Convex Optimization and Application
Mechanical Engineering
Computer Science
Physics

Robust Optimal Control of Linear Discrete-Time Systems

A modern implementation using CVX

Lecturer :

Hervé Lebret

Students :

Nicolas Lefebure
Damian Dudzicz
Niels Mortensen

June 10, 2019

Abstract

In this report, we present a new comparative study of the paper from Hansson and Boyd *Robust Optimal Control of Linear Discrete-Time Systems using Primal-Dual Interior-Point Methods*, June 1998 [1] using modern optimization programming tools such as the CVX library for MATLAB. We consider the same problem setup consisting of a double-tank discrete-time process control system as in [1]. We present the reader with our solution written in MATLAB and using the CVX library [2] for this setting. We compare the performances of our implementation with those of [1] dating form 1998 and run on a personal workstation of this period to a Lenovo Thinkpad T450s with a 2.3 Ghz Intel i5 Dual Core CPU with 8 GB of RAM. The aim of the report is not derive and implement the Interior-Point method presented in [1] but to consider the advantages of a recent solution which uses CVX in terms of ease of use and performances over a more evolved techniques for a specific optimization problem but executed on older hardware. We show the practicality and efficiency for the considered problem of a more simple and naive solution thanks to the progess of computation power over the last 20 years.

Contents

1 Introduction **3**

2 Control Problem **3**

3 Example **6**

4 Matlab Implementation **10**

5 Results **15**

5.1 Numerical Results 15

5.2 Computational Performance Results 17

6 Conclusion **19**

1 Introduction

In [1] authors describe how to efficiently solve a robust optimal control problem using primal-dual Interior-Point method. The solution is adapted to computation capacities available on a personal workstation from 1998.

In our report, we consider this same specific optimization problem as in [1]. However; instead of implementing the Interior-Point Method introduced by Hannsson and Boyd, we choose to use the CVX library developped by Andersen, Vandenberghe, Grant and Boyd.

The report is organised as follows. In section 2, we explain the modelling of a general Linear Discrete-Time System and how to adapt it as an optimization problem. In section 3, we introduce the considered control system optimization problem along with an explanation of the derivation presented in [1]. In section 4, we presented the Matlab implementation of code along an explanation of the design choices. In section 5, we compare numerical results obtained with our solution to those yielded in [1] in addition we perform benchmarking tests on the computational performance of our implementation. Eventually, in section 6 we state some concluding remarks regarding the considered problem and the results presented in the report.

2 Control Problem

Any linear or linearized (we develop a linearization method in section 3) and time-varying dynamical system can be written as a set of differential first order equations. Then, the system's evolution can be discretized (we also provide the reader with a discretization method in the section 3) for computer implementation. If one consider $i = 1, \dots, L$ models, those can be expressed as

$$x_i(k+1) = A_i(k)x_i(k) + B_i(k)u(k) \;\; k = 0, \dots, N-1 \tag{1}$$

where $A_i(k) \in \mathbb{R}^{n \times n}$ are the system's evolution parameters, and $x_i(k) \in \mathbb{R}^n$ is its current state. $u(k) \in \mathbb{R}^m$ is the control signal, which modifies the curse of the system's evolution with an influence characterised by the parameters in the matrix $B_i(k) \in \mathbb{R}^{n \times m}$.

By definition, the control signal is providing an external influence one can apply on the system's state at each timestep, and is fully set by the user. The encountered optimization problem is to find out which control sequence to apply over time, in order to minimize a given objective on the system's output, while satisfying some constraints. The output of the system is defined as

$$y_i(k) = C_i(k)x_i(k) + D_i(k)u(k) \;\; k = 0, \dots, N \tag{2}$$

where $y_i(k) \in \mathbb{R}^p$ is the output signal, $C_i(k) \in \mathbb{R}^{p \times n}$ the output parameters and $D_i(k) \in \mathbb{R}^{p \times m}$ modelling a possible direct influence of the control on the output.

For optimization purposes, it is possible to set up constraints on the output of the system. These constraints depend on the physical limits of the system or the sensors that are used. We can write them as:

$$d_i(k) \geq C_i(k)x_i(k) + D_i(k)u(k) \;\; k = 0, \dots, N \tag{3}$$

Hence the systems set of $i = 1, \dots, L$ is

$$\begin{aligned} x_i(k+1) &= A_i(k)x_i(k) + B_i(k)u(k) \;\; k = 0, \dots, N-1 \\ d_i(k) &\geq C_i(k)x_i(k) + D_i(k)u(k) \;\; k = 0, \dots, N \end{aligned} \tag{4}$$

In order to find the optimal control sequence, it is possible to use a general performance criteria function defined as

$$\begin{aligned}\phi_i &= \frac{1}{2}\sum_{k=0}^{N}\begin{pmatrix} x_i(k) \\ u(k)\end{pmatrix}^T\begin{pmatrix} Q_i(k) & S_i(k) \\ S_i(k)^T & R_i(k)\end{pmatrix}\begin{pmatrix} x_i(k) \\ u(k)\end{pmatrix}\ i = 0,\ldots,L \\ &= \frac{1}{2}\sum_{k=0}^{N} x_i(k)^T Q_i(k) x_i(k) + u(k)^T R_i(k) u(k) + 2x_i(k)^T S_i(k) u(k)\end{aligned} \tag{5}$$

where $Q_i(k) \in \mathbb{R}^{n\times n}$, $R_i(k) \in \mathbb{R}^{m\times m}$ and $S_i(k) \in \mathbb{R}^{n\times m}$. This performance criteria function can also be seen as a cost function that characterises the outlay of your system going to the final state. The parameters $Q_i(k)$ and $R_i(k)$ can be used as design parameters to penalize the state variables and the control signals. The larger these values are, the more you penalize these signals. The matrix $Q_i(k)$ represents how important a penalty is it if the state $x_i(k)$ is not close to the expected, while $R_i(k)$ corresponds to the amount of energy $u(k)$ that needs to be spent to bring the system to its final state.

Once the performance criteria determined, the optimization problem can be expressed as

$$\begin{aligned}&\text{minimize } \gamma \\ &\text{subject to } \phi_i \leq \gamma \quad i = 0,\ldots,L\end{aligned} \tag{6}$$

Consequently, the concern is to turn the robust optimal control problem into a robust quadratic optimization problem that can be solved using the CVX library. In order to achieve the latter, we introduce the extended state and system matrices:

$$A(k) = \begin{pmatrix} A_1(k) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_L(k)\end{pmatrix}$$

$$C(k) = \begin{pmatrix} C_1(k) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_L(k)\end{pmatrix}$$

$$B(k) = \begin{pmatrix} B_1(k) \\ \vdots \\ B_L(k)\end{pmatrix}; D(k) = \begin{pmatrix} D_1(k) \\ \vdots \\ D_L(k)\end{pmatrix}$$

$$x(k) = \begin{pmatrix} x_1(k) \\ \vdots \\ x_L(k)\end{pmatrix}; \quad d(k) = \begin{pmatrix} d_1(k) \\ \vdots \\ d_L(k)\end{pmatrix}$$

The system can be rewritten as

$$\begin{aligned}x(k+1) &= A(k)x(k) + B(k)u(k),\ \ k = 0,\ldots,N-1 \\ d(k) &\geq C(k)x(k) + D(k)u(k),\ \ k = 0,\ldots,N\end{aligned} \tag{7}$$

We define

$$x = \begin{pmatrix} x(0) \\ u(0) \\ \vdots \\ x(L) \\ u(L) \end{pmatrix}; \quad \overline{Q}_i(k) = \begin{pmatrix} Q_i(k) & S_i(k) \\ S_i(k)^T & R_i(k) \end{pmatrix} \tag{8}$$

$$E_i = \begin{pmatrix} 0 & \cdots & 0 & I_n & 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & I_m \end{pmatrix}$$

The cost function can be defined as

$$\phi_i = \frac{1}{2} x^T Q_i x \tag{9}$$

where Q_i is given by

$$Q_i = \begin{pmatrix} E_i(0)^T \overline{Q}_i(0) E_i(0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & E_i(N)^T \overline{Q}_i(N) E_i(N) \end{pmatrix} \tag{10}$$

Then,

$$F = \begin{pmatrix} I & 0 & 0 & 0 & \cdots & 0 \\ -A(0) & -B(0) & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & -A(N-1) & -B(N-1) & I & 0 \end{pmatrix} \tag{11}$$

$$g = \begin{pmatrix} x^T(0) & 0 & \cdots & 0 \end{pmatrix}^T \tag{12}$$

$$C = \begin{pmatrix} C(0) & D(0) & \cdots & \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & C(N) & D(N) \end{pmatrix} \tag{13}$$

$$d = \begin{pmatrix} d^T(0) & d(1)^T & \cdots & d(N)^T \end{pmatrix}^T \tag{14}$$

Eventually, the optimization problem has the following form

$$\begin{aligned} &\text{minimize } \gamma \\ &\text{subject to } x^T Q_i x \leq \gamma \quad i = 0, \ldots, L \\ &Fx = g; \quad Cx \leq d \end{aligned} \tag{15}$$

It is important to state that $Q_i = Q_i^T \succcurlyeq 0$ thus making this a convex optimization problem. Indeed, the main function to minimize is $J = \frac{1}{2} x^T Q_i x$. The Gradient of J is therefore $\nabla J = 2 Q_i x$ and the corresponding Hessian is $\nabla^2 J = Q_i$. It is a well-known property that, the function J is convex if and only if its Hessian is non-negative which confirms the statement made previously.

Based on this first analysis of the general setting, Hannsson and Boyd present in [1] the KKT conditions associated to the problem along with an Interior-Point method to efficiently compute the search direction of the Newton step. In this report, we decide not to pursue those derivation and use directly the CVX library to solve the problem taking advantages of the convenience and ease of use of the tool while comparing its usability especially in terms of computation time required to achieve a solution for the setting with the same parameters as specified in [1].

3 Example

In order to test the performances of the CVX optimization toolbox, we study the same double-tank process setting as it is described in [1]. The system is represented in fig.1. The objective of the control design is to achieve a desired water level in the lower tank by adjusting the water supply to the first tank driven by the pump, and the water level set-point of the first tank.

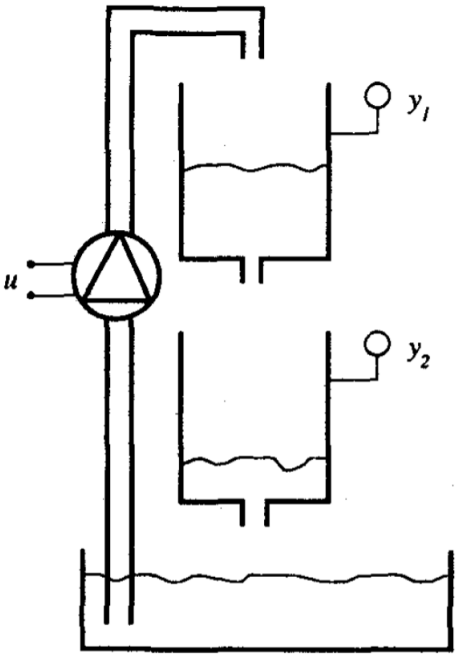

Figure 1: Double-tank-process

The dynamical equations of the system can be written as follows. The idea is to write a mass-conservation law for each tank

$$\begin{aligned} \dot{m} &= m_{in} - m_{out} \\ \rho A\dot{h} &= \rho(q_{in} - q_{out}) \end{aligned} \tag{16}$$

where ρ is the mass density of the liquid transiting, A the cross-section of the tank, h the varying height of the liquid in the tank and q_{in} and q_{out} the in and out flow of the tank. If we apply this equation to the first tank, we find

$$A_1\dot{h_1} = q - q_{out} \tag{17}$$

where q the in-flow is given by $q = ku$, with u the voltage applied to the pump, and $k = 27{\cdot}10^{-7}\left[\frac{\mathrm{m}^3}{\mathrm{V}}\right]$ a proportionality constant. The out-flow can be expressed using Bernoulli's law

$$\frac{v^2}{2} + g \cdot z + \frac{P}{\rho} = \text{Cst.} \tag{18}$$

Applying the law to our setting, relatively to the surface of the liquid and the output of the first tank, we obtain the following relation

$$\frac{v_{in}^2}{2} + g \cdot z_{in} + \frac{P_{atm}}{\rho} = \frac{v_{out}^2}{2} + g \cdot z_{out} + \frac{P_{atm}}{\rho} \tag{19}$$

We approximate the speed of the water at the surface to be equal to zero as it is too slow to be considered ($v_{in} \approx 0$). We also set the reference of the height h_1 to the bottom of the tank yielding $z_{in} = h_1$ and $z_{out} = 0$. Using the previous assumptions, we obtain the following equation

$$g \cdot h_1 = \frac{v_{out}^2}{2} \Rightarrow v_{out} = \sqrt{2gh_1}$$

Eventually, we denote $q_{out} = a_1 \cdot v_{out} = a_1\sqrt{2gh_1}$, where a_1 is the area of the cross-section of the outlet of the first tank. Hence the dynamical equation of the first tank is $A_1\dot{h_1} = ku - a_1\sqrt{2gh_1}$. The equation of the second tank is derived in a similar fashion as for the first one. Consequently, the whole process is described as follows.

$$\begin{aligned} A_1\dot{h_1} &= ku - a_1\sqrt{2gh_1} \\ A_2\dot{h_2} &= a_2\sqrt{2gh_1} - a_2\sqrt{2gh_2} \end{aligned} \tag{20}$$

As the sensors of each tank measure an output voltage proportional to the height of the water inside ($y_i = k'h_i$, $k' = 50\left[\frac{\mathrm{V}}{\mathrm{m}}\right]$), it is possible to apply a variable change. The system can then be expressed as

$$\begin{aligned} \dot{y_1} &= \frac{kk'u}{A_1} - \frac{a_1k'}{A_1}\sqrt{\frac{2gy_1}{k'}} \\ \dot{y_2} &= \frac{a_1k'}{A_2}\sqrt{\frac{2gy_1}{k'}} - \frac{a_2k'}{A_2}\sqrt{\frac{2gy_2}{k'}} \end{aligned} \tag{21}$$

The obtained output is however non-linear. In order to linearize it, we apply a Taylor expansion around the steady-state solution

$$\begin{aligned} \dot{y} &= f(y,u) \\ \Delta\dot{y} &= f(\bar{y},\bar{u}) + \frac{\partial f}{\partial y}(\bar{y},\bar{u})(y-\bar{y}) + \frac{\partial f}{\partial u}(\bar{y},\bar{u})(u-\bar{u}) \\ \Delta\dot{y} &= f(\bar{y},\bar{u}) + \frac{\partial f}{\partial y}(\bar{y},\bar{u})\Delta y + \frac{\partial f}{\partial u}(\bar{y},\bar{u})\Delta u \\ \Delta\dot{y} &= A(\bar{y},\bar{u})\Delta y + B(\bar{y},\bar{u})\Delta u \end{aligned} \tag{22}$$

For our case the system is

$$\dot{y} = \begin{pmatrix} -\frac{a_1k'}{A_1}\sqrt{\frac{2g}{k'}} & 0 \\ \frac{a_1k'}{A_2}\sqrt{\frac{2g}{k'}} & -\frac{a_2k'}{A_2}\sqrt{\frac{2g}{k'}} \end{pmatrix}\begin{pmatrix} \sqrt{y_1} \\ \sqrt{y_2} \end{pmatrix} + \begin{pmatrix} \frac{kk'}{A_1} \\ 0 \end{pmatrix}u \tag{23}$$

Given that the steady state solution of the water levels is $\bar{y} = (5,5)$ and $\bar{u} = 1.82$ for the voltage applied to the pump, the matrices A and B are then given by

$$A = \begin{pmatrix} \frac{\partial f_1}{\partial y_1}(\bar{y},\bar{u}) & \frac{\partial f_1}{\partial y_2}(\bar{y},\bar{u}) \\ \frac{\partial f_2}{\partial y_1}(\bar{y},\bar{u}) & \frac{\partial f_2}{\partial y_2}(\bar{y},\bar{u}) \end{pmatrix} = \begin{pmatrix} -\frac{a_1k'}{A_1}\sqrt{\frac{2g}{k'}}\frac{1}{2\sqrt{\bar{y_1}}} & 0 \\ \frac{a_1k'}{A_2}\sqrt{\frac{2g}{k'}}\frac{1}{2\sqrt{\bar{y_1}}} & -\frac{a_2k'}{A_2}\sqrt{\frac{2g}{k'}}\frac{1}{2\sqrt{\bar{y_2}}} \end{pmatrix} = \begin{pmatrix} -\alpha_1 & 0 \\ \alpha_{12} & -\alpha_2 \end{pmatrix} \tag{24}$$

$$B = \begin{pmatrix} \frac{\partial f_1}{\partial u}(\bar{y},\bar{u}) \\ \frac{\partial f_2}{\partial u}(\bar{y},\bar{u}) \end{pmatrix} = \begin{pmatrix} \frac{kk'}{A_1} \\ 0 \end{pmatrix} = \begin{pmatrix} \beta \\ 0 \end{pmatrix} \tag{25}$$

where it happens to be that $\alpha_1 = \alpha_2 = \alpha_{12} = 0.0179$ and $\beta = 0.0494$. The new system can then by concisely expressed as

$$\Delta\dot{y} = \begin{pmatrix} -\alpha_1 & 0 \\ \alpha_{12} & -\alpha_2 \end{pmatrix} \Delta y + \begin{pmatrix} \beta \\ 0 \end{pmatrix} \Delta u \tag{26}$$

and we can discretize the system with a sampling time of $T = 2$[s] s using the zero order hold method:

$$\begin{aligned} \Delta y(k+1) &= e^{AT}\Delta y(k) + \int_0^T e^{\Phi T} B dt \Delta u(k) \\ &= e^{AT}\Delta y(k) + A^{-1}(e^{AT} - I)B\Delta u(k) \\ &= \Phi\Delta y(k) + \Gamma\Delta u(k) \end{aligned} \tag{27}$$

Using Matlab command `expm()` to compute exponential of matrices, we obtain the follwoing numerical values

$$\Phi = \begin{pmatrix} 0.9648 & 0 \\ 0.0345 & 0.9648 \end{pmatrix} \tag{28}$$

$$\Gamma = \begin{pmatrix} 0.0971 \\ 0.0017 \end{pmatrix} \tag{29}$$

We need to set a constraint on the slew rate of the pump (limit $|\dot{u}|$) such that it set a limit (expressed in of Volts per seconds) for the speed at which the pump is able to change its regime. This guarantees the solution to stay realistic and stable. In order to achieve this, we need to augment the state of the system considered to be. This can be obtained by adding a variable which provides the system with a memory of what voltage is applied to the pump at the previous time-step of its evolution. This implies that the voltage doesn't vary more than a given threshold in the next time-step. This is fromalized in the following manner

$$\begin{pmatrix} \Delta y(k+1) \\ \Delta\xi(k+1) \end{pmatrix} = \begin{pmatrix} \Phi & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta y(k) \\ \Delta\xi(k) \end{pmatrix} + \begin{pmatrix} \Gamma \\ 1 \end{pmatrix} \Delta u(k) \tag{30}$$

where we can observe that the state variables are updated in the same fashion as previously, the new 'memory' variable is however updated for each next time-step to the current value of the voltage applied to the pump.

Now that the dynamics of the discrete-time system is done, we can impose its constraints:

$$\begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 8.18 \\ 1.82 \\ 2 \\ 2 \end{pmatrix} \succcurlyeq \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta y_1(k) \\ \Delta y_2(k) \\ \Delta\xi(k) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \Delta u(k) \tag{31}$$

where in the leftmost matrix after the inequality

- the first column expresses an upper and lower bound on the variation of the water level in the first tank which are equal to 5 in absolute value
- the second column expresses the same constraints but for the second tank

- lines 4 to 6 express the upper and lower bounds for the voltage applied to control the pump
- and the last two rows in the last column set the limitation of the evolution of u to 2 Volts per sampling period (in our case the period T is set to 2 s) both increasing and decreasingly. This induces a slew rate of 1 V/s.

Before solving the optimisation problem, it is important to define the performance criteria. The performance indexes considered are

$$\phi_i = \sum_{k=0}^{N} \Delta y_2^2(k),\ i = 1, 2, ..., L \tag{32}$$

It implies that the performance matrices is given by

$$Q_i = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix};\ S_i = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\ R_i = 0 \tag{33}$$

We can notice that the only variable solicited is Δy_2, as the aim of the system is to achieve steady-state (5 [V] corresponds to a water level of 0.1m) in the second tank.

Having formulated the optimization problem exactly as in the paper we are able to compare the solution yielded by our CVX implementation to the numerical results presented in [1].

4 Matlab Implementation

In this section we present the reader with the source code of our solution written in MATLAB and using the CVX library. The reasons for the choice of MATLAB over Python is due to two factors. The first one being purely practical as the CVX library for MATLAB has better integration with the MATLAB environment over its Python counterpart. Also, while trying to implement the solution in Python, some issues regarding Python2/Python3 compatibility with the NumPy and CVX libraries made very tedious and cumbersome the conversion of matrices in order to be able to use all required functions from both packets. The second reason lies in the desire to compare performances with respect to the same programming framework as used in [1] which is MATLAB.

Listing 1: Generic function used to solve the general problem case for any set of parameters

```
function [F, phi_Q, g, C, d_c, x, gam, timestamp] = OptRobustControl(A,B,C,D,d
    ,Q,R,S,T,L,N,sigma,u0,x0)
%Obtain the computation time required to run the the process
tic
timestamp = 0;

n = size(A);
n = n(1);
m = size(B);
m = m(2);
p = length(d);
A_L = zeros(n+1,L*(n + 1));
B_L = [zeros(n,L*m); ones(1,L*m)];

% (1) obtention of the differents indexes i of alpha1, alpha12 , etc by taking
     a
% random normally distributed term with relative standard deviation
% (2) transormation of the matrix A -> Phi to discretise the system
for i = 1:L
    for j = 1:n
        for p = 1:n
                A_L(j,3*(i-1)+p) = normrnd(A(j,p),sigma*abs(A(j,p)));
        end
    end
    A_L(1:n,1 + (i-1)*(n+1): i*(n+1)-1) = expm(A_L(1:n,1 + (i-1)*(n+1):i*(n+1)
        -1)*T);
end

% (1) obtention of the differents indexes i of beta by taking a
% random normally distributed term with relative standard deviation
% (2) transormation of the matrix B -> Gamma to discretise the system
for i = 1:L
     for j = 1:n
        for p = 1:m
                B_L(j,p-m+i*m) = normrnd(B(j,p),sigma*abs(B(j,p)));
        end
     end
```

```
    B_L(1:n,1 + m*(i-1):i*m) = inv(A_L(1:n,1 + (i-1)*(n+1):n + (i-1)*(n+1)))*(
        A_L(1:n,1 + (i-1)*(n+1):n + (i-1)*(n+1))*T - eye(n,n))*B_L(1:n,i*m);
end

n = n + 1;

% concatenation by Block diagonal of the Matrix A
A_Blk = [];
for i = 1:L
    A_Blk = blkdiag(A_Blk, A_L(:,1 + n*(i-1):i*n));
end

% concatenation by Block diagonal of the Matrix C
C_Blk = [];
for i = 1:L
    C_Blk = blkdiag(C_Blk, C);
end

% concatenation of Matrix B
B_Blk = [];
for i = 1:L
    B_Blk = [B_Blk; B_L(:,i)];
end

% concatenation of Matrix D
D_tmp = [];
D_Blk = [];
for i = 1:L
    D_tmp = D;
    D_Blk = [D_Blk; D_tmp];
end

% concatenation of vector d
d_tmp = [];
d_Blk = [];
for i = 1:L
    d_tmp = d;
    d_Blk = [d_Blk; d_tmp];
end

% Quadraric inegality constraint Matrix Qi concatenation
E_tmp = [];
Q_Blk = [];
phi_Q = [];

for i = 1:L
    for j = 1:N+1
        E_tmp = [zeros(n+m,n*(i-1)), [eye(n,n); zeros(1,n)], zeros(n+m,n*L-n*(
            i-1)-n),[zeros(n,m); eye(m,m)]];
```

```
        Q_bar = E_tmp'*[Q S; S' R]*E_tmp;
        Q_Blk = blkdiag(Q_Blk, Q_bar);
    end
    phi_Q = [phi_Q Q_Blk];
    Q_Blk = [];
end

% Linaer equality constraint Matrix F concatenation
F = [];
for i = 1:N
    F = blkdiag(F, [-A_Blk, -B_Blk]);
end

tmp1 = size(F);
tmp2 = size(A_Blk);

F = [zeros(tmp2(1),tmp1(2)+tmp2(2)+1); F, zeros(tmp1(1),tmp2(2)+1)];

tmp1 = size(F);
tmp3 = 0;
for i = 1:(tmp1(2)-N*m-1)/tmp2(2)
    for j = 1:tmp1(1)/tmp2(2)
        if (j == i )
        F((j-1)*tmp2(1)+1:j*tmp2(1),(i-1)*(tmp2(2)+m)+1:i*tmp2(2)+(i-1)*m) =
            eye(tmp2);
        tmp3 = tmp3 + 1;
        end
    end
end

% Linaer inequality constraint Matrix C concatenation
C = [];
for i = 1:N+1
    C = blkdiag(C,[C_Blk, D_Blk]);
end

% Linaer equality constraint vector g concatenation
g = [];
for i = 1:L
g = [g; x0];
end
g = [g; zeros(N*tmp2(1),1)];
g(n*L+1) = 4-1.82;

% Linaer inequality constraint vector d concatenation
d_tmp = [];
d_c = [];
for i = 1:N+1
    d_tmp = d_Blk;
```

```
    d_c = [d_c; d_tmp];
end

%CVX optimisation

cvx_begin
    variables gam(1) x((N+1)*(L*n+m));
    minimise(gam)
    subject to
        F*x == g;
        C*x <= d_c;
        x(n*L+1) == u0; % initial input condition added to constraints based
            on paper graph
        for i = 1:L
           x'*phi_Q(:,(i-1)*length(x)+1:i*length(x))*x <= gam;
        end
cvx_end

%Update of the timestamp
timestamp = toc;
end
```

The reader can observe that the code that solves the problem is relatively short and especially the actual optimization part. This shows one great advantage of the CVX library which is its ease of use and conciseness. The major part of the implementation lies in the preparation process which sets a proper model representation of the problem.

Listing 2: Code to solve the specific considered setting and to plot the corresponding results

```
clc;
close all
clear all

%Array to hold the yielded timestamps when benchmarking is performed
%For the deliverable code we decided to remove the for loop used for
%different values of the N and L parameters
timestamps = []

N = 50; % number of iterations
A1 = 2734e-6; %[m^2] Area of the cross-section of the first tank
A2 = 2734e-6; %[m^2] Area of the cross-section of the second tank
a1 = 7e-6; %[m^2] Area of of the cross-section of the first outlet
a2 = 7e-6; %[m^2] Area of of the cross-section of the second outlet
k_p = 50; %[V/m] Proportional constant between sensor and water height
k = 27e-7; %[m^3/Vs] Proportional contant of the pump
y_ss = 5; %[v] steady-state solution
u_ss = 1.82; %[V] steady-state solution
grv = 9.81; %[m.s^-2]
alpha1 = (-a1*k_p/A1)*sqrt(2*grv/k_p)*(1/2)*(1/sqrt(y_ss)); %[s^-1]
alpha12 = (a1*k_p/A2)*sqrt(2*grv/k_p)*(1/2)*(1/sqrt(y_ss)); %[s^-1]
alpha2 = (-a2*k_p/A2)*sqrt(2*grv/k_p)*(1/2)*(1/sqrt(y_ss)); %[s^-1]
```

```
beta = k*k_p/A1; %[S^-1]
A = [alpha1, 0; alpha12, alpha2]; %[S^-1]
B = [beta; 0]; %[S^-1]
d = [5; 5; 5; 5; 8.18; 1.82]; %[V]
C = [1 0 0;0 1 0;-1 0 0;0 -1 0; 0 0 0; 0 0 0];
D = [0 0 0 0 1 -1]';
Q = [0, 0, 0; 0, 2, 0; 0, 0, 0];
R = 0;
S = zeros(3,1);
T = 100/N; %[s] sampling time
L = 5; % Number of systems
sigma = 0.25; % relative standard deviation
u_0 = 4-1.82; %[V] initial condition
x_0 = [5-5; 1-5; 0]; %[V] initial condition

n = size(A);
n = n(1);
n = n+1;
m = size(B);
m = m(2);
p = length(d);

[F, phi_Q, g, C, d, x, gam,timestamp] = OptRobustControl(A,B,C,D,d,Q,R,S,T,L,N
    ,sigma,u_0,x_0);

t = (0:T:N*T); % Time vector

figure % Tanks level
xlabel('Time [s]')
ylabel('Water Level [V]')
for i = 1:3:L*n
    hold on
    plot(t,x(i:n*L+1:end)+y_ss);
end
for i = 2:3:L*n
    hold on
    plot(t,x(i:n*L+1:end)+y_ss);
end
hold off

figure % Pump voltage
xlabel('Time [s]')
ylabel('Pump Voltage [V]')
hold on
plot(t,x(n*L+1:n*L+1:end)+u_ss);
hold off
timestamps = [timestamps timestamp]
fileID = fopen('timestamps.txt','w');
fprintf(fileID,'%6.8f \n',timestamps);
```

```
71 fclose(fileID);
```

5 Results

5.1 Numerical Results

In this section we present the reader with the numerical results obtained with our Matlab implementation using CVX. The aim is to consider the same setting's specifications as in the experimental part of [1]. We plot the obtained values in the same fashion as Hannson and Boyd and we compare the obtained curves.

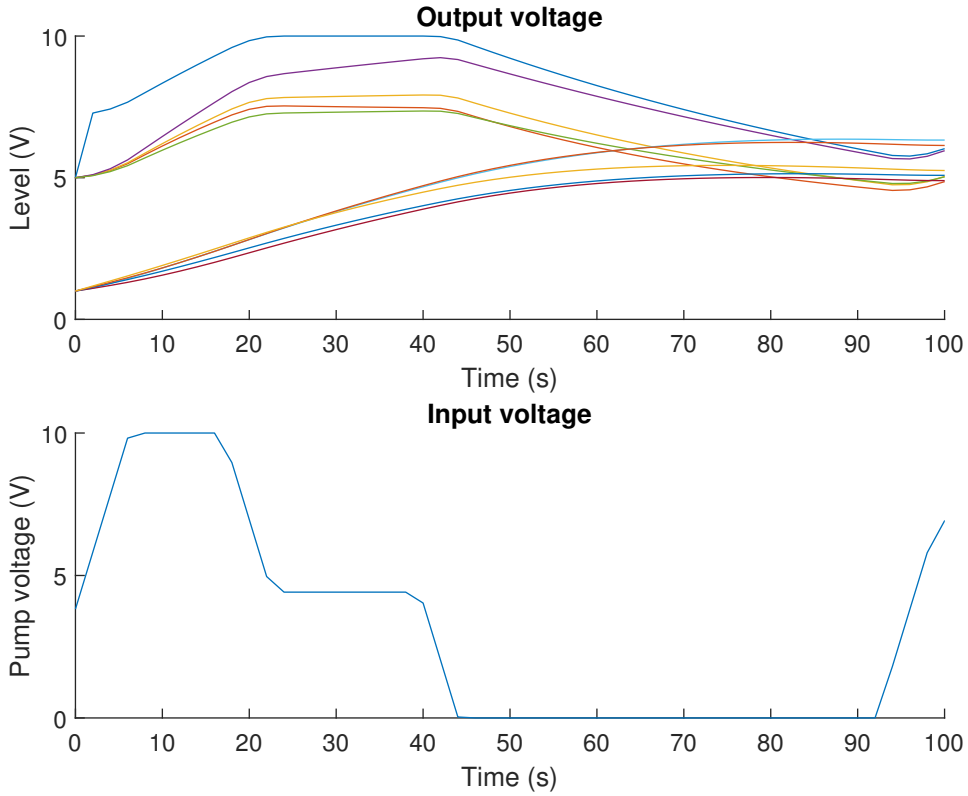

Figure 2: Plot of the performance of the robust optimal control signal for parameters $L = 5$, $N = 50$, $T = 2$ with initial value of $y(0) = [5 \quad 1]^T$ $[V]$.

In fig.2 the upper subplot shows the levels of the two tanks as a function of time for the $L = 5$ different models. The upper curves corresponds to the level in the first tank and the lower ones for those of the second tank.

We present the reader with the plots produced by Hansson and Boyd for the very same setting as considered in fig.2.

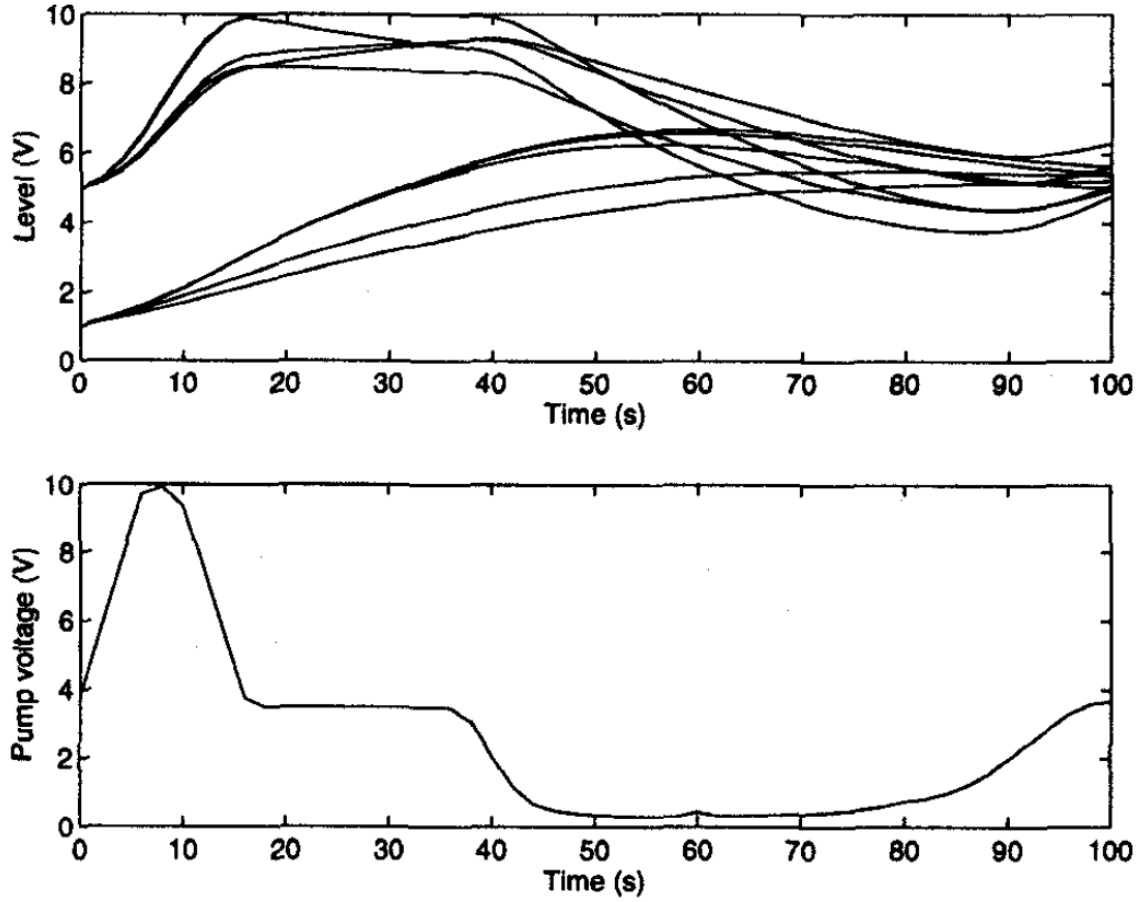

Figure 3: Plot of the performance of the robust optimal control signal taken from [1].

In both plots (figures 2 and 3), we can observe how the control sequence was optimized to bring the second tank to the steady-state value of 5 [V] corresponding to a water level of 0.1 [m] as fast as possible. At first, since the second tank is nearly empty, the pump fills the first tank with maximum slew-rate for less than 10 [s]. Both, pump voltage and level voltage in the first tank (for a given single model), reach their maximum which is at 10 [V]. As there is enough reserve in the first tank to fill the second tank, the voltage of the pump decreases slowly during the next 25 [s]. It attains its minimum of 0 [V] at an approximate time of 45 [s], letting the two tanks naturally reach the steady-state value. Eventually, around a time of $80 - 90$ [s], the pump gives a voltage impulse to compensate an offset error.

For longer runs of the control sequence, we observe the pump compensating for the errors with impulses while also achieving its steady-state value which is 1.82 [V]. In addition, we observe a final increase in the optimal control sequence which persists regardless of the length of the run, in addition to the voltage impulse at around $100[s]$. We can interpret this behaviour by the fact that the system doesn't plan on keeping satisfying the objectives after the time window considered in the optimization process is over. The system seizes the opportunity to reduce its costs for the end of the run by giving this final impulsion. Hence it approaches to the desired state, while not being affected by the disadvantages of the increase. This is due to the fact that those are not considered in the cost function given that they fall outside of the time window. This phenomenon can be observed in fig.4.

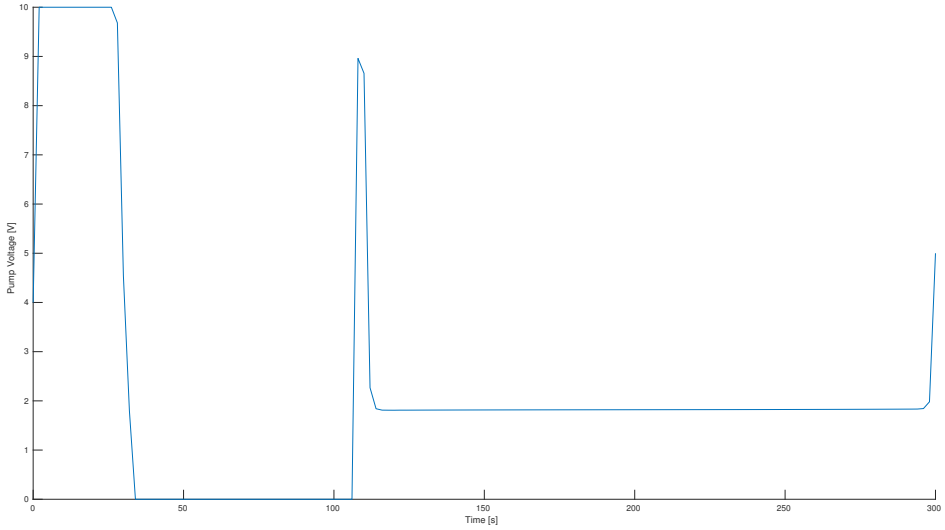

Figure 4: Plot of the performance of the robust optimal control signal for a longer execution with parameters $L = 5$, $N = 150$, $T = 2$ with initial value of $y(0) = [5 \quad 1]^T$ $[V]$.

One can observe that our numerical results obtained are very similar to those from Hannsson and Boyd. Thus this shows that our solution is quite similar in terms of produced values to the original solution presented in [1] while not requiring the use of the Interior-Point method which is mainly used for performances concerns as the computational capacities 20 years ago were obviously several orders smaller to those available nowadays.

It is interesting to note that the results obtained with our implementation (and probably also in the case of the original solution) can vary quite significantly due to the randomness induced by the sampling of the normal distribution realized at lines 20 and 32 in Listing 1. Thus some executions required to be restarted in order to yield a similar result as the one from fig 3.

5.2 Computational Performance Results

In this section, we present a comparative analysis of the performances of the two solutions with respect to computation time. As mentioned in section 4, we compare the executions of the two implementations written in the same scripting language MATLAB. However; this is the sole parameters that is shared among the two solutions. Indeed, neither the machine on which the code is run nor the the MATLAB version nor the actual implementation of the solution (with CVX for our case and an Interior-Point method for Hannson and Boyd) remains the same.

Nonetheless, this still has practical interest since it studies the possible advantages provided by modern software and hardware for efficiently solving convex optimization problems.

We present the reader with experimental performance results obtained while varying the number of model L and keeping the number of samples constant with $N = 50$ and $T = 2$ and also those yielded by maintaining the number of models constant $L = 5$ while varying the number of samples N and adjusting T correspondingly s.t. $T = \frac{100}{N}$ in order to keep the execution of the run to be $100[s]$. All experiments were carried on our personal computer a

Lenovo Thinkpad T450s with a 2.3 Ghz Intel i5 Dual Core CPU with 8GB of RAM.

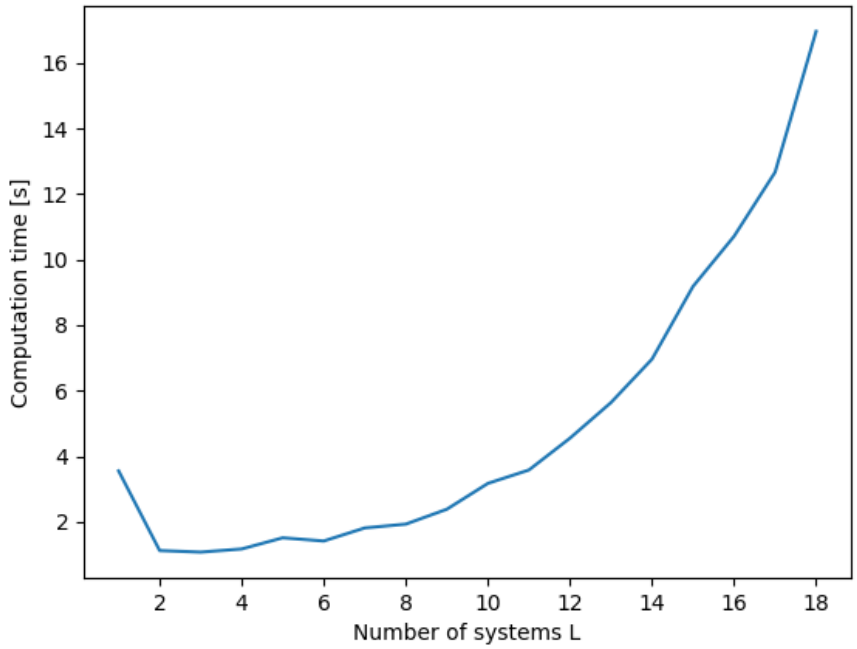

Figure 5: Computation time as a function of the number of models L for $N = 50$.

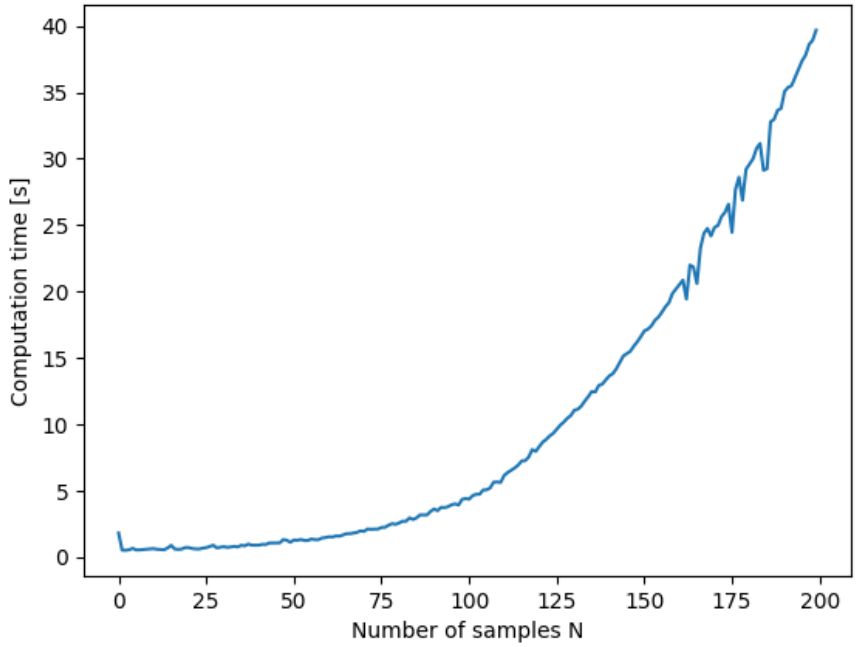

Figure 6: Computation time as a function of the number of samples N for $L = 5$.

One can see that in both cases our solution does not scale linearly (as expected) as

the number of models L or samples N increases the computation increases more rapidly. However; the reader can observe this tendecy to be more important in the case of a varying L parameter than for a changing N. Indeed, while performing experiments, we observed L to be the bottleneck of the system as varying the number of samples N was possible even for values larger than 200 but increasing L was not possible passed 18 as a memory error was raised by MATLAB.

In the conference paper [1] Hannsson and Boyd do not specify in details the computation time performances of their solution, however in the full version of the paper [3] Hannsson indicates that the experiment considered in section 5.1 required 135 $[s]$ to complete while our solution run on a modern laptop completed in 1.13 $[s]$. In [3], the author also specifies than a problem setting with parameters $N = 200$ and $L = 20$ took more than 8 hours to run. We were not able to simulate such a system with our solution due to memory limitations. Hannsson indicates that in order to complete the task it was required to use an iterative method on the number of models L which we decided not to implement as very few elements are presented in [3] regarding this matter.

6 Conclusion

In this report we presented a modern solution using an easily usable tool which is the CVX library for MATLAB to solve a convex optimization control systems problem. We implemented a relatively simple (in terms of the number of lines of code) solution to solve a problem which required 20 years ago a specifically designed Interior-Point resolution method in order to have a software of practical interest which could be run on a workstation with limited computation capacities compared to modern hardware. Our implementation allows the same degree of flexibility and modularity as in [1] where one can change the various parameters of the model. In addition , our performances surpass greatly those of [1]. This shows that modern optimization tools such as CVX are of great interest since they present good performances while being easy to use.

As a future work, it could be interesting to compare the performances of a Python implementation with its corresponding CVX library to the MATLAB solution presented in this report.

References

[1] A. Hansson and S. Boyd. Robust optimal control of linear discrete-time systems using primal-dual interior-point methods. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, volume 1, pages 183–187 vol.1, June 1998.

[2] Michael Grant and Stephen Boyd. Cvx: Matlab software for disciplined convex programming.

[3] A. Hansson. A primal-dual interior-point method for robust optimal control of linear discrete-time systems. *IEEE Transactions on Automatic Control*, 45(9):1639–1655, Sep. 2000.