-1
    Q_italic((i*nx)+1:(i*nx)+nx,(i*nx)+1:(i*nx)+nx) = diag(diag(Q));
    R_italic((i*nu)+1:(i*nu)+nu,(i*nu)+1:(i*nu)+nu) = diag(diag(R));
end
Q_italic((N*nx)+1:(N*nx)+nx,(N*nx)+1:(N*nx)+nx) = P;

prod = B_italic' * Q_italic * B_italic;

% Approssimazione di cifre non significative per incrementare la velocità
% di convergenza di quadprog (rendendo prod una matrice quadrata)
for i=1:length(prod)
    for j=1:length(prod)
            prod(i,j)=round(prod(i,j),5,"significant");
    end
end

H = R_italic + prod;

f = (x_ci' * A_italic' * Q_italic * B_italic)';

% Calcolo matrici dei vincoli sugli ingressi
Hu_italic = zeros(2*nu*N,N*nu);
hu_italic = zeros(2*nu*N,1);
for i = 0:1:N-1
    if(i==0)
        hu_italic = hu;
        Hu_italic(1:2*nu,:) = [Hu zeros(2*nu,(N-1)*nu)];
    else
        hu_italic = [hu_italic;hu];
        if(i==N-1)
            Hu_italic((i*2*nu)+1:(i*2*nu)+(2*nu),:) = [zeros(2*nu,i*nu) Hu];
        else
            Hu_italic((i*2*nu)+1:(i*2*nu)+(2*nu),:) = [zeros(2*nu,i*nu) Hu zeros(2*nu,(N-i-1)*n
        end
    end
end

% Calcolo matrici dei vincoli sullo stato
Hx_italic = zeros(2*nx*(N+1),(N+1)*nx);
```

```
hx_italic = zeros(2*nx*(N+1),1);
for i = 0:1:N
    if(i==0)
        hx_italic = hx;
        Hx_italic(1:2*nx,:) = [Hx zeros(2*nx,N*nx)];
    else
        hx_italic = [hx_italic;hx];
        if(i==N)
            Hx_italic((i*2*nx)+1:(i*2*nx)+(2*nx),:) = [zeros(2*nx,N*nx) Hx];
        else
            Hx_italic((i*2*nx)+1:(i*2*nx)+(2*nx),:) = [zeros(2*nx,i*nx) Hx zeros(2*nx,(N-i)*nx)
        end
    end
end

Aineq = [Hu_italic; Hx_italic * B_italic];
bineq = [hu_italic; hx_italic - Hx_italic * A_italic * x_ci];
end
```

4. Aggiornamento del sistema con la prima azione di controllo ottenuta dal quadprog

```
function x_dot = uploadsystem(t,x,u,k_bar,k_ext,T_ext,C,tau)

k12 = k_bar(1)+(4/(1+exp(-0.5*norm(x(1)-x(2),2))^2));
k13 = k_bar(2)+(4/(1+exp(-0.5*norm(x(1)-x(3),2))^2));
k23 = k_bar(3)+(4/(1+exp(-0.5*norm(x(2)-x(3),2))^2));

% Equazioni di stato
x_dot = zeros(6,1);
x_dot(1) = (x(4) -k12*(x(1)-x(2)) -k13*(x(1)-x(3)) -k_ext*(x(1)-T_ext))/C(1);
x_dot(2) = (x(5) +k12*(x(1)-x(2)) -k23*(x(2)-x(3)) -k_ext*(x(2)-T_ext))/C(2);
x_dot(3) = (x(6) +k13*(x(1)-x(3)) +k23*(x(2)-x(3)) -k_ext*(x(3)-T_ext))/C(3);
x_dot(4) = (u(1)-x(4))/tau(1);
x_dot(5) = (u(2)-x(5))/tau(2);
x_dot(6) = (u(3)-x(6))/tau(3);

end
```