Open Loop Optimization of LTI systems

Trym Arve L. Gabrielsen

March 5, 2023

Contents

Abstract

An informal, simple summary of open loop optimization methods for LTI systems, with MATLAB code examples.

1 Open Loop Optimization for Discrete LTI Systems

Consider the discrete time LTI system model

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k. \tag{1}$$

with initial state $\boldsymbol{x}_0$. We also assume that we want the system to reach some desired end state $\boldsymbol{x}_f$. In this document, we use the following symbols to simplify notation.

The input and state trajectories are denoted using

$$\boldsymbol{X}_N = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}, \quad \boldsymbol{U}_N = \begin{bmatrix} \boldsymbol{u}_0 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} \tag{2}$$

where $\boldsymbol{x}_k \in \mathbb{R}^{nx}$ and $\boldsymbol{u}_k \in \mathbb{R}^{nu}$ is the state and input vector at time k, respectively. For the cost matrices, we use

$$\mathcal{Q}_N = \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_N \end{bmatrix}, \mathcal{R}_N = \begin{bmatrix} R_0 & & \\ & \ddots & \\ & & R_{N-1} \end{bmatrix}, \tag{3}$$

where $Q_k \in \mathbb{R}^{nx \times nx}$ and $R_k \in \mathbb{R}^{nu \times nu}$ are the state and input vector costs, respectively, such that the state cost f_k^x and input cost f_k^u at time k are

$$f_k^x = \boldsymbol{x}_k^\top Q_k \boldsymbol{x}_k, \quad f_k^u = \boldsymbol{u}_k^\top R_k \boldsymbol{u}_k. \tag{4}$$

For the MATLAB examples in this section, we use the following:

Listing 1: MATLAB Example - LTI system

```
% Model:
% ( x_{k+1} = A*x_{k} + B*u_{k} )
A = [    1        0      0     ;
         0        0      1     ;
         0.1    -0.76    2     ];
B = [  2          ;
       0.0        ;
```

```
         0.1    ];

[nx,nu] = size(B);

% Initial state
x0 = [   4   ;
         5   ;
         8   ];

% Desired end state
xf = [ 2.3   ;
         1   ;
        -1   ];

% Bounds on input:
ulb = -20*ones(nu,1);
uub = 20*ones(nu,1);

% Horizon
N = 10;
```

1.1 Reduced Space

In the "Reduced Space" approach we only optimize over the control variables, that is, only the control variables are part of the decision variables. When diving into the world of nonlinear dynamic optimization, this approach is reminiscent of the "single shooting" method. The state at time k can then be found as

$$
\begin{aligned}
\boldsymbol{x}_1 =& A\boldsymbol{x}_0 + B\boldsymbol{u}_0 \\
\boldsymbol{x}_2 =& A\boldsymbol{x}_1 + B\boldsymbol{u}_1 \\
=& A(A\boldsymbol{x}_0 + B\boldsymbol{u}_0) + B\boldsymbol{u}_1 \\
=& A^2\boldsymbol{x}_0 + AB\boldsymbol{u}_0 + B\boldsymbol{u}_1 \\
\boldsymbol{x}_3 =& A\boldsymbol{x}_2 + B\boldsymbol{u}_2 \\
=& A(A^2\boldsymbol{x}_0 + AB\boldsymbol{u}_0 + B\boldsymbol{u}_1) + B\boldsymbol{u}_2 \\
=& A^3\boldsymbol{x}_0 + A^2B\boldsymbol{u}_0 + AB\boldsymbol{u}_1 + B\boldsymbol{u}_2 \\
\vdots \\
\boldsymbol{x}_k =& A^k\boldsymbol{x}_0 + \sum_{i=1}^{k} A^{k-i}B\boldsymbol{u}_{i-1}.
\end{aligned}
\tag{5}
$$

Let's write (5) on matrix form;

$$
\boldsymbol{x}_k = \underbrace{A^k}_{A_{red,k}} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} A^{k-1}B & A^{k-2}B & \dots & AB & B \end{bmatrix}}_{B_{red,k}} \boldsymbol{U}_k.
\tag{6}
$$

1.1.1 Desired End State, $\boldsymbol{x}_f$

If the only requirements of the state trajectory is that the end state is some desired final state, $\boldsymbol{x}_f$, then one can formulate the optimization problem using (6):

$$
\begin{aligned}
\min_{\boldsymbol{U}_N} \quad & f(\boldsymbol{U}_N) \\
\text{s.t.} \quad & \boldsymbol{x}_f = A_{red,N}\boldsymbol{x}_0 + B_{red,N}\boldsymbol{U}_N \\
& \underline{\boldsymbol{U}}_N \leq \boldsymbol{U}_N \leq \overline{\boldsymbol{U}}_N.
\end{aligned}
\tag{7}
$$

Notice that $\boldsymbol{x}_f = A_{red,N}\boldsymbol{x}_0 + B_{red,N}\boldsymbol{U}_N$ can be implemented as an equality constraint on the general form: $A_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$ (or in this case: $A_{eq}\boldsymbol{U}_N = \boldsymbol{b}_{eq}$) , by using;

$$
\begin{aligned}
A_{eq} &= B_{red,N} \\
\boldsymbol{b}_{eq} &= \boldsymbol{x}_f - A_{red,N}\boldsymbol{x}_0.
\end{aligned}
\tag{8}
$$

If we say that the input-cost is

$$f(\boldsymbol{U}_N) = \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N, \tag{9}$$

making problem (7) a QP, it can be implemented as:

Listing 2: Implementation of the Reduced Space without state constraints

```
% Get A_red_N and B_red_N:
[A_red_N, B_red_N] = ReducedSpace_k(A,B,N);

% Apply desired end state via equality constraints:
Aeq = B_red_N;
beq = xf-A_red_N*x0;

% Input limits:
Ulb = kron(ones(N,1),ulb);
Uub = kron(ones(N,1),uub);

% Solve:
U = quadprog(2*eye(N*nu),[],[],[],Aeq,beq,Ulb,Uub);

function [A_red_k, B_red_k] = ReducedSpace_k(A,B,k)
    A_red_k = A;
    B_red_k = B;
    for i = 1:k-1
        B_red_k = [A_red_k*B B_red_k];
        A_red_k = A*A_red_k;
    end
end
```
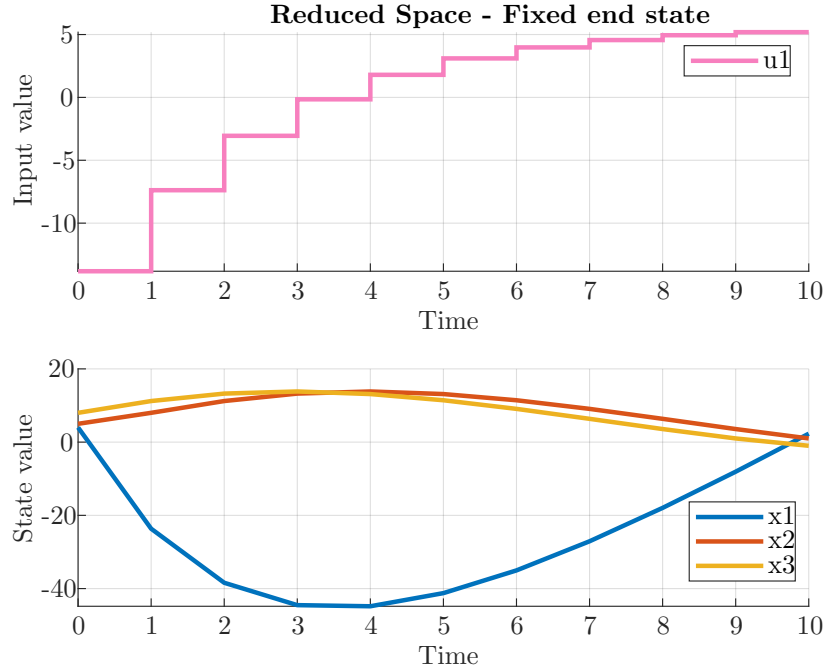
which gives the output:

Figure 1: Result of Listing 2, Reduced Space Approach with only end state constraints.

3

1.1.2 Constraints and Cost of State Trajectory

If there are requirements on the state trajectory, then all $\boldsymbol{x}_{1,\dots,N}$ need to be described in the optimization problem. One can describe the state trajectory as;

$$
\boldsymbol{X}_N =
\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}
=
\begin{bmatrix}
A_{red,1}\boldsymbol{x}_0 + B_{red,1}\boldsymbol{U}_1 \\
A_{red,2}\boldsymbol{x}_0 + B_{red,2}\boldsymbol{U}_2 \\
\vdots \\
A_{red,N}\boldsymbol{x}_0 + B_{red,N}\boldsymbol{U}_N
\end{bmatrix}
$$

$$
=
\underbrace{\begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{A_{RED,N}} \boldsymbol{x}_0 +
\underbrace{\begin{bmatrix}
B & & & & \\
AB & B & & & \\
\vdots & & \vdots & & \\
A^{N-1}B & A^{N-2}B & \dots & AB & B
\end{bmatrix}}_{B_{RED,N}} \boldsymbol{U}_N.
\tag{10}
$$

One can then formulate the state-constrained optimization problem as;

$$
\begin{aligned}
\min_{\boldsymbol{U}_N} \quad & f(\boldsymbol{X}_N, \boldsymbol{U}_N) \\
\text{s.t.} \quad & \underline{\boldsymbol{X}}_N \le A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N \le \overline{\boldsymbol{X}}_N \\
& \underline{\boldsymbol{U}}_N \le \boldsymbol{U}_N \le \overline{\boldsymbol{U}}_N.
\end{aligned}
\tag{11}
$$

Notice that the state constraints can be implemented as inequalities as

$$
\begin{aligned}
A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\le \overline{\boldsymbol{X}}_N \\
A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\ge \underline{\boldsymbol{X}}_N,
\end{aligned}
\tag{12}
$$

which becomes

$$
\begin{aligned}
A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\le \overline{\boldsymbol{X}}_N \\
-A_{RED,N}\boldsymbol{x}_0 - B_{RED,N}\boldsymbol{U}_N &\le -\underline{\boldsymbol{X}}_N.
\end{aligned}
\tag{13}
$$

This can be written on general form; $A_{in}\boldsymbol{x} \le b_{in}$ (or in our case: $A_{in}\boldsymbol{U}_N \le b_{in}$) , as

$$
\begin{aligned}
B_{RED,N}\boldsymbol{U}_N &\le \overline{\boldsymbol{X}}_N - A_{RED,N}\boldsymbol{x}_0 \\
-B_{RED,N}\boldsymbol{U}_N &\le -\underline{\boldsymbol{X}}_N + A_{RED,N}\boldsymbol{x}_0.
\end{aligned}
\tag{14}
$$

where we see that

$$
\begin{aligned}
A_{in} &= \begin{bmatrix} B_{RED,N} \\ -B_{RED,N} \end{bmatrix} \\
b_{in} &= \begin{bmatrix} \overline{\boldsymbol{X}}_N - A_{RED,N}\boldsymbol{x}_0 \\ -\underline{\boldsymbol{X}}_N + A_{RED,N}\boldsymbol{x}_0 \end{bmatrix}.
\end{aligned}
\tag{15}
$$

Let's say that the state constraints are

$$
\underline{\boldsymbol{X}}_N =
\begin{bmatrix} \underline{\boldsymbol{x}}_1 \\ \underline{\boldsymbol{x}}_2 \\ \vdots \\ \underline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \end{bmatrix},
\quad
\overline{\boldsymbol{X}}_N =
\begin{bmatrix} \overline{\boldsymbol{x}}_1 \\ \overline{\boldsymbol{x}}_2 \\ \vdots \\ \overline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \end{bmatrix}.
\tag{16}
$$

Notice that the end state, $\boldsymbol{x}_N$ is constrained to be equal to the desired final state, $\boldsymbol{x}_f$. This could of course also be included as an equality constraint, instead of implementing it through the inequality constraints. Though, by using inequality constraints, could also implement bounds on the end state rather than a specific value, with $\underline{\boldsymbol{x}}_N$ and $\overline{\boldsymbol{x}}_N$.

To emphasize that one can add cost to the state trajectory as well, we define the cost as

$$
f(\boldsymbol{X}_N, \boldsymbol{U}_N) = \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + \boldsymbol{X}_N^\top \mathcal{Q}_N \boldsymbol{X}_N,
\tag{17}
$$

which using (10) can be written

$$
\begin{aligned}
f(\sim, \boldsymbol{U}_N) = {} & \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N)^\top \mathcal{Q}_N (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\
= {} & \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + \big((A_{RED,N}\boldsymbol{x}_0)^\top + (B_{RED,N}\boldsymbol{U}_N)^\top\big)\mathcal{Q}_N (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\
= {} & \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + (\boldsymbol{x}_0^\top A_{RED,N}^\top + \boldsymbol{U}_N^\top B_{RED,N}^\top)\mathcal{Q}_N (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\
= {} & \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + \boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N A_{RED,N}\boldsymbol{x}_0 \\
& + \boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N + \boldsymbol{U}_N^\top B_{RED,N}^\top \mathcal{Q}_N A_{RED,N}\boldsymbol{x}_0 \\
& + \boldsymbol{U}_N^\top B_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N \\
= {} & \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N + \boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N A_{RED,N}\boldsymbol{x}_0 \\
& + 2\boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N \\
& + \boldsymbol{U}_N^\top B_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N
\end{aligned}
\tag{18}
$$

which can finally be written

$$
f(\sim, \boldsymbol{U}_N) = \boldsymbol{U}_N^\top (\mathcal{R}_N + B_{RED,N}^\top \mathcal{Q}_N B_{RED,N})\boldsymbol{U}_N + 2\boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N. \tag{19}
$$

Notice that the term $\boldsymbol{x}_0^\top A_{RED,N}^\top A_{RED,N}\boldsymbol{x}_0$ is omitted since we assume that the initial state is given, making the term constant. It is also worth mentioning that the terms $\boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N + \boldsymbol{U}_N^\top B_{RED,N}^\top \mathcal{Q}_N A_{RED,N}\boldsymbol{x}_0$ can be combined into $2\boldsymbol{x}_0^\top A_{RED,N}^\top \mathcal{Q}_N B_{RED,N}\boldsymbol{U}_N$, if $\mathcal{Q}_N$ is symmetric, which it always is, since there is never a reason to use a non-symmetric state-cost in a QP. Using (15), (16) and (19) one can implement problem (11) as

Listing 3: Implementation of Reduced Space method with state cost and constraints

```matlab
% Get A_RED_N and B_RED_N:
[A_RED_N, B_RED_N] = ReducedSpace_full(A,B,N);

% Define state limits:
xlb =  -inf(3,1);
xub =   ones(3,1)*14.5;

% Apply limits to trajectory:
Xlb = kron(ones(N,1),xlb);
Xub = kron(ones(N,1),xub);

% Apply desired end state
Xlb(end+1-nx:end) = xf;
Xub(end+1-nx:end) = xf;

% Input limits:
Ulb = kron(ones(N,1),ulb);
Uub = kron(ones(N,1),uub);

% Implement state limits:
Ain = [ B_RED_N;
       -B_RED_N];
bin = [ Xub - A_RED_N*x0;
       -Xlb + A_RED_N*x0];

R_cal = eye(N*nu); % caligrafic R - Input cost
Q_cal = eye(N*nx); % caligrafic Q - state cost

% Solve:
U = quadprog(2*(R_cal + B_RED_N'*Q_cal*B_RED_N),2*x0'*A_RED_N'*Q_cal*B_RED_N,Ain,bin,[],[],
    Ulb,Uub);

% % View resulting state trajectory:
X = A_RED_N*x0 + B_RED_N*U;

function [A_RED_N,B_RED_N] = ReducedSpace_full(A,B,N)
    A_RED_N = [];
    [nx,nu] = size(B);
    B_RED_N = zeros(nx*N,nu*N);
    for i = 1:N
        [A_red_i,B_red_i] = ReducedSpace_k(A,B,i);
        A_RED_N = [A_RED_N; A_red_i];
        B_RED_N((i-1)*nx + (1:nx), 1:(nu*i)) = B_red_i;
    end
end
```
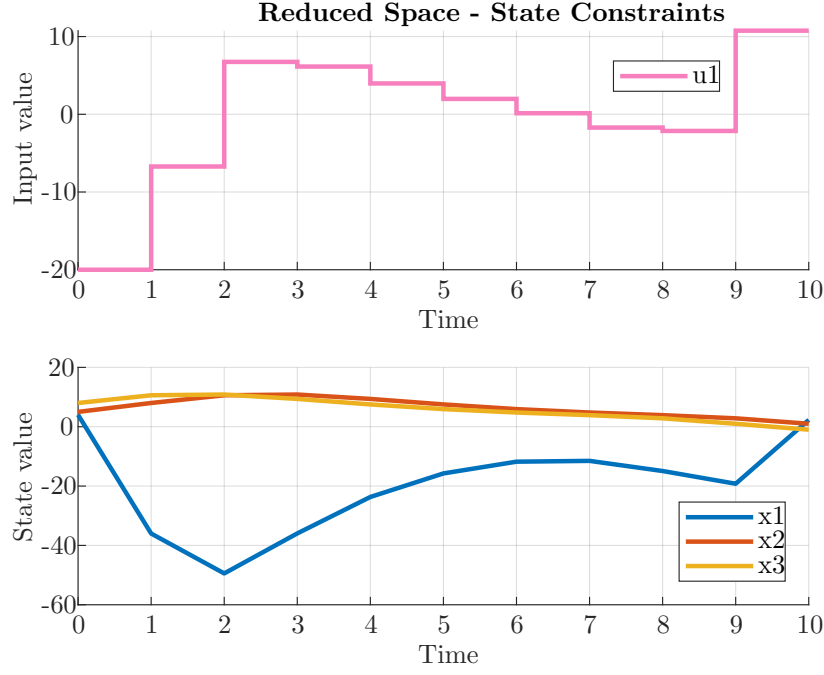
Which gives the following output:

Figure 2: Result of Listing 3, Reduced Space Approach with full state trajectory constraints and cost.

1.2 Full Space

In the full space approach, both inputs and state variables are decision variables. This approach is neatly described by Foss and Heirung (2013), but is summarized here as well. By using (1) we can write the state trajectory as

$$\boldsymbol{X}_N = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} A\boldsymbol{x}_0 + B\boldsymbol{u}_1 \\ A\boldsymbol{x}_1 + B\boldsymbol{u}_2 \\ \vdots \\ A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_N \end{bmatrix}. \tag{20}$$

We then rewrite on matrix form;

$$\underbrace{\begin{bmatrix} I & & & & -B & & & \\ -A & I & & & & -B & & \\ & -A & I & & & & -B & \\ & & \ddots & & & & & \ddots \\ & & -A & I & & & & -B \end{bmatrix}}_{A_{eq}} \underbrace{\begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}}_{\boldsymbol{z}_N} = \underbrace{\begin{bmatrix} A\boldsymbol{x}_0 \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}}_{b_{eq}}. \tag{21}$$

Note that there is a slight error in equation (3.49a) by Foss and Heirung (2013), where the last index of A and B should be "$N-1$". The dimensions are

$$\begin{aligned} A_{eq} &\in \mathbb{R}^{nx \cdot N \ \times \ (nx+nu) \cdot N} \\ b_{eq} &\in \mathbb{R}^{nx \cdot N} \end{aligned}. \tag{22}$$

One can then implement a quadratic objective as

$$\begin{aligned} f(\boldsymbol{X}_N, \boldsymbol{U}_N) = & \ \boldsymbol{X}_N^\top \mathcal{Q}_N \boldsymbol{X}_N + \boldsymbol{U}_N^\top \mathcal{R}_N \boldsymbol{U}_N \\ = & \ \begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}^\top \underbrace{\begin{bmatrix} \mathcal{Q}_N & \\ & \mathcal{R}_N \end{bmatrix}}_{H_N} \begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}. \end{aligned} \tag{23}$$

1.2.1 Adding end constraint

If you want to add an end constraint $\boldsymbol{x}_N = \boldsymbol{x}_f$, you can do so by either adding another equation to the equality constraints as

$$
\left[\begin{array}{cccc|cccc}
I & & & & -B & & & \\
-A & I & & & & -B & & \\
& -A & I & & & & -B & \\
& & \ddots & & & & & \ddots \\
& & -A & I & & & & -B \\
& & & I & & & &
\end{array}\right]
\begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}
=
\begin{bmatrix} A\boldsymbol{x}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{x}_f \end{bmatrix},
\tag{24}
$$

or by adding them via inequalities;

$$
\left[\begin{array}{cccc|c}
0 & \cdots & 0 & -I & \\
0 & \cdots & 0 & I &
\end{array}\ [0]^{N\cdot nx \times N\cdot nu}\ \right]
\begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}
\leq
\begin{bmatrix} -\boldsymbol{x}_f \\ \boldsymbol{x}_f \end{bmatrix},
\tag{25}
$$

or by simply adding limits of the state directly;

$$
\underline{\boldsymbol{Z}}_N = \begin{bmatrix} \underline{\boldsymbol{X}}_N \\ \underline{\boldsymbol{U}}_N \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{x}}_1 \\ \underline{\boldsymbol{x}}_2 \\ \vdots \\ \underline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \\ \underline{\boldsymbol{u}}_0 \\ \underline{\boldsymbol{u}}_1 \\ \vdots \\ \underline{\boldsymbol{u}}_{N-1} \end{bmatrix}, \quad
\overline{\boldsymbol{Z}}_N = \begin{bmatrix} \overline{\boldsymbol{X}}_N \\ \overline{\boldsymbol{U}}_N \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{x}}_1 \\ \overline{\boldsymbol{x}}_2 \\ \vdots \\ \overline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \\ \overline{\boldsymbol{u}}_0 \\ \overline{\boldsymbol{u}}_1 \\ \vdots \\ \overline{\boldsymbol{u}}_{N-1} \end{bmatrix}.
\tag{26}
$$

The full space approach can be implemented in MATLAB as:

Listing 4: Implementation of the Full Space method in MATLAB

```matlab
% Obtain full space equality constaints:
[Aeq,beq] = FullSpace(A,B,N);

% Add initial value separately:
beq(1:nx) = A*x0;

% Define state limits:
xlb = -inf(3,1);
xub =  ones(3,1)*14.5;

% Apply limits to trajectory:
Xlb = kron(ones(N,1),xlb);
Xub = kron(ones(N,1),xub);

% Apply desired end state
Xlb(end+1-nx:end) = xf;
Xub(end+1-nx:end) = xf;

% Input limits:
Ulb = kron(ones(N,1),ulb);
Uub = kron(ones(N,1),uub);

% Limits:
lb = [Xlb;Ulb];   ub = [Xub;Uub];

% state costs:
q(1) = 0.87;
q(2) = 1.21;
q(3) = 1.15;
Q = diag(q);

% input cost:
r(1) = 6.45;
R = diag(r);

% Build objective hessian:
H = FullSpaceObjective(Q,R,N);
```

```
% Solve:
Z = quadprog(2*H,[],[],[],Aeq,beq,lb,ub);

% Extract solution:
U = Z((N*nx) + (1:N*nu));
X = Z(1:(N*nx));

function[A_eq,b_eq] = FullSpace(A,B,N)
    nx = size(A,1);
    A_diag = kron( -diag(ones(N-1,1),-1), A);
    A_diag = eye(N*nx) + A_diag;
    B_diag = kron( -diag(ones(N,1)), B);
    A_eq = [A_diag B_diag];
    b_eq = zeros(N*nx,1);
end

function[H] = FullSpaceObjective(Q,R,N)
    H_Q = kron(diag(ones(N,1)), Q);
    H_R = kron(diag(ones(N,1)), R);
    H = blkdiag(H_Q,H_R);
end
```
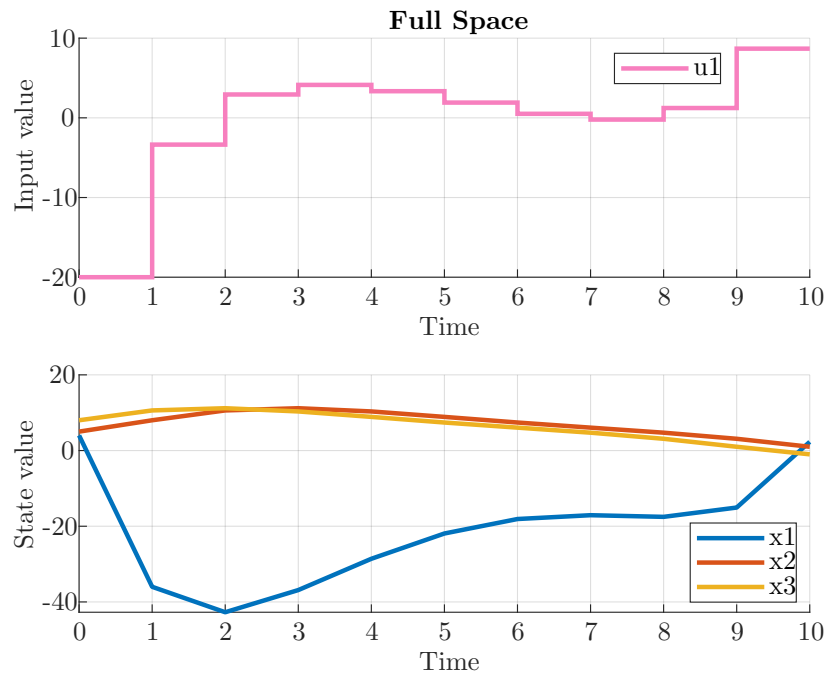
which gives:

Figure 3: Result of Listing 4, Full Space Approach.

References

Foss, Bjarne and Tor Aksel N Heirung (2013). "Merging optimization and control". In: *Lecture Notes*.