Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)

Constrained Numerical Optimization for Estimation and Control

Laboratory session F - Quadratic Programming and linear-quadratic Model Predictive Control

Course 052358 - M.Sc. Programme in Automation and Control Engineering

Date:	September 12, 2021
Version:	1.0
Authors:	Lorenzo Fagiano Marco Lauricella
Contacts:	Lorenzo Fagiano +39.02.2399.9609 lorenzo.fagiano@polimi.it

Contents

1 Introduction and learning goals

Quadratic programs (QPs) are an important class of constrained optimization problems, arising in many fields of science and engineering. In estimation and control, one of the most important applications of QPs is the formulation and solution to Finite Horizon Optimal Control Problems (FHOCPs) with quadratic cost function, linear and time-invariant system dynamics, and linear equality and inequality constraints. In turn, such FHOCPs are at the base of Model Predictive Control (MPC), the most common advanced, multi-variable control technique in industry.

The term linear-quadratic MPC refers to MPC for LTI systems with quadratic cost function. Linear-quadratic MPC entails a QP to be solved at each time step: such a computation can be carried out in milliseconds with today's software and hardware. Even in such seemingly restricted settings (linear dynamics and constraints, quadratic cost), there are many different variants of linear quadratic MPC, depending for example on the employed terminal cost and constraints, on the inclusion of integral action, on the use of soft constraints.

In this laboratory session we will experience the use of QPs, in particular resulting from FHOCPs, and the design of a MPC strategy for an example concerned with a simple electromechanical positioning system. We will consider first a FHOCP with terminal equality constraint only, and derive in this case the analytic solution formula resorting to the KKT conditions for equality-constrained QPs. We will then compare the results with a Linear Quadratic Regulator (LQR), which represents the analytic solution to an infinite-horizon optimal control problem for LTI systems. Finally, we will include inequality constraints and embed the resulting QP in a MPC strategy.

The learning goals are:

- To learn how to formulate and solve QPs;
- To learn how to write finite horizon problems (control, estimation) for LTI systems with linear constraints and quadratic cost as QPs;
- To learn how to implement a linear-quadratic MPC strategy.

As computational tool, we will resort to Matlab's `quadprog`. Any other QP solver can be employed.

2 FHOCP and MPC for LTI systems with quadratic cost, linear inequality constraints, and terminal equality constraint

Consider the LTI, discrete-time system:

$$\begin{aligned} \boldsymbol{z}(t+1) &= A\boldsymbol{z}(t)+B\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= C\boldsymbol{z}(t)+D\boldsymbol{u}(t) \end{aligned} \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^{n_{\boldsymbol{z}}}$, $\boldsymbol{u} \in \mathbb{R}^{n_{\boldsymbol{u}}}$, $\boldsymbol{y} \in \mathbb{R}^{n_{\boldsymbol{y}}}$ and A, B, C, D are the system matrices. We assume that the system is fully reachable and observable. The FHOCP of interest takes the following form:

$$\min_U \sum_{i=0}^{N}(\boldsymbol{y}_{\text{ref}}-\boldsymbol{y}(i|t))^T Q(\boldsymbol{y}_{\text{ref}}-\boldsymbol{y}(i|t)) + \sum_{i=0}^{N-1}\boldsymbol{u}^T R\boldsymbol{u} \tag{2a}$$

s.t.

$$\boldsymbol{z}(i+1|t) = A\boldsymbol{z}(i|t)+B\boldsymbol{u}(i|t) \tag{2b}$$

$$\boldsymbol{y}(i|t) = C\boldsymbol{z}(i|t)+D\boldsymbol{u}(i|t) \tag{2c}$$

$$\boldsymbol{z}(0|t) = \boldsymbol{z}(t) \tag{2d}$$

$$C_{\boldsymbol{z}}\boldsymbol{z}(i|t)+\boldsymbol{d}_{\boldsymbol{z}} \geq 0,\ i=0,\dots,N \tag{2e}$$

$$C_{\boldsymbol{u}}\boldsymbol{u}(i|t)+\boldsymbol{d}_{\boldsymbol{u}} \geq 0,\ i=0,\dots,N-1 \tag{2f}$$

$$\boldsymbol{z}(N-1|t)-\boldsymbol{z}(N|t) = 0 \tag{2g}$$

Notation $\boldsymbol{y}(i|t)$ indicates the value of $\boldsymbol{y}$ at time $t+i$, predicted at time t. N is the prediction horizon, $U = [\boldsymbol{u}(0|t)^T, \dots, \boldsymbol{u}(N-1|t)^T]^T$, $Q > 0$, $R > 0$ are symmetric weighting matrices, and $C_{\boldsymbol{z}} \in \mathbb{R}^{q_{\boldsymbol{z}} \times n_{\boldsymbol{z}}}$, $C_{\boldsymbol{u}} \in \mathbb{R}^{q_{\boldsymbol{u}} \times n_{\boldsymbol{u}}}$, $\boldsymbol{d}_{\boldsymbol{z}} \in \mathbb{R}^{q_{\boldsymbol{z}}}$, $\boldsymbol{d}_{\boldsymbol{u}} \in \mathbb{R}^{q_{\boldsymbol{u}}}$ are known matrices and vectors defining the state and input inequality constraints. The initial state $\boldsymbol{z}(t)$ is assumed to be known. The equality constraint (2g) requires that the terminal state is a steady state. This is a possible way to guarantee recursive feasibility and closed-loop stability when the FHOCP (2) is embedded in a MPC strategy.
Other formulations, involving mixed state-input constraints and additional equality constraints, can be considered with minor modifications.
We shall now re-write (2) as a QP. First, note that (2b)-(2d) can be eliminated by substitution, defining vector $Z = [\boldsymbol{z}(0|t)^T, \dots, \boldsymbol{z}(N|t)^T]^T \in \mathbb{R}^{(N+1)n_{\boldsymbol{z}}}$ and $Y = [\boldsymbol{y}(0|t)^T, \dots, \boldsymbol{y}(N|t)^T]^T \in \mathbb{R}^{(N+1)n_{\boldsymbol{y}}}$:

$$Z = \underbrace{\begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}}_{\Lambda_{\boldsymbol{z}}} \boldsymbol{z}(t) + \underbrace{\begin{bmatrix} \mathbf{0} & \cdots & \cdots & \cdots & \mathbf{0} \\ B & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ AB & B & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ A^{N-1}B & \cdots & \cdots & AB & B \end{bmatrix}}_{\Gamma_{\boldsymbol{z}}} U$$

$$Y = \underbrace{\begin{bmatrix} C & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & C & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & C \end{bmatrix}}_{\bar{C}} Z + \underbrace{\begin{bmatrix} D & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & D \end{bmatrix}}_{\bar{D}} U = \underbrace{\bar{C}\Lambda_{\boldsymbol{z}}}_{\Lambda_{\boldsymbol{y}}} \boldsymbol{z}(t) + \underbrace{(\bar{C}\Gamma_{\boldsymbol{z}} + \bar{D})}_{\Gamma_{\boldsymbol{y}}} U$$

Moreover, we define vector $\bar{Y}_{\text{ref}} = [\boldsymbol{y}_{\text{ref}}^T, \dots, \boldsymbol{y}_{\text{ref}}^T]^T \in \mathbb{R}^{(N+1)n_{\boldsymbol{y}}}$, $\bar{D}_{\boldsymbol{z}} = [\boldsymbol{d}_{\boldsymbol{z}}^T, \dots, \boldsymbol{d}_{\boldsymbol{z}}^T]^T \in \mathbb{R}^{(N+1)q_{\boldsymbol{z}}}$, $\bar{D}_{\boldsymbol{u}} = [\boldsymbol{d}_{\boldsymbol{u}}^T, \dots, \boldsymbol{d}_{\boldsymbol{u}}^T]^T \in \mathbb{R}^{(N)q_{\boldsymbol{u}}}$, and matrices

$$\bar{C}_{\boldsymbol{z}} = \begin{bmatrix} C_{\boldsymbol{z}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & C_{\boldsymbol{z}} & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & C_{\boldsymbol{z}} \end{bmatrix} \qquad \bar{C}_{\boldsymbol{u}} = \begin{bmatrix} C_{\boldsymbol{u}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & C_{\boldsymbol{u}} & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & C_{\boldsymbol{u}} \end{bmatrix}$$

$$\bar{Q} = \begin{bmatrix} Q & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & Q & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & Q \end{bmatrix} \qquad \bar{R} = \begin{bmatrix} R & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & R & \ddots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & R \end{bmatrix}$$

We can now re-write the cost function in (2) as:

$$\bar{Y}_{\text{ref}}^T\bar{Q}\bar{Y}_{\text{ref}} + (\Lambda_{\boldsymbol{y}}\boldsymbol{z}(t))^T\bar{Q}(\Lambda_{\boldsymbol{y}}\boldsymbol{z}(t)) + (\Gamma_{\boldsymbol{y}}U)^T\bar{Q}(\Gamma_{\boldsymbol{y}}U) - 2\bar{Y}_{\text{ref}}^T\bar{Q}\Lambda_{\boldsymbol{y}}\boldsymbol{z}(t) - 2\bar{Y}_{\text{ref}}^T\bar{Q}\Gamma_{\boldsymbol{y}}U + 2(\Lambda_{\boldsymbol{y}}\boldsymbol{z}(t))^T\bar{Q}\Gamma_{\boldsymbol{y}}U + U^T\bar{R}U$$

By eliminating terms in the cost function that do not depend on the decision variables U and dividing the cost by two, we obtain the following QP:

$$\min_U \underbrace{((\Lambda_{\boldsymbol{y}}\boldsymbol{z}(t))^T - \bar{Y}_{\text{ref}}^T)\bar{Q}\Gamma_{\boldsymbol{y}}}_{\boldsymbol{c}^T} U + \frac{1}{2}U^T \underbrace{(\Gamma_{\boldsymbol{y}}^T\bar{Q}\Gamma_{\boldsymbol{y}} + \bar{R})}_{H} U \tag{3a}$$

s.t.

$$\begin{bmatrix} \bar{C}_{\boldsymbol{z}}\Gamma_{\boldsymbol{z}} \\ \bar{C}_{\boldsymbol{u}} \end{bmatrix} U + \begin{bmatrix} \bar{C}_{\boldsymbol{z}}\Lambda_{\boldsymbol{z}}\boldsymbol{z}(t) + \bar{D}_{\boldsymbol{z}} \\ \bar{D}_{\boldsymbol{u}} \end{bmatrix} \geq 0 \tag{3b}$$

$$PU + M\boldsymbol{z}(t) = 0 \tag{3c}$$

where

$$\begin{aligned} P &= \left(\Gamma_{\boldsymbol{z}((N-2)n_{\boldsymbol{z}}+1:(N-1)n_{\boldsymbol{z}},:)} - \Gamma_{\boldsymbol{z}((N-1)n_{\boldsymbol{z}}+1:Nn_{\boldsymbol{z}},:)}\right) \\ M &= \left(\Lambda_{\boldsymbol{z}((N-2)n_{\boldsymbol{z}}+1:(N-1)n_{\boldsymbol{z}},:)} - \Lambda_{\boldsymbol{z}((N-1)n_{\boldsymbol{z}}+1:Nn_{\boldsymbol{z}},:)}\right) \end{aligned}$$

Note that if no inequality constraints (3b) are present, the solution to (3) can be computed in closed form resorting to the KKT conditions (see Chapter 6 of the lecture notes):

$$\begin{bmatrix} H & -P^T \\ P & \mathbf{0} \end{bmatrix} \begin{bmatrix} U \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\Gamma_{\boldsymbol{y}}^T\bar{Q}\Lambda_{\boldsymbol{y}} & \Gamma_{\boldsymbol{y}}^T\bar{Q} \\ -M & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}(t) \\ \bar{Y}_{\text{ref}} \end{bmatrix}$$

thus:

$$\begin{bmatrix} U^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \underbrace{\begin{bmatrix} H & -P^T \\ P & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} -\Gamma_{\boldsymbol{y}}^T\bar{Q}\Lambda_{\boldsymbol{y}} & \Gamma_{\boldsymbol{y}}^T\bar{Q} \\ -M & \mathbf{0} \end{bmatrix}}_{G} \begin{bmatrix} \boldsymbol{z}(t) \\ \bar{Y}_{\text{ref}} \end{bmatrix} \tag{4}$$

Note that the optimal control sequence is linear in the initial state and desired reference (extension to non-constant reference values along the prediction horizon is straightforward). The first $n_{\boldsymbol{u}}$ rows of matrix G form a matrix $K \in \mathbb{R}^{n_{\boldsymbol{u}} \times n_{\boldsymbol{z}}}$ which defines a static, linear, stabilizing state-feedback control law $\boldsymbol{u}(t) = K\boldsymbol{z}(t)$. This can be seen easily by considering the receding horizon implementation of the FHOCP, which leads to the following MPC strategy.

Linear-Quadratic Model Predictive Control

1. Acquire the current state $\boldsymbol{z}(t)$;
2. Solve the QP (3);
3. Apply the first input in the optimal sequence to the plant, i.e. $\boldsymbol{u}(t) = \boldsymbol{u}^*(0|t)$;
4. Set $t = t + 1$, go to 1.

When inequality constraints are present, then the analytical solution still exists and consists of a continuous, piecewise affine map from the state and reference values to the input [1]. Such a map becomes rather complex already with relatively small number of states, inputs, constraints, and prediction horizon. Thus, a sensible way to implement MPC with inequality constraints is to directly solve the QP (3) in the receding horizon strategy.

3 Electromechanical positioning system

Consider the positioning system depicted in Figure 1. It consists of an electric gearmotor where we indicate with V the input voltage, θ_i and θ_o the angular positions of the input and output shafts respectively, T the torsional momentum applied to the output shaft, whose torsional stiffness is indicated with K_θ. R_{mot} is the motor electrical resistance, K_T is the motor constant, J_i and J_o the moments of inertia of the input and output shafts, β_i and β_o the friction coefficients of the shafts, finally τ_g is the transmission ratio of the gearbox. The numerical values of the system parameters are reported in Table 1.

The input variable is the voltage V. The control objective is to track a desired reference $\theta_{o,\text{ref}}$ of the output angular position θ_o, while keeping T below its yield point of $\pm\bar{T}$ Nm. The input voltage limits are equal to $\pm\bar{V}$.

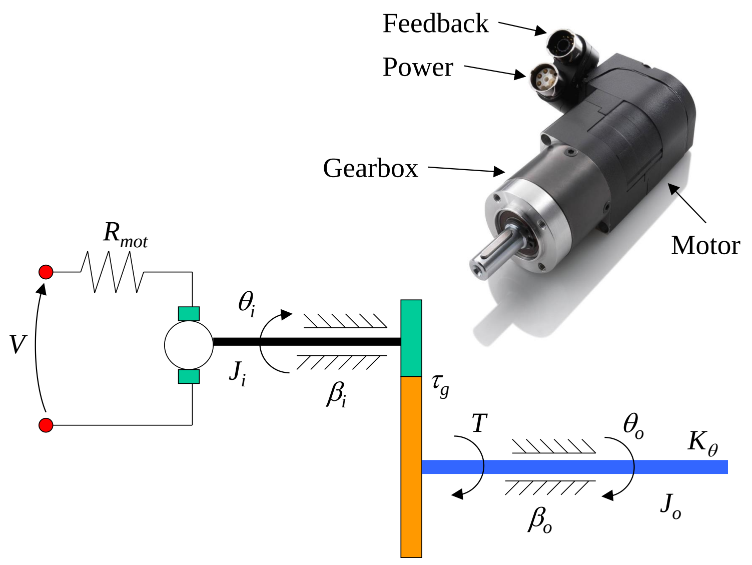

Figure 1: Example of servomotor with gearbox used in positioning systems, and related model.

Symbol	Description	Value
R_{mot}	Motor electrical resistance (Ω)	20
K_T	Motor constant (Nm/A)	10
K_θ	Output shaft torsional stiffness (Nm/rad)	1280
J_i	Input shaft moment of inertia (kg m^2)	0.5
J_o	Output shaft moment of inertia (kg m^2)	25
β_i	Input shaft friction coefficient (Nm s/rad)	0.1
β_o	Output shaft friction coefficient(Nm s/rad)	25
τ_g	Gear ratio (input/output)	20
$\bar{T}$	Maximal output shaft torsional moment (Nm)	78.5
$\bar{V}$	Maximal input voltage (V)	220

Table 1: Electromechanical positioning system - model parameters

The model equations in continuous time read ($\tau \in \mathbb{R}$ is the continuous time variable):

$$\begin{aligned} J_i\ddot{\theta}_i(\tau) + \beta_i\dot{\theta}_i(\tau) &= \frac{K_T V(\tau) - K_T^2\dot{\theta}_i(\tau)}{R_{mot}} - \frac{T(\tau)}{\tau_g} \\ J_o\ddot{\theta}_o(\tau) + \beta_o\dot{\theta}_o(\tau) &= T(\tau) \\ T(\tau) &= K_\theta\left(\frac{\theta_i(\tau)}{\tau_g} - \theta_o(\tau)\right) \end{aligned} \tag{5}$$

We can arrange the equations in the standard form for LTI systems:

$$\begin{aligned} \dot{\boldsymbol{z}}(\tau) &= \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_\theta}{J_i\tau_g^2} & \frac{K_\theta}{J_i\tau_g} & -\frac{\beta_i + \frac{K_T^2}{R_{mot}}}{J_i} & 0 \\ \frac{K_\theta}{J_o\tau_g} & -\frac{K_\theta}{J_o} & 0 & -\frac{\beta_o}{J_o} \end{bmatrix}}_{A^{\text{CT}}} \boldsymbol{z}(\tau) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \frac{K_T}{J_i R_{mot}} \\ 0 \end{bmatrix}}_{B^{\text{CT}}} u(\tau) \\ y(\tau) &= \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}}_{C^{\text{CT}}} \boldsymbol{z}(\tau) \end{aligned} \tag{6}$$

where $\boldsymbol{z} = [\theta_i, \theta_o, \dot{\theta}_i, \dot{\theta}_o]^T$, $u = V$, $y = \theta_o$. After choosing a suitable sampling time T_s, we obtain a discrete time LTI model of the form (1).

4 Laboratory session's assignments

This laboratory session includes three parts and one extra assignment.

I) Derive a discrete-time model for the electromechanical positioning system.
Write in Matlab the model equations, and derive the discrete-time system matrices A, B, C, D using a method of choice and a suitable sampling time T_s, e.g. 0.1s as detailed in the attached, commented Matlab code.

II) Formulate and solve a FHOCP for this problem.
The control objective is to track a desired reference $\theta_{o,\text{ref}}$ of the output angular position θ_o, while keeping T below its yield point of $\pm\bar{T}$ Nm. The input voltage limits are equal to $\pm\bar{V}$. This can be translated into a FHOCP of the form (3) with both input and state constraints (the torsional momentum is in fact linear in the state).
Formulate first a problem without inequality constraints and possibly with terminal equality constraints. You can use the provided function `Traj_matrices.m` to compute matrices $\Lambda_{\boldsymbol{z}}$, $\Gamma_{\boldsymbol{z}}$, $\Lambda_{\boldsymbol{y}}$, $\Gamma_{\boldsymbol{y}}$ and you have to build the other matrices as shown in Section2. When no inequalities are present, you can compute the explicit solution (4). Compare the results with LQR. To do so, for simplicity set the reference to zero and use consistent initial conditions (remember that the output shaft position is equal to the input one divided by τ_g). Compare also with the FHOCP without any constraints.
Finally, compare the results with a problem with inequality constraints as well. See what happens when changing the weighting matrices. For example, see the attached, commented Matlab code.

III) Formulate a MPC strategy, and simulate the positioning system with MPC.
Set up a strategy as described in Section 2, using the constrained problem. You can then simulate the feedback system. Try different prediction horizon values, and compare the results with the FHOCP (i.e. no receding horizon implementation). For an example, see the attached, commented Matlab code.

IV) (Extra) Try different MPC formulations.
Many variants of MPC exist, for example with soft constraints (by adding suitable slack variables); with a velocity form to include integral action; with disturbance estimation. Search in the literature or in reference textbooks such formulations, and try to implement them.

References

[1] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38:3–20, 2002.

```
function [Lambda_y,Gamma_y,Lambda_z,Gamma_z]=Traj_matrices(M,A,B,C,D)
% TRAJ_MATRICES Computes the matrices that relate the state and output
% trajectories of a LTI system to the initial condition and input sequence
% from the current time t up to t+N:
%                   XI   =   Lambda_z*z0+Gammma_z*U
%                   Y    =   Lambda_y*z0+Gammma_y*U
% where XI = [z(t)^T,...,z(t+M)^T]^T, Y = [y(t)^T,...,y(t+M)^T]^T and
% U = [u(t)^T,...,u(t+M)^T]^T
%
% The system is assumed to be modeled in discrete time:
%              z(t+1)   =   A*z(t)+B*u(t)
%                 y(t)   =   C*z(t)+D*u(t)
%
%   INPUTS:
%           M           =   prediction horizon
%           A,B,C,D     =   discrete-time system matrices
%
%   OUTPUTS:
%           Lambda_y    =   Matrix relating Y to z(t)
%           Gamma_y     =   Matrix relating Y to U
%           Lambda_z   =   Matrix relating Z to z(t)
%           Gamma_z    =   Matrix relating Z to U

nz      =   size(A,1);
nu       =   size(B,2);
ny       =   size(C,1);

Lambda_y  =   zeros(ny*(M+1),nz);
Gamma_y   =   zeros(ny*(M+1),nu*M);
Lambda_z =   zeros(nz*(M+1),nz);
Gamma_z  =   zeros(nz*(M+1),nu*M);

for ind = 1:M+1
   Lambda_y((ind-1)*ny+1:ind*ny,:)          =   C*A^(ind-1);
   Lambda_z((ind-1)*nz+1:ind*nz,:)       =   A^(ind-1);
   for ind2 =   1:ind-1
       Gamma_z((ind-1)*nz+1:ind*nz,(ind2-1)*nu+1:ind2*nu)=A^(ind-ind2-1)*B;
       if ind2==ind-1
           Gamma_y((ind-1)*ny+1:ind*ny,(ind2-1)*nu+1:ind2*nu)=C*B+D;
       else
           Gamma_y((ind-1)*ny+1:ind*ny,(ind2-1)*nu+1:ind2*nu)=C*A^(ind-ind2-1)*B;
       end
   end
end
```

```matlab
% Constrained Numerical Optimization for Estimation and Control
% Laboratory session F
% Script to formulate and solve a Finite Horizon Optimal Control Problem
% for the linear servo-mechanism example, with terminal equality constraint
% only

clear all
close all
clc

%% System model
% Model parameters
R       = 20;           % Electric resistance
Ktheta  = 1280;         % Torsional stiffness output shaft
Kt      = 10;           % Motor constant
tau_g   = 20;           % Gear ratio
J_i     = .5;           % Input shaft moment of inertia
J_o     = 25;           % Output shaft moment of inertia
beta_i  = 0.1;          % Input shaft friction coefficient
beta_o  = 25;           % Output shaft friction coefficient

Act = [0 0 1 0;
        0 0 0 1;
        -Ktheta/(J_i*tau_g^2) Ktheta/(J_i*tau_g) -(beta_i+Kt^2/R)/J_i 0;
        Ktheta/(J_o*tau_g) -Ktheta/J_o 0 -beta_o/J_o];

Bct = [0;0;Kt/(J_i*R);0];

Cct = [0 1 0 0];

Dct = 0;

Ts  =   0.1;            % Sampling time

Model           =   c2d(ss(Act,Bct,Cct,Dct),Ts);  % Discrete-time model with zoh

% System matrices:
[A,B,C,D]       =   ssdata(Model);                  % Model matrices

%% Signal dimensions
nz      =   size(A,1);
nu       =   size(B,2);
ny       =   size(C,1);

%% Prediction horizon and cost function weighting matrices
N        =   50;                   % prediction horizon
Q        =   1e4;                 % Output weights
R        =   1e-3;                % Input variation weights

%% Reference output and initial state
z0      =   0*ones(nz,1);
yref     =   10*ones(ny,1);

%% Build overall matrices and vectors for KKT system
[Lambda_y,Gamma_y,Lambda_z,Gamma_z]    =   Traj_matrices(N,A,B,C,D);
```

```
Qbar                                    =   zeros((N+1)*ny);
Rbar                                    =   zeros(N*nu);
Yref                                    =   zeros((N+1)*ny,1);

for ind = 1:N+1
    Qbar((ind-1)*ny+1:ind*ny,(ind-1)*ny+1:ind*ny)   =   Q;
    Yref((ind-1)*ny+1:ind*ny,1)                     =   yref;
end

for ind = 1:N
    Rbar((ind-1)*nu+1:ind*nu,(ind-1)*nu+1:ind*nu)   =   R;
end

P           =   [Gamma_y'*Qbar*Gamma_y+Rbar -(Gamma_z(end-nz+1:end,:)-Gamma_z(end-2*nz+1:↙
end-nz,:))';
                Gamma_z(end-nz+1:end,:)-Gamma_z(end-2*nz+1:end-nz,:) zeros(nz)];
S           =   [-Gamma_y'*Qbar*Lambda_y Gamma_y'*Qbar;
                -(Lambda_z(end-nz+1:end,:)-Lambda_z(end-2*nz+1:end-nz,:)) zeros(nz,(N+1)↙
*ny)];

Gains       =   P\S;

%% Compute optimal input sequence - terminal equality constraint - reference tracking
sol         =   Gains*[z0;Yref];
U           =   sol(1:nu*N,1);

%% Plot results
Z          =   Lambda_z*z0+Gamma_z*U;
figure(1),subplot(3,1,2),plot(Ts*[0:1:N-1]',U),grid on, hold on,title('Input voltage')
xlabel('Time (s)')
subplot(3,1,1),plot(Ts*[0:1:N]',Lambda_y*z0+Gamma_y*U),grid on, hold on,title('Output↙
position')
xlabel('Time (s)')
subplot(3,1,3),plot(Ts*[0:1:N]',Ktheta*(Z(1:nz:end)/tau_g-Z(2:nz:end))),grid on, hold↙
on,title('Torsional stiffness')
xlabel('Time (s)')

%% Check terminal equality constraint
Z          =   Lambda_z*z0+Gamma_z*U;
Z(end-2*nz+1:end-nz)-Z(end-nz+1:end);

%% Extract feedback gain
Keq         =   Gains(1:nu,1:nz);

%% Compute optimal input sequence - terminal equality constraint - regulation to the↙
origin
z0         =   [tau_g*10;10;0;0];
yref        =   zeros(ny,1);
Yref        =   zeros((N+1)*ny,1);

for ind = 1:N+1
    Yref((ind-1)*ny+1:ind*ny,1) =   yref;
end

sol         =   Gains*[z0;Yref];
```

```
U       =   sol(1:nu*N,1);

%% Plot results
Z      =   Lambda_z*z0+Gamma_z*U;
figure(2),subplot(3,1,2),plot(Ts*[0:1:N-1]',U),grid on, hold on,title('Input voltage')
xlabel('Time (s)')
subplot(3,1,1),plot(Ts*[0:1:N]',Lambda_y*z0+Gamma_y*U),grid on, hold on,title('Output↙
position')
xlabel('Time (s)')
subplot(3,1,3),plot(Ts*[0:1:N]',Ktheta*(Z(1:nz:end)/tau_g-Z(2:nz:end))),grid on, hold↙
on,title('Torsional stiffness')
xlabel('Time (s)')

%% Simulate with state feedback (explicit feedback law)
Nsim                =   150;
Z_fb               =   zeros(Nsim+1*nz,1);
Y_fb                =   zeros(Nsim+1*ny,1);
U_fb                =   zeros(Nsim*nu,1);
Z_fb(1:nz,1)      =   z0;
Y_fb(1:ny,1)        =   (C+D*Keq)*z0;
U_fb(1:nu,1)        =   Keq*z0;
for ind = 2:Nsim+1
    Z_fb((ind-1)*nz+1:ind*nz,1)  =   (A+B*Keq)*Z_fb((ind-2)*nz+1:(ind-1)*nz,1);
    U_fb((ind-1)*nu+1:ind*nu,1)     =   Keq*Z_fb((ind-1)*nz+1:ind*nz,1);
    Y_fb((ind-1)*ny+1:ind*ny,1)     =   (C+D*Keq)*Z_fb((ind-1)*nz+1:ind*nz,1);
end
figure(3),subplot(3,1,2),plot(Ts*[0:1:Nsim]',U_fb),grid on, hold on,title('Input↙
voltage')
xlabel('Time (s)')
subplot(3,1,1),plot(Ts*[0:1:Nsim]',Z_fb(2:nz:end)),grid on, hold on,title('Output↙
position')
xlabel('Time (s)')
subplot(3,1,3),plot(Ts*[0:1:Nsim]',Ktheta*(Z_fb(1:nz:end)/tau_g-Z_fb(2:nz:end))),grid↙
on, hold on,title('Torsional stiffness')
xlabel('Time (s)')

% Compare with LQR simulation
Qlqr                =   zeros(nz);
Qlqr(2,2)           =   Q;
Klqr                =   -dlqr(A,B,Qlqr,R);
Z_lqr              =   zeros(Nsim+1*nz,1);
Y_lqr               =   zeros(Nsim+1*ny,1);
U_lqr               =   zeros(Nsim*nu,1);
Z_lqr(1:nz,1)     =   z0;
Y_lqr(1:ny,1)       =   (C+D*Klqr)*z0;
U_lqr(1:nu,1)       =   Keq*z0;
for ind = 2:Nsim+1
    Z_lqr((ind-1)*nz+1:ind*nz,1) =   (A+B*Klqr)*Z_lqr((ind-2)*nz+1:(ind-1)*nz,1);
    U_lqr((ind-1)*nu+1:ind*nu,1)     =   Keq*Z_lqr((ind-1)*nz+1:ind*nz,1);
    Y_lqr((ind-1)*ny+1:ind*ny,1)    =   (C+D*Klqr)*Z_lqr((ind-1)*nz+1:ind*nz,1);
end
figure(3),subplot(3,1,2),plot(Ts*[0:1:Nsim]',U_lqr)
legend('Explicti QP solution','LQR')
subplot(3,1,1),plot(Ts*[0:1:Nsim]',Z_lqr(2:nz:end))
legend('Explicti QP solution','LQR')
```

```
subplot(3,1,3),plot(Ts*[0:1:Nsim]',Ktheta*(Z_lqr(1:nz:end)/tau_g-Z_lqr(2:nz:end)))
legend('Explicti QP solution','LQR')
```

```
% Constrained Numerical Optimization for Estimation and Control
% Laboratory session F
% Script to formulate and solve a Finite Horizon Optimal Control Problem
% for the linear servo-mechanism example, with inequality constraints and
% terminal equality constraint

clear all
close all
clc

%% System model
% Model parameters
R       =   20;             % Electric resistance
Ktheta  =   1280;           % Torsional stiffness output shaft
Kt      =   10;             % Motor constant
tau_g   =   20;             % Gear ratio
J_i     =   0.5;            % Input shaft moment of inertia
J_o     =   25;             % Output shaft moment of inertia
beta_i  =   0.1;            % Input shaft friction coefficient
beta_o  =   25;             % Output shaft friction coefficient
Tbar    =   78.5;           % Maximum torsional moment
Vbar    =   220;            % Maximum input voltage

Act = [0 0 1 0;
        0 0 0 1;
        -Ktheta/(J_i*tau_g^2) Ktheta/(J_i*tau_g) -(beta_i+Kt^2/R)/J_i 0;
        Ktheta/(J_o*tau_g) -Ktheta/J_o 0 -beta_o/J_o];

Bct = [0;0;Kt/(J_i*R);0];

Cct = [0 1 0 0];

Dct = 0;

Ts  =   0.1;            % Sampling time

Model           =   c2d(ss(Act,Bct,Cct,Dct),Ts);  % Discrete-time model with zoh

% System matrices:
[A,B,C,D]       =   ssdata(Model);                  % Model matrices

%% Signal dimensions
nz     =   size(A,1);
nu      =   size(B,2);
ny      =   size(C,1);

%% Prediction horizon and cost function weighting matrices
N       =   50;
Q       =   1e3;
R       =   1e-3;

%% Inequality constraints
% Input inequalities
Au      =   [eye(nu);-eye(nu)];
bu      =   Vbar*ones(2*nu,1);
```

```
nqu     =   size(Au,1);                   % Number of input inequality constraints per↙
stage

% State inequalities
Az      =   [Ktheta/tau_g -Ktheta 0 0;-Ktheta/tau_g +Ktheta 0 0];
bz      =   Tbar*ones(2,1);
nqz     =   size(Az,1);

%% Reference output and initial state
z0      =   zeros(nz,1);
yref     =   10*ones(ny,1);

%% Build overall matrices and vectors for QP (note - quadprog solves: min 0.5*x'*H*x +↙
f'*x   subject to:  A*x <= b, Aeq*x = beq)
[Lambda_y,Gamma_y,Lambda_z,Gamma_z]   =   Traj_matrices(N,A,B,C,D);
Qbar                                    =   zeros((N+1)*ny);
Rbar                                    =   zeros(N*nu);
Yref                                    =   zeros((N+1)*ny,1);
Aubar                                   =   zeros(N*nqu,N*nu);
bubar                                   =   zeros(N*nqu,1);
Azbar                                  =   zeros((N+1)*nqz,(N+1)*nz);
bzbar                                  =   zeros((N+1)*nqz,1);

for ind = 1:N+1
    Qbar((ind-1)*ny+1:ind*ny,(ind-1)*ny+1:ind*ny)           =   Q;
    Yref((ind-1)*ny+1:ind*ny,1)                             =   yref;
    Azbar((ind-1)*nqz+1:ind*nqz,(ind-1)*nz+1:ind*nz)   =   Az;
    bzbar((ind-1)*nqz+1:ind*nqz,1)                       =   bz;
end

for ind = 1:N
    Rbar((ind-1)*nu+1:ind*nu,(ind-1)*nu+1:ind*nu)           =   R;
    Aubar((ind-1)*nqu+1:ind*nqu,(ind-1)*nu+1:ind*nu)        =   Au;
    bubar((ind-1)*nqu+1:ind*nqu,1)                          =   bu;
end

Aineq    =   [Aubar;Azbar*Gamma_z];
bineq    =   [bubar;bzbar-Azbar*Lambda_z*z0];

% Terminal equality constraint
Aeq      =   Gamma_z(end-nz+1:end,:)-Gamma_z(end-2*nz+1:end-nz,:);
beq      =   -(Lambda_z(end-nz+1:end,:)-Lambda_z(end-2*nz+1:end-nz,:))*z0;

% Cost function
f        =   z0'*Lambda_y'*Qbar*Gamma_y-Yref'*Qbar*Gamma_y;
H        =   (Gamma_y'*Qbar*Gamma_y)+Rbar;
H        =   0.5*(H+H');

%% Solve QP
options =   optimset('display','none');

tic
% U        =   quadprog(H,f,Aineq,bineq,Aeq,beq,[],[],[],options);
U        =   quadprog(H,f,Aineq,bineq,[],[],[],[],[],options);
```

```
toc
%% Plot results
Z      =   Lambda_z*z0+Gamma_z*U;

figure(1),subplot(3,1,2),plot(Ts*[0:1:N-1]',U),grid on, hold on,title('Input voltage')
xlabel('Time (s)')
subplot(3,1,1),plot(Ts*[0:1:N]',Z(2:nz:end)),grid on, hold on,title('Output position')
xlabel('Time (s)')
subplot(3,1,3),plot(Ts*[0:1:N]',Ktheta*(Z(1:nz:end)/tau_g-Z(2:nz:end))),grid on, hold↙
on,title('Torsional stiffness')
xlabel('Time (s)')
```

```
% Constrained Numerical Optimization for Estimation and Control
% Laboratory session F
% Script to set up and simulate a linear-quadratic MPC strategy
% for the linear servo-mechanism example, with inequality constraints and
% terminal equality constraint

clear all
close all
clc

%% System model
% Model parameters
R        =   20;             % Electric resistance
Ktheta   =   1280;           % Torsional stiffness output shaft
Kt       =   10;             % Motor constant
tau_g    =   20;             % Gear ratio
J_i      =   0.5;            % Input shaft moment of inertia
J_o      =   25;             % Output shaft moment of inertia
beta_i   =   0.1;            % Input shaft friction coefficient
beta_o   =   25;             % Output shaft friction coefficient
Tbar     =   78.5;           % Maximum torsional moment
Vbar     =   220;            % Maximum input voltage

Act = [0 0 1 0;
        0 0 0 1;
        -Ktheta/(J_i*tau_g^2) Ktheta/(J_i*tau_g) -(beta_i+Kt^2/R)/J_i 0;
        Ktheta/(J_o*tau_g) -Ktheta/J_o 0 -beta_o/J_o];

Bct = [0;0;Kt/(J_i*R);0];

Cct = [0 1 0 0];

Dct = 0;

Ts  =   0.1;            % Sampling time

Model           =   c2d(ss(Act,Bct,Cct,Dct),Ts);  % Discrete-time model with zoh

% System matrices:
[A,B,C,D]       =   ssdata(Model);                 % Model matrices

%% Signal dimensions
nz      =   size(A,1);
nu       =   size(B,2);
ny       =   size(C,1);

%% Prediction horizon and cost function weighting matrices
N        =   7;
Q        =   1e4;
R        =   1e-5;

%% Inequality constraints
% Input inequalities
Au       =   [eye(nu);-eye(nu)];
bu       =   Vbar*ones(2*nu,1);
```

```
nqu     =   size(Au,1);                  % Number of input inequality constraints per↙
stage

% State inequalities
Az     =   [Ktheta/tau_g -Ktheta 0 0;-Ktheta/tau_g +Ktheta 0 0];
bz     =   Tbar*ones(2,1);
nqz    =   size(Az,1);

%% Reference output and initial state
z0     =   zeros(nz,1);
yref    =   10*ones(ny,1);

%% Build overall matrices and vectors for QP (note - quadprog solves: min 0.5*x'*H*x +↙
f'*x   subject to:  A*x <= b, Aeq*x = beq)
[Lambda_y,Gamma_y,Lambda_z,Gamma_z]    =   Traj_matrices(N,A,B,C,D);
Qbar                                    =   zeros((N+1)*ny);
Rbar                                    =   zeros(N*nu);
Yref                                    =   zeros((N+1)*ny,1);
Aubar                                   =   zeros(N*nqu,N*nu);
bubar                                   =   zeros(N*nqu,1);
Azbar                                  =   zeros((N+1)*nqz,(N+1)*nz);
bzbar                                  =   zeros((N+1)*nqz,1);

for ind = 1:N+1
    Qbar((ind-1)*ny+1:ind*ny,(ind-1)*ny+1:ind*ny)           =   Q;
    Yref((ind-1)*ny+1:ind*ny,1)                             =   yref;
    Azbar((ind-1)*nqz+1:ind*nqz,(ind-1)*nz+1:ind*nz)    =   Az;
    bzbar((ind-1)*nqz+1:ind*nqz,1)                        =   bz;
end

for ind = 1:N
    Rbar((ind-1)*nu+1:ind*nu,(ind-1)*nu+1:ind*nu)           =   R;
    Aubar((ind-1)*nqu+1:ind*nqu,(ind-1)*nu+1:ind*nu)        =   Au;
    bubar((ind-1)*nqu+1:ind*nqu,1)                          =   bu;
end

Aineq   =   [Aubar;Azbar*Gamma_z];
bineq   =   [bubar;bzbar-Azbar*Lambda_z*z0];

% Terminal equality constraint
Aeq     =   Gamma_z(end-nz+1:end,:)-Gamma_z(end-2*nz+1:end-nz,:);
beq     =   -(Lambda_z(end-nz+1:end,:)-Lambda_z(end-2*nz+1:end-nz,:))*z0;

% Cost function
f       =   z0'*Lambda_y'*Qbar*Gamma_y-Yref'*Qbar*Gamma_y;
H       =   (Gamma_y'*Qbar*Gamma_y)+Rbar;
H       =   0.5*(H+H');

%% QP options
options =   optimset('display','none');

%% Simulate with MPC
Nsim                    =   150;
Zsim_MPC               =   zeros((Nsim+1)*nz,1);
Ysim_MPC                =   zeros(Nsim*ny,1);
```

```
Usim_MPC              =   zeros(Nsim*nu,1);
Zsim_MPC(1:nz,1)   =   z0;
zt                   =   z0;
tQP                   =   zeros(Nsim-1,1);

for ind=2:Nsim+1
    bineq                              =   [bubar;bzbar-Azbar*Lambda_z*zt];
    beq                                =   -(Lambda_z(end-nz+1:end,:)-Lambda_z(end-↙
2*nz+1:end-nz,:))*zt;
    f                                  =   zt'*Lambda_y'*Qbar*Gamma_y-↙
Yref'*Qbar*Gamma_y;
    tic
    U                                  =   quadprog(H,f,Aineq,bineq,Aeq,beq,[],[],[],↙
options);
    tQP(ind-1,1)                       =   toc;
    Usim_MPC((ind-2)*nu+1:(ind-1)*nu,1) =   U(1:nu,1);
    Zsim_MPC((ind-1)*nz+1:ind*nz,1)   =   A*Zsim_MPC((ind-2)*nz+1:(ind-1)*nz,1)↙
+B*Usim_MPC((ind-2)*nu+1:(ind-1)*nu,1);
    Ysim_MPC((ind-2)*ny+1:(ind-1)*ny,1) =   C*Zsim_MPC((ind-2)*nz+1:(ind-1)*nz,1)↙
+D*Usim_MPC((ind-2)*nu+1:(ind-1)*nu,1);
    zt                                =   Zsim_MPC((ind-1)*nz+1:ind*nz,1);
end
figure(1),subplot(3,1,1),plot(Ts*[0:1:Nsim-1],Zsim_MPC(2:nz:Nsim*nz)),grid on, hold↙
on,title('Output position')
xlabel('Time (s)')
subplot(3,1,2),plot(Ts*[0:1:(Nsim-1)]',Usim_MPC),grid on, hold on,title('Input↙
voltage')
xlabel('Time (s)')
subplot(3,1,3),plot(Ts*[0:1:Nsim]',Ktheta*(Zsim_MPC(1:nz:end)/tau_g-Zsim_MPC(2:nz:↙
end)))
grid on, hold on,title('Torsional stiffness')
xlabel('Time (s)')
figure(2),plot(Ts*[0:1:(Nsim-1)]',tQP),grid on, hold on,title('QP solution time (s)')

%% Compare with MPC without terminal equality constraint
Zsim_MPC_noeq              =   zeros((Nsim+1)*nz,1);
Ysim_MPC_noeq               =   zeros(Nsim*ny,1);
Usim_MPC_noeq               =   zeros(Nsim*nu,1);
Zsim_MPC_noeq(1:nz,1)      =   z0;
zt                          =   z0;

for ind=2:Nsim+1
    bineq                                        =   [bubar;bzbar-Azbar*Lambda_z*zt];
    f                                            =   zt'*Lambda_y'*Qbar*Gamma_y-↙
Yref'*Qbar*Gamma_y;
    tic
    U                                            =   quadprog(H,f,Aineq,bineq,[],[],[],↙
[],[],options);
    Usim_MPC_noeq((ind-2)*nu+1:(ind-1)*nu,1)     =   U(1:nu,1);
    Zsim_MPC_noeq((ind-1)*nz+1:ind*nz,1)      =   A*Zsim_MPC_noeq((ind-2)*nz+1:(ind-1)↙
*nz,1)+B*Usim_MPC_noeq((ind-2)*nu+1:(ind-1)*nu,1);
    Ysim_MPC_noeq((ind-2)*ny+1:(ind-1)*ny,1)     =   C*Zsim_MPC_noeq((ind-2)*nz+1:(ind-↙
1)*nz,1)+D*Usim_MPC_noeq((ind-2)*nu+1:(ind-1)*nu,1);
    zt                                          =   Zsim_MPC_noeq((ind-1)*nz+1:ind*nz,↙
1);
```

```
end
figure(1),subplot(3,1,1),plot(Ts*[0:1:Nsim-1],Zsim_MPC_noeq(2:nz:Nsim*nz))
legend('MPC with terminal constraint','MPC without terminal constraint')
subplot(3,1,2),plot(Ts*[0:1:Nsim-1]',Usim_MPC_noeq)
legend('MPC with terminal constraint','MPC without terminal constraint')
subplot(3,1,3),plot(Ts*[0:1:Nsim]',Ktheta*(Zsim_MPC_noeq(1:nz:end)/tau_g-Zsim_MPC_noeq↙
(2:nz:end)))
legend('MPC with terminal constraint','MPC without terminal constraint')
```