Open Loop Optimization of LTI systems

Trym Arve L. Gabrielsen

February 28, 2023

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 som desired end state $\boldsymbol{x}_f$. 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 = [ -2.1   0.7   0.3 ;
        1     0     0  ;
        0     1     0  ];
B = [ 0.8 ;
        0 ;
      3.2 ];

[nx,nu] = size(B);

% Initial state
x0 = [ 1.1 ;
        0  ;
        0  ];

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

% Bounds on input:
ulb = -7*ones(nu,1);
uub = 8*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. The next states can then be found as

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

Let's write (2) on matrix form;

$$\boldsymbol{x}_k = \underbrace{A^k}_{A_{red,k}}\boldsymbol{x}_0 + \underbrace{\begin{bmatrix} A^{k-1}B & A^{k-2}B & \dots & AB & B \end{bmatrix}}_{B_{red,k}}\boldsymbol{U}_k, \tag{3}$$

where

$$\boldsymbol{U}_k = \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{k-1} \end{bmatrix}. \tag{4}$$

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 (3) and (4):

$$\begin{aligned}
\min_{\boldsymbol{U}_N} \quad & f(\boldsymbol{U}_N) \\
\text{s.t.} \quad & \boldsymbol{x}_f = A_{red,N}\boldsymbol{x}_0 + B_{red,N}\boldsymbol{U}_N \\
& \underline{\boldsymbol{U}}_N \leq \boldsymbol{U}_N \leq \overline{\boldsymbol{U}}_N.
\end{aligned} \tag{5}$$

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;

$$\begin{aligned}
A_{eq} &= B_{red,N} \\
\boldsymbol{b}_{eq} &= \boldsymbol{x}_f - A_{red,N}\boldsymbol{x}_0.
\end{aligned} \tag{6}$$

If we say assume that the input-cost is

$$f(\boldsymbol{U}_N) = \boldsymbol{U}_N^\top \boldsymbol{U}_N, \tag{7}$$

making problem (5) 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:

Listing 3: Output of Listing 2

```
U =

    5.4006
   -2.2798
    1.0215
   -0.3403
    0.3449
    0.2181
    0.7711
    0.8774
    4.0778
   -0.8029
```

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;

$$\begin{aligned}\boldsymbol{X}_N = \begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \vdots\\ \boldsymbol{x}_N\end{bmatrix} &= \begin{bmatrix} A_{red,1}\boldsymbol{x}_0 + B_{red,1}\boldsymbol{U}_1\\ A_{red,2}\boldsymbol{x}_0 + B_{red,2}\boldsymbol{U}_2\\ \vdots\\ A_{red,N}\boldsymbol{x}_0 + B_{red,N}\boldsymbol{U}_N\end{bmatrix}\\ &= \underbrace{\begin{bmatrix}A^1\\ A^2\\ \vdots\\ A^N\end{bmatrix}}_{A_{RED,N}}\boldsymbol{x}_0 + \underbrace{\begin{bmatrix} B & & & & \\ AB & B & & & \\ \vdots & \vdots & & & \\ A^{N-1}B & A^{N-2}B & \dots & AB & B\end{bmatrix}}_{B_{RED,N}}\boldsymbol{U}_N.\end{aligned} \tag{8}$$

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

$$\begin{aligned}\min_{\boldsymbol{U}_N}\quad & f(\boldsymbol{X}_N, \boldsymbol{U}_N)\\ \text{s.t.}\quad & \underline{\boldsymbol{X}}_N \le A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N \le \overline{\boldsymbol{X}}_N\\ & \underline{\boldsymbol{U}}_N \le \boldsymbol{U}_N \le \overline{\boldsymbol{U}}_N.\end{aligned} \tag{9}$$

Notice that the state constraints can be implemented as inequalities as

$$\begin{aligned}A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\le \overline{\boldsymbol{X}}_N\\ A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\ge \underline{\boldsymbol{X}}_N,\end{aligned} \tag{10}$$

which becomes

$$\begin{aligned}A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N &\le \overline{\boldsymbol{X}}_N\\ -A_{RED,N}\boldsymbol{x}_0 - B_{RED,N}\boldsymbol{U}_N &\le -\underline{\boldsymbol{X}}_N.\end{aligned} \tag{11}$$

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

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

where we see that

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

Let's say that the state constraints are

$$\underline{\boldsymbol{X}}_N = \begin{bmatrix} \underline{\boldsymbol{x}}_1 \\ \underline{\boldsymbol{x}}_2 \\ \vdots \\ \underline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \end{bmatrix}, \quad \overline{\boldsymbol{X}}_N = \begin{bmatrix} \overline{\boldsymbol{x}}_1 \\ \overline{\boldsymbol{x}}_2 \\ \vdots \\ \overline{\boldsymbol{x}}_{N-1} \\ \boldsymbol{x}_f \end{bmatrix}. \tag{14}$$

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. One 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

$$f(\boldsymbol{X}_N, \boldsymbol{U}_N) = \boldsymbol{U}_N^\top \boldsymbol{U}_N + \boldsymbol{X}_N^\top \boldsymbol{X}_N, \tag{15}$$

which using (8) can be written

$$\begin{aligned} f(\sim, \boldsymbol{U}_N) = \; & \boldsymbol{U}_N^\top \boldsymbol{U}_N + (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N)^\top (A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\ = \; & \boldsymbol{U}_N^\top \boldsymbol{U}_N + ((A_{RED,N}\boldsymbol{x}_0)^\top + (B_{RED,N}\boldsymbol{U}_N)^\top)(A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\ = \; & \boldsymbol{U}_N^\top \boldsymbol{U}_N + (\boldsymbol{x}_0^\top A_{RED,N}^\top + \boldsymbol{U}_N^\top B_{RED,N}^\top)(A_{RED,N}\boldsymbol{x}_0 + B_{RED,N}\boldsymbol{U}_N) \\ = \; & \boldsymbol{U}_N^\top \boldsymbol{U}_N + \boldsymbol{x}_0^\top A_{RED,N}^\top A_{RED,N}\boldsymbol{x}_0 \\ & + \boldsymbol{x}_0^\top A_{RED,N}^\top B_{RED,N}\boldsymbol{U}_N + \boldsymbol{U}_N^\top B_{RED,N}^\top A_{RED,N}\boldsymbol{x}_0 \\ & + \boldsymbol{U}_N^\top B_{RED,N}^\top B_{RED,N}\boldsymbol{U}_N \\ = \; & \boldsymbol{U}_N^\top \boldsymbol{U}_N + \boldsymbol{x}_0^\top A_{RED,N}^\top A_{RED,N}\boldsymbol{x}_0 \\ & + 2\boldsymbol{x}_0^\top A_{RED,N}^\top B_{RED,N}\boldsymbol{U}_N \\ & + \boldsymbol{U}_N^\top B_{RED,N}^\top B_{RED,N}\boldsymbol{U}_N \end{aligned} \tag{16}$$

which can finally be written

$$f(\sim, \boldsymbol{U}_N) = \boldsymbol{U}_N^\top (I + B_{RED,N}^\top B_{RED,N})\boldsymbol{U}_N + 2\boldsymbol{x}_0^\top A_{RED,N}^\top B_{RED,N}\boldsymbol{U}_N. \tag{17}$$

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. Using (13), (14) and (17) one can implement problem (9) as

Listing 4: Implementation of Reduced Space method with state cost and constraints

```
% 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];

% Solve:
U = quadprog(2*(eye(N*nu) + B_RED_N'*B_RED_N),2*x0'*A_RED_N'*B_RED_N,Ain,bin,[],[],Ulb,Uub)

%% View resulting state trajectory:
X = A_RED_N*x0 + B_RED_N*U;
X = reshape(X,nx,[])

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:

Listing 5: Output of Listing 4

```
U =

    4.5312
   -4.1643
    1.3333
   -0.4731
    0.2424
   -0.0673
    0.2698
    0.6603
    4.2625
   -0.7486


X =

  1.3150   -0.9730    0.3626   -0.1465    0.0095    0.1652   -0.2332    1.3954    1.0000    2.3000
  1.1000    1.3150   -0.9730    0.3626   -0.1465    0.0095    0.1652   -0.2332    1.3954    1.0000
 14.5000  -12.2258    5.5816   -2.4867    1.1384   -0.3620    0.8727    2.2783   13.4069   -1.0000
```

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. We continue using

$$\boldsymbol{X}_N = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}, \quad \boldsymbol{U}_N = \begin{bmatrix} \boldsymbol{u}_0 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} \tag{18}$$

By using (1) we can write the state trajectory as

$$\boldsymbol{X}_N = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} A\boldsymbol{x}_0 + B\boldsymbol{u}_1 \\ A\boldsymbol{x}_1 + B\boldsymbol{u}_2 \\ \vdots \\ A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_N \end{bmatrix}. \tag{19}$$

We then rewrite on matrix form;

$$\underbrace{\left[\begin{array}{cccc|cccc} I & & & & -B & & & \\ -A & I & & & & -B & & \\ & -A & I & & & & -B & \\ & & \ddots & & & & \ddots & \\ & & -A & I & & & & -B \end{array}\right]}_{A_{eq}} \begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix} = \underbrace{\begin{bmatrix} A\boldsymbol{x}_0 \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}}_{b_{eq}}. \tag{20}$$

One can then implement the objective as

$$\begin{aligned} f(\boldsymbol{X}_N, \boldsymbol{U}_N) = &\quad \boldsymbol{X}_N^\top Q \boldsymbol{X}_N + \boldsymbol{U}_N^\top R \boldsymbol{U}_N \\ = &\quad \begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}^\top \begin{bmatrix} Q & \\ & R \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_N \\ \boldsymbol{U}_N \end{bmatrix}, \end{aligned} \tag{21}$$

where

$$Q = \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_N \end{bmatrix} \tag{22}$$

are the state penalties, and

$$R = \begin{bmatrix} R_0 & & \\ & \ddots & \\ & & R_{N-1} \end{bmatrix} \tag{23}$$

are the input penalties on the form

$$f_k^x = \boldsymbol{x}_k^\top Q_k \boldsymbol{x}_k, \quad f_k^u = \boldsymbol{u}_k^\top R_k \boldsymbol{u}_k, \tag{24}$$

where f_k^x and f_k^u are the costs of state $\boldsymbol{x}_k$ and input $\boldsymbol{u}_k$ respectively. The dimensions are

$$\begin{aligned} A_{eq} &\in R^{nx \cdot N \times (nx+nu) \cdot N} \\ b_{eq} &\in R^{nx \cdot N} \end{aligned}. \tag{25}$$

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 full space approach can be implemented in MATLAB as:

Listing 6: Implementation of the Full Space method in 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 = reshape(z(1:(N*nx)),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:

Listing 7: Output of Listing (6)

```
U =

    4.5312
   -4.1042
    1.4372
   -0.5439
    0.2837
   -0.0100
    0.4188
    0.7215
    4.2135
   -0.7628


X =

   1.3150   -0.9248    0.4024   -0.1534    0.0311    0.2123   -0.1447    1.4409    1.0000    2.3000
   1.1000    1.3150   -0.9248    0.4024   -0.1534    0.0311    0.2123   -0.1447    1.4409    1.0000
  14.5000  -12.0334    5.9140   -2.6654    1.3102   -0.1855    1.3711    2.5209   13.3387   -1.0000
```

References

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