Numerical Extensions of H_2 Optimal Control: Model Predictive Control and Reinforcement Learning

Xiaozhu Fang [a],

[a] *Department of Mechanical Engineering, University of Michigan, 2350 Hayward, Ann Arbor, MI 48109*

Abstract

In this paper, we discuss the extension of H_2 optimal problem. For linear time invariant system, linear quadratic regulator provide the optimal solution in theory. However, it cannot deal with the strict constraint and uncertainty. To solve such additional demand, two numerical methods are introduced. Model predictive control is powerful for the constraint and Q-learning is used to learning the system online. The experiment applied both methods on the double integrator model and we quantify the performance by H_2 criterion. The result shows the feasibility of two methods and their great potential to improve the conventional controller in the complex environment.

key words: linear time invariant ,H_2 optimal, linear quadratic regulator, model predictive control, Q-learning,

1 Introduction

The theory system begins with the simplest one. People build the state-space model and implement the analysis in the frequency domain on the linear time invariant(LTI) model. To deal with the real complex problem, various extensions appear and even create new fields. Optimal Control is one of them focusing on the optimal control strategy to the demand.

Linear quadratic regulator (LQR) is a typical example of optimal control. Given a system, the objective is to find the control strategy minimizing the cost function. LQR is a special case in H_2 optimal control. H_2 optimal means the minimum norm of the system's transfer function. The superiority of this norm over other refers to Parseval's theorem.

$$\int_0^\infty \parallel F(t) \parallel_F^2 \, dt = \frac{1}{2\pi} \int_{-\infty}^\infty \parallel \hat{F}(iw) \parallel_F^2 \, dw \qquad (1)$$

This equality builds the bridge over the time domain analysis and frequency domain analysis, Therefore, H_2 optimal problem deserves more investigation. The H_2 optimal problem in a complex system is attractive. For the nonlinear system and additional constraints, model predictive control (MPC) is prevailing. According to the prescribed model, the prediction with a finite horizon can be generated so as to provide the optimal control law accordingly. MPC only provides the feasible under the enforced constraints. Its drawback is the huge computation in the prediction, whereas LQR is just a linear control law.

Both LQR and MPC highly relays on the precision of the model. Dynamic programming (DP) provides a sufficient condition for optimally by solving a partial differential equation, named as the Hamilton–Jacobi–Bellman (HJB) equation[1]. Nonetheless, all model are wrong, referring to George Box. The uncertainty of the system is a huge topic of control. The small fluctuation caused by the noise can be compensated by Kalman filter, whereas the large error has to learn by experience. On solution is the adaptive control changes models' parameters in terms of the observation. Furthermore, the model-free learning comes to stage recently due to the rise of Machine Learning. Reinforcement learning(RL) trial and error to find the optimal control law.

In this paper, we will start with LQR in the simple Double Integrator model like the homework. Then we introduce additional demands: strict constraint and unknown dynamics. Thus, MPC and RL are implemented to solve these additional demands. We will see the performance and compare them under the same criterion, H_2 optimal.

The code of this problem is original and available on Github, github.com/XiaozhuFang/598CO-project

⋆ This is a project report in EECS598 Convex Optimization instructed by Professor Peter Seiler in University of Michigan.

Email address: fangxz@umich.edu (Xiaozhu Fang).

2 Backgrounds

2.1 Linear Quadratic Control

LQR is the H_2 optimal control on LTI system.If the view is time domain, the cost function is following

$$J = \frac{1}{2}\int_0^\infty x(t)^T Q x(t) + u(t)^T R u(t) dt \tag{2}$$

where Q and R are designed weighted parameters;x(t) is the state; u(t) is the input. In the lecture and homework, the frequency-shaped LQR is introduced. Basically, we can transform Eqn.2 into frequency domain by Parseval's therom, then we add additional weight in frequency to modify the objective. Its variation of LQR is very practical because the noisy high frequency signal can be neglected.

The solution of LQR is special due to its linearity. The control law is also linear as $u = -R^{-1}B^T P x$, where P is derived from continuous time algebraic Riccati equation(CARE).

$$CAREA^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{3}$$

As for the discrete-time system, there is also discrete time algebraic Riccati equation(DARE). On one hand, to prove CARE, there are numerous ways such as (DP). On the other hand, calculate CARE is problematic for a higher dimension. In the lecture, Linear Matrix Inequalities and semi-definite programming are introduced to calculate CARE in practice.

$$\begin{aligned} &\inf_{Z,W,L} \ trace(W) \\ &\text{subject to :} \begin{bmatrix} W & I_n \\ I_n & Z \end{bmatrix} > 0 \\ &\begin{bmatrix} ZA^T + AZ - L^T B^T - BL & Z & L^T \\ Z & -Q^{-1} & 0 \\ L & 0 & -R^{-1} \end{bmatrix} < 0 \end{aligned} \tag{4}$$

where Z,W are symmetric matrix with $-R^{-1}B^T P = K = LZ^{-1}$. The derivation from DARE to Eqn.(4) require Schur Complement Lemma. The formulation can be implemented in CVX tool in Matlab.

2.2 Model Predictive Control

If we add the constraint in LQR problem, the solution turns to be MPC. MPC has the prediction horizon N. Given the horizon, we can stack all the state and input in a huge vector and calculate the input sequence at one time. Higher N indicates the higher dimension of the stacked model, which requires greater computation. The solution of MPC is actually a Quadratic Programming (QP) problem

$$\begin{aligned} \min_z \quad & \frac{1}{2} z^T H z + g^T z \\ \text{subject to} \quad & Pz \le h \\ & Cz = b \end{aligned} \tag{5}$$

There are numerous methods to solve QP, such as interior point method, projected gradient method, and Alternating Direction Method of Multipliers(ADMM). ADMM is another contribution of Stephen Boyd[2].

2.3 Temporal difference and Q-learning

RL develops from the MDP problem, which provides the basic algorithm value iteration and policy iteration. The novelty of RL starts from the Temporal difference algorithm advanced by Sutton[3], but we need to introduce the value function in MDP at first. The state-dependent value function represents the total expected cost from that state to the end, which is the solution of HJB equation[4].

$$V^\pi(s) = E_{a\sim\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r_t(a_t) | s_0 = s\right\} \tag{6}$$

However, the value function as well as solution of HJB equation require the full information of dynamics, so temporal difference (TD) is the algorithm learning the value function by the observation.

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{7}$$

Eqn.7 is the update equation for TD(0) algorithm where S represents the states, respectively;R is the reward in this interval;α is the learning rate; γ is the discounted value.

TD(0) has the issue of convergence [5], and it can be improved up to TD(1), TD(λ). The difference is that the update equation is extended to the state after the following one.

Q-learning is the off-policy TD control. It separates the value function by actions. $Q(S, A)$ represents the action-value function. The update equation becomes the fallowing.

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t + A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)) \tag{8}$$

where A is the action. Such off-policy methods evaluate or improve a policy different from that used to generate the data[6]. If we just improve the policy used, it becomes the on-policy TD control, Sarsa.

Eqn.8 is the policy improvement part of the reinforcement learning, but we also need to explore more when we generate the new data. If we choose the optimal action (greedy policy) every time from the beginning, the result fast converges to the poor end. $\epsilon - greedy$ policy tells that other non-optimal actions to have the small probability ϵ to be selected. So far we have seen the whole framework of the Q-learning.

Another essential concept in reinforcement learning is the function approximation. The motivation is the huge dimension of the states and actions. The features grouping the states reduce the dimensionality so as to simplify the learning. The option of the feature refers to the methods from machine learning. Either model based basis or neural networks achieve great performance[7].

3 Problem statement

The problem is the optimal control of Double Integrator, which is based on the homework problem.

$$\dot{x} = Ax + Bu \tag{9}$$

where $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ The objective is to minimize the H_2 cost function.$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix}$ and $R = 1$ to emphasise the cost in displacement. 0.01 is added to inverse matrix because we want to use SDP formula (Eqn.4) to solve LQR.

Besides, we make the following changes to incorporate RL in this problem,

(1) Replace A by $A' = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -\delta \end{bmatrix}$, δ is fixed all the experiments, as the unexpected friction term.
(2) Add constraints as the safety guarantees:

$$|x_1| \leq 1, |x_2| \leq 0.3$$

We need to generate new data to interact with the plant system and learn from that. We assume Double Integrator can generate multiple impulses one by one. The magnitude of the impulse, M, is equal to one. We can respond by different control policies and learn with Q-learning.

The input is assumed to have sampling time $Ts = 0.01s$, so the input is fixed in such a tiny interval, which is zero-order hold(ZOH). For simplicity, we calculate the loss in discrete form both in LQR and Q-learning.

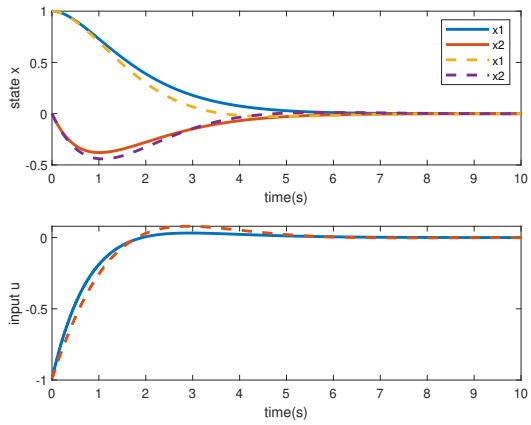

Fig. 1. Impulse response of LQR to the friction-included system. The dash line is the correct model A', while the solid line uses A.

$$J = \sum_{t=0}^{\infty} x(t)^T Q x(t) + u(t)^T R u(t) \tag{10}$$

We can see the system decay to zero after 10 second in Fig.1, so we cut off the cost until to 10 second, which summing total 1000 terms.

4 Experiments

4.1 Friction and constraint

First we want to see the influence of inaccurate model on LQR. Although the system concerns with sampling time, we still use continuous-time result from CARE for LQR controller. Set $\delta = 0.5$ of A' in the real plant, the Fig.1 the $x(1)$ and control input responding to the unit impulse(M=1). Then we compare the cost J in quantity.

$$J_A = 1.5521, J_{A'} = 1.5083 \tag{11}$$

It proofs that the designed policy is not optimal if the model is not accurate. The difference is slight because the gap δ is small. The policy would deviate more when δ rises up. Apart form that, we can see $x(2)$ is out of the constraint, which cannot be enforced except we increase q_{22} in Q.

4.2 MPC

If we want to enforce the constraint $|x_2| \leq 0.3$, MPC is the answer. We use MTP3 tool box in MATLAB to solve this problem. Set horizon $N = 20$, the result of Fig.2 show that MPC controller find the optimal solution under the constraint, where x_2 has the horizon period at the boundary. The cost turns larger due to the

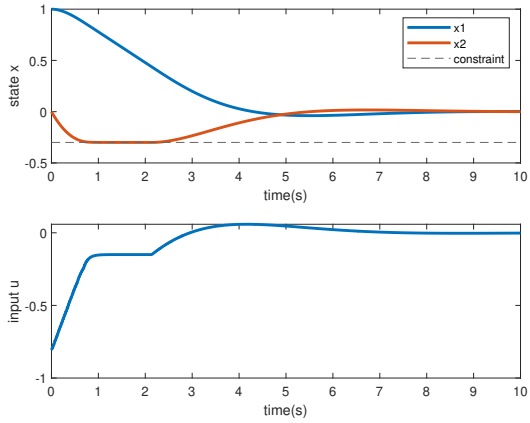

Fig. 2. The response of MPC under the constraint $|x_2| \leq 0.3$

constraint.

$$J_{MPC} = 1.6308 \tag{12}$$

]

Because of that, the input is quite different from Fig.1. The choice of N affect the solution. Arising N, the result turns better but it sacrifices the computation. Thus, the solution is an approximate optimal one,

4.3 Q-learning

To implemment the Q-learning, we need to set aciton-value function $Q(x, u)$, which can be regarded as lots of boxes. Since the states are continuous, we need to discretize them into many divisions. According to Fig.1, we can see the rough range of the states and input, so I set the following 50 boxes for each state and action.

$$x_1 \in \text{linspace(-1,1,50)} \tag{13}$$
$$x_2 \in \text{linspace(-1,1,50)} \tag{14}$$
$$u \in \text{linspace(-1,1,50)} \tag{15}$$

It means that every time the input should only be one of the 50 values in its 50 boxes. Note we have ϵ-greedy exploration at e very time step for every experiment, so the curves show in Fig.3 is experimental data rather than the current optimal solution based on Q. ϵ-greedy exploration will randomly choose non-optimal solutions randomly, but we only choose the neighboring action of the optimal in consideration of the efficiency.

Next, we initialize $Q(x, u)$. A good initialization would save lots of training time. Since our cost is quadratic in x_1, our Q_0 relys on x_1.

$$Q_0(x, u) = 1 + 10|x_1| \tag{16}$$

The reward of $R_t(x, u) = x(t)^T Q x(t) + u(t)^T R u(t)$ as usual. Other parameters used in the learning shows in Table.1.

Attribute	symbol	Value
Learnign rate	α	0.5
Discounted value	γ	0.9
Greedy exploration	ϵ	0.2
Impulse magnitude	M	1

Table 1
Parameters in Q-learning.

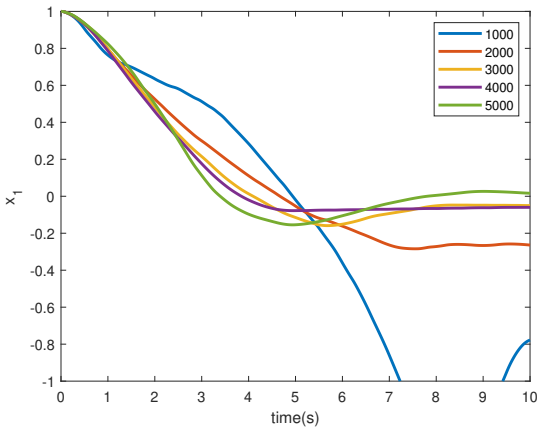

Fig. 3. The generated responses after 1000, 2000, 3000, 4000, 5000 times of practice and learning.

We implement TD(0) based Q-learning, the learning efficiency is too low. TD(λ) is a good improvement, but we use a simple way to improve the learning efficiency for convenience. For one observation, apart from $Q(x_t, u_t)$, $Q(x_t, \cdot)$ also updates its value by Eqn.8 with a different learning rate 0.1. Fig.3 shows the x_1 learning performance after different learning times, from whcih we can observe how the learning develops. Fig.4 shows the $Q(x, a)$ result after 5000 times of learning. We average over all 50 actions to show it straightforward. Fig.5 shows the evolution of the cost function over experiments. The optimal cost after 5000 experiments is $J = 1.974$, which is smaller than the lower bound in Fig.5 because the experimental cost incorporates the exploration. Finally, We can see the result converges to the stable optimal solution (Fig.6).

Compared with LQR, the optimal cost J=1.947 is larger. It is because Q-learning uses many approximation for the system. Note we divide the continuous variables into boxes. If we increase the number of boxes, the result would be improved closer to the optimal one. Q learning seems to be the linear approximation of the optimal solution, but it does not need model prescribed.

4.4 Q-learning with constraints

So far we are also interested in the performance of Q learning if we add the same constraint as MPC. Thus, the reward turns significantly large $R_t(x, u) = 50$ when

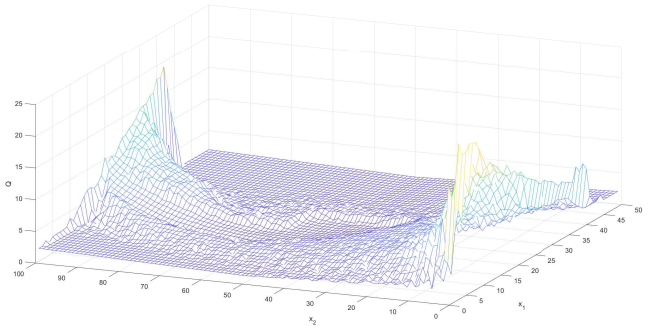

Fig. 4. The average $Q(s,\cdot)$ over actions after 5000 practice and learning. It can be regarded as the Lyapunov function .

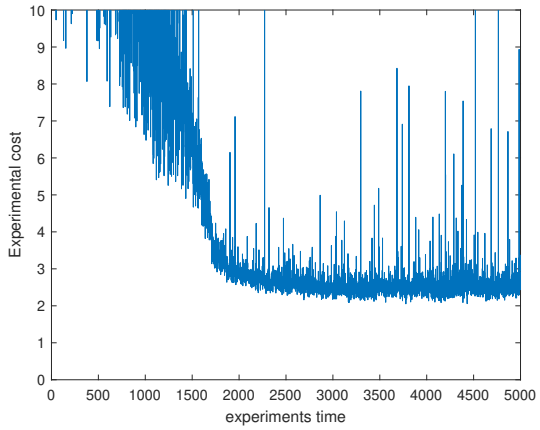

Fig. 5. The loss function over experiments of Q-learning. We can see the spikes caused by exploration

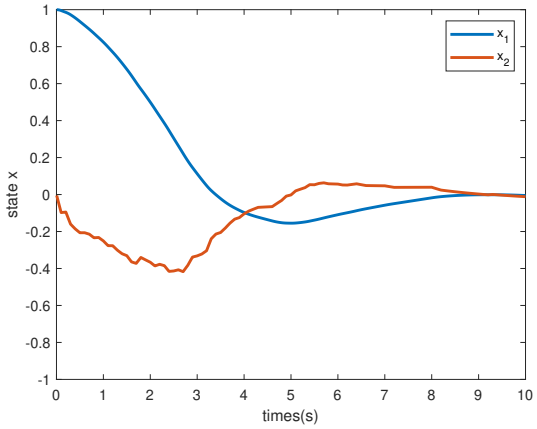

Fig. 6. The Q-learning optimal solution after 5000 experiments. The optimal cost J=1.947

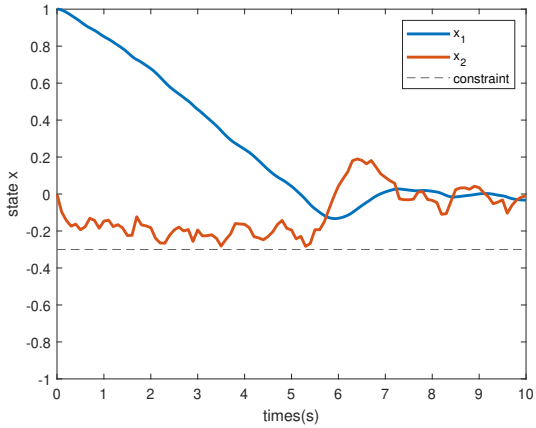

Fig. 7. The Q-learning optimal solution under the constraint $|x_2| < 0.3$ after 5000 experiments. The optimal cost J=3.4407

x_2 is out of the region.

We can see Q-learning find the optimal solution within the constraint in Fig.7

5 Conclusion

In terms of the theories and experiments, we show the limit of LQR controller and two popular extensions of LQR with numerical methods. MPC can deal with strict constraints in terms of its prediction. Beyond such simple linear MPC, more topics such as robustness have been investigated about that[8]. RL is another direction solving the uncertainty, where Q-learning is the basic but powerful part of it. Q-learning requires nothing of system. It learns the knowledge from the practice and gives the command to generate the new one, alternatively. The result of our experiments is very close to the optimal solution after enough training. However, RL is not applicable to any case. In reality, we also need to consider the expense of the experiments. It also explains the current situation: RL is thriving in the gaming, while MPC is dominant in the aerospace.

References

[1] William M McEneaney. *Max-plus methods for nonlinear control and estimation*. Springer Science & Business Media, 2006.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[3] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[4] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[5] Michael Fairbank and Eduardo Alonso. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[8] Munoz de la Pena, Alberto Bemporad, and Carlo Filippi. Robust explicit mpc based on approximate multi-parametric convex programming. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 3, pages 2491–2496. IEEE, 2004.