Linear Dynamic Systems with Unknown Parameters

MS&E 251 Final Project

Hujia YU and Gaelle SMAGGHE

Contents

1 Introduction

Many controllable and non-controllable dynamic systems can be modeled with linear dynamic equations, however in the real world the parameters driving the linear dynamic equation are likely to be unknown. The aim of this project was to define a simple controllable or non controllable linear quadratic dynamic system, with perfect or imperfect state information, and to study different algorithms providing estimates for the optimal control and the unknown parameters.
The following paper explains our models and our findings, expanding on results obtained by implementing the different algorithms on Python.

2 Problem Formulation

2.1 Notations

Throughout the paper, we will define the following:

- n is the dimension of the state space
- r is the dimension of the control space
- x_k is the state at time k, where $x_k \in \mathbb{R}^n$
- u_k is the control at time k, where $u_k \in \mathbb{R}^r$
- z_k is the observed state at time k, where $z_k \in \mathbb{R}^n$. Note that here the observed state has the same dimension as the real state.
- A is a $n \times n$ matrix, with coefficients in $\mathbb{R}$. When initializing A in the algorithms, A will be randomized with coefficients in (0,1).
- B is a $n \times r$ matrix, with coefficients in $\mathbb{R}$. B will also be randomized with coefficients in (0,1) in all the algorithms.
- C is an $n \times n$ matrix with coefficients in $\mathbb{R}$
- θ is a random scalar parameter, normally distributed with mean θ_m and variance Σ_θ
- The matrices A and B will sometimes be defined as functions of θ, in which case we will use the notation $A(\theta)$ and $B(\theta)$
- w_k is a white noise with zero mean and covariance M_k. It is a vector of $\mathbb{R}^n$
- e_k is a white noise with zero mean and covariance N_k. It is a vector of $\mathbb{R}^n$
- We will define $L = riccati(A, B, Q, R)$ as the matrix L such that $L = -\left(B^T K B + R\right)^{-1} B^T K A$, where K is the unique solution of the Riccati equation $K = A^T \left(K - KB(B^T K B + R)^{-1} B^T K\right) A + Q$

2.2 Linear System Equations

2.2.1 LQR with Perfect State Information

We are in this paper considering two different systems.
The first model is the **linear quadratic model with unknown parameters and perfect information**, which is the following:

$$x_{k+1} = Ax_k + Bu_k + w_k$$
$$J = E\left(\frac{x_n^T x_n}{2} + \frac{1}{2}\sum_{k=0}^{n-1}(x_k^T x_k + u_k^T u_k)\right)$$

Where x_k is perfectly observed at the end of each step, but where A and B are unknown time-invariant parameters that we want to estimate.

2.2.2 LQR with Imperfect State Information

The second model is the **linear quadratic model with one unknown parameter and imperfect information**, which is the following:

$$x_{k+1} = A(\theta)x_k + B(\theta)u_k + w_k$$
$$z_k = Cx_k + e_k$$
$$J = E\left(\frac{x_n^T x_n}{2} + \frac{1}{2}\sum_{k=0}^{n-1}(x_k^T x_k + u_k^T u_k)\right)$$

Where z_k is an imperfect observation of x_k at the end of each step and where θ is an unknown parameter driving A and B. The matrix C is known. We want to estimate θ.

In this project we approach and solve both types of LQR parameter learning problems separately and show how each type can be solved accurately and efficiently using different algorithms.

3 Linear Quadratic System with Perfect State Information

3.1 Basic Algorithm

3.1.1 Overview of the algorithm

This algorithm essentially uses the optimal control configuration of the system, and uses closed-loop controls to update state estimate at each time. Figure 1 below illustrates the optimal controller configuration. The current state x_k is fed back as input of the system through the linear feedback gain matrix L_{est}, forming a closed-loop system. The optimal policy is given by $u_k = L_{est}x_k$, where L_{est} is given by computing the K solving algebraic Riccati equation and computing the corresponding L_{est} (as defined in 2.1).

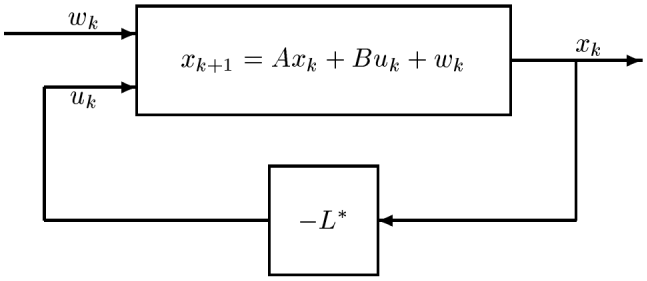

Figure 1: Closed-loop system

With perfect state information, we observe the state at every step, and can therefore compare it to what was expected. Let A_{est} and B_{est} be our estimates for the matrices A and B, then given we have observed x_k and apply control u_k at time k, we expect to observe $A_{est}x_k + B_{est}u_k$ at time $k+1$, and can compare it to x_{k+1}, the real state we observe at time $k+1$.The squared error in estimation at step $k+1$ is $||x_{k+1} - A_{est}x_k - B_{est}u_k||^2$.

The idea of the basic algorithm is to start with estimates A_{est} and B_{est}, to run and observe m steps of the system, then to update A_{est} and B_{est} to be the least square estimates given the $m+1$ vector of observations made. During the m steps, the basic algorithm uses the optimal control law given the estimates A_{est} and B_{est}, which means it applies the control $u_k = L_{est}x_k$ where L_{est} is the optimal control associated with A_{est} and B_{est}, ie $L_{est} = riccati(A_{est}, B_{est}, I_n, I_r)$.

Using the optimal control law at every episode ensures we always pick the control that gives the lowest cost. After m steps, the algorithm updates A_{est} and B_{est}, computes the new optimal law L_{est} and runs a new set of m steps. We define the stopping metrics to be the absolute error of estimated L matrix: $|L_{est} - L|^2$. This process repeats until the this absolute error falls below some predefined threshold.

3.1.2 The Algorithm

- Pick a number of episodes n_e **or** a threshold error e, and a number of steps m
- Initialize A_{est} and B_{est} to random initial estimates
- For k in 1...n_e (episode k) **or** while $|L_{est} - L|^2 > e$
  - $x_0 = 0$
  - $L = riccati(A_{est}, B_{est}, I_n, I_r)$
  - For t in 1...$m+1$ (step t)
    - $u_{t-1} = Lx_{t-1}$
    - $x_t = Ax_{t-1} + Bu_{t-1} + w_t$

    * store x_t and u_{t-1}
  - Construct $\phi = \begin{pmatrix} u_0^T & x_0^T \\ \vdots & \vdots \\ u_m^T & x_m^T \end{pmatrix}$
  - Construct $Y = \begin{pmatrix} x_1^T \\ \vdots \\ x_{m+1}^T \end{pmatrix}$
  - Update estimates to $\begin{pmatrix} B_{est}^T \\ A_{est}^T \end{pmatrix} = (\phi^T \phi)^{-1} \phi^T Y$

3.1.3 Proof of Linear Regression Approach

The goal of this section is to show why the linear squares estimator formula can be used to calculated A_{est} and B_{est} in basic algorithm. The proof is valid for any linear quadratic control system with unknown parameters problem for any control u_k, so we will prove it in the general case.

Le u_k be the control applied at time k, A_{est} and B_{est} our current estimates for A and B.

- The expected state at time $k+1$ is $E[x_{k+1}] = A_{est}x_k + B_{est}u_k$
- The observed state at time $k+1$ is $x_{k+1} = y_k^T$
- The squared error in estimation is $||x_{k+1} - A_{est}x_k - B_{est}u_k||^2$
- m observations are made, so the total squared error is $e(A_{est}, B_{est}) = \sum_{k=0}^{m} ||x_{k+1} - A_{est}x_k - B_{est}u_k||^2$
- We look for the A_{est} and B_{est} that would have minimized this error $e(A, B)$ by setting the derivatives to zero
  - $\frac{\partial e}{\partial A} = -2\sum_{k=0}^{m} x_k \left(x_{k+1}^T - x_k^T A^T - u_k^T B^T\right) = 0$
  - $\frac{\partial e}{\partial B} = -2\sum_{k=0}^{m} u_k \left(x_{k+1}^T - x_k^T A^T - u_k^T B^T\right) = 0$
  - This yields: $\sum_{k=0}^{m} x_k x_{k+1}^T = \sum_{k=0}^{m} x_k x_k^T A^T + \sum_{k=0}^{m} x_k u_k^T B^T$
  - And: $\sum_{k=0}^{m} u_k x_{k+1}^T = \sum_{k=0}^{m} u_k x_k^T A^T + \sum_{k=0}^{m} u_k u_k^T B^T$
  - Which is exactly $\phi^T \phi \begin{pmatrix} B^T \\ A^T \end{pmatrix} = \phi^T Y \Leftrightarrow \begin{pmatrix} B^T \\ A^T \end{pmatrix} = \left(\phi^T \phi\right)^{-1} \phi^T Y$

3.1.4 Results and Analysis

Figure 1 shows the error (in norm) in approximating A, B and the optimal policy matrix L as a function of the number of episodes for a number of steps equal to $m = 50$ and for 3 different runs of the algorithm. The dimensions used were $r = 3$ and $n = 5$.

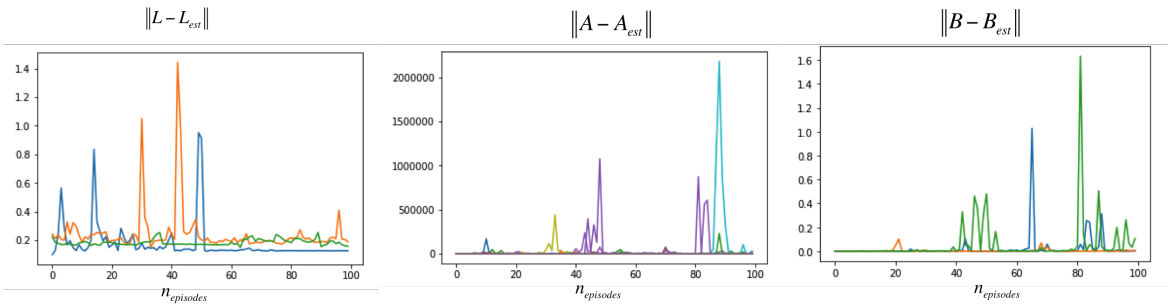

Figure 2: Convergence of the algorithm for n=5, r=3, m=50

The results show that convergence of the error in approximating all 3 matrices (A,B and L) is unstable and that the error does not always converge to zero. The spikes in the graphs indicate that during some steps, the estimates for A, B and L are very far away from the real matrices, therefore the control applied is very different to the optimal control and we can expect the cost incurred to be very high.

Figure 2 shows the error (in norm) in approximating L as a function of the number of episodes for a number of steps equal to $m = 200$ and $m = 400$. The dimensions used were $r = 3$ and $n = 5$.

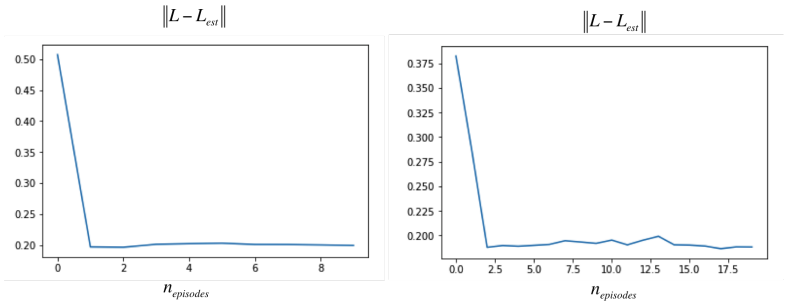

Figure 3: Convergence of the algorithm for m=200 and m=400

These results show that increasing the number of steps increases the speed and the accuracy of the convergence. Since increasing the number of steps corresponds to increasing the number of observations the maximum likelihood estimate is taken over, it is understandable that the estimate is more accurate. However, the graphs show that convergence is still sometimes unstable, and that the error very slowly converges to zero.

3.1.5 Why the closed-loop control learning algorithm doesn't work in practice

First, we find that during implementation, this algorithm is very unstable in performance. Sometimes the system crashes and we have error saying the ϕ matrix is a singular matrix and cannot be inverted (in the steps of calculating $A_{est}B_{est}$ using ϕ), sometimes the system runs but does not converge after 10000 steps, with occasional spikes in costs, indicating that A_{est} or B_{est} has exploded somehow, but that spike will decrease almost right away and the control u_k will go back to really small values. For the very few times that the algorithm did work and converge at the end, convergence is still unstable and happens very slowly. We think this phenomenon has to do with the random numbers A_{est} and B_{est} have been initialized with and the numerical stability during the process of inverting large matrices.

Out of the times that this algorithm worked, the convergence was slow and unstable. Regarding this phenomenon, we think it is because that the learning phase in this basic algorithm is taking place by greedily plugging in the optimal control calculated by L_{est} at each time step. This way, we don't actively search for control vectors that help the estimators learn the environment better, this passive way of learning is what causes the convergence to be slow. Thus, to improve the algorithm performance, a way to keep the system excited, that is, a way to actively search in the control space the set of control vectors that could help the system learn its environment better is needed. The time taken by the algorithm is polynomial in the dimension n of the state-space and in the dimension r of the control when the ratio n/r is a constant.

3.2 PAC: Adaptive Control Learning Algorithm [1]

In this section, we introduce PAC adaptive control learning algorithm that directly tackles the convergence problem we faced above. This algorithm actively explores its environment by performing actions that look suboptimal in terms of rewards but help build a better environment for learning. Moreover, it actively searches in the control space the set of control vectors that renders linearly independent observations so that they span the entire observation space.

This algorithm makes no assumption on noise distribution or the system parameter distributions, and therefore can be applied to all types of systems. PAC algorithm uses open-loop control (does not rely on feedbacks) at each time step to update estimated next state. Therefore this algorithm works with any types of control.

3.2.1 Overview of the algorithm

Linear regression estimator is still applied in this algorithm, meaning we need to collect a matrix of ϕ that consists of m observations of u_k and x_k and a column vector Y_m that also contains m state observations x_{k+1}, and we once again use a linear square estimator to compute a closed form solution for the optimal A_{est} and B_{est} corresponding to collected observations.

The key difference between this algorithm and the basic algorithm we described in the first section is that here the learning algorithm will actively explore the state-space to quickly obtain a good approximation of the unknown parameters. The basic algorithm, on the contrary, always chooses the control that looks best from an exploitation point of view and only learns passively.

More specifically, the learning takes place in a series of trials, within each trial, the agent performs multiple controls in a specific order to collect one transition instance u_k and x_k. The total number of trials is m, which is the number of transition pairs u_k and x_k that we have in ϕ. For each trial, the agent selects and applies a series of control u_k and the system makes a transition to the next state according to the state equation and noise w_k. Only the last transition pair u_k and x_k at the end of each trial is collected (thus m trials corresponds to ϕ of length m).

3.2.2 The Algorithm

The pseudo code for each exploration/epoch of learning is shown in Figure 4 below. Basically for each exploration, there are m trials, where each trial gives one data point in Y_m and ϕ_m matrix that we later use to update A_{est} and B_{est}. Each episode consists of m successive trials of length l that cover different control choices in control space so as to produce m linearly independent observations.

More specifically, $\bar{u}_i$ is a vector that combines the control $u_0, u_1, ..., u_{l-1}$ applied during the l steps of trial i of an episode, and one data instance of u_{l-1}, x_{l-1} are collected at the end of each trial. So one episode consists of a series of trials that we use to apply controls, but only the observation made on the last transition of that trial is collected.

$$\bar{u}_i = \begin{pmatrix} u_{l-1}^T \\ \vdots \\ u_0^T \end{pmatrix}, \psi_i = \begin{pmatrix} u_{l-1} \\ x_{l-1} \end{pmatrix}$$

Some of the parameters we chose in implementation of the open-loop learning algorithm vary slightly from the paper in order to adapt the algorithm to control vectors of dimension $r > 1$. Our algorithm overall strictly follows method introduced in the paper and we present it in details below:

- Pick a number of episodes n_e and a length for the trials l
- Initialize A_{est} and B_{est} to random initial estimates
- For k in 1...rl (trial k)
  - $x_0 = 0$
  - Pick control law $\hat{u}^k = \begin{pmatrix} u_{l-1}^T \\ \vdots \\ u_0^T \end{pmatrix} = e_k$ where e_k is the k^{th} vector of the canonical base of $\mathbb{R}^{l \times r}$
  - For t in 1...l (step t)

{exploration}
$m \leftarrow 0$
for $i = 1$ to M **do** {episode i}
  for $j = 1$ to p **do** {trial j}
    perform a $(\ell+1)$-step trial using the open-loop control law $\bar{u}_j$
    $m \leftarrow m+1, \quad \varphi_m \leftarrow \begin{bmatrix} u_\ell \\ x_\ell \end{bmatrix}, \quad y_m \leftarrow x_{\ell+1}^T$
  end for
end for

{compute exploitation policy}
1: compute $\hat{\theta}_m = [\hat{B} \quad \hat{A}]^T$ using the least-square algorithm (8) with observation-output pairs (φ_i, y_i) for $i = 1, \ldots, m$.
2: compute $\tilde{\pi}$ using (3) to (5) in Section 2.1 with A and B replaced by their estimates $\hat{A}$ and $\hat{B}$.

Algorithm 1: Learning Algorithm

  - $u_{t-1} = (\hat{u}_{l-t+1}^k)^T$ (use the controls associated with the control law $\hat{u}^k$)
  - $x_t = Ax_{t-1} + Bu_{t-1} + w_k$
- store x_l^k, x_{l-1}^k and u_{l-1}^k, the last control and last two states for trial k

- Construct $\phi = \begin{pmatrix} (u_{l-1}^1)^T & (x_{l-1}^1)^T \\ \vdots & \vdots \\ (u_{l-1}^{rl})^T & (x_{l-1}^{rl})^T \end{pmatrix}$
- Construct $Y = \begin{pmatrix} (x_l^1)^T \\ \vdots \\ (x_l^{rl})^T \end{pmatrix}$
- Repeat n_e times, appending every new rl trials in the same way to the same vectors ϕ and Y
- Update estimates to $\begin{pmatrix} B_{est}^T \\ A_{est}^T \end{pmatrix} = (\phi^T\phi)^{-1}\phi^T Y$
- Solve $L_{est} = riccati(A_{est}, B_{est}, Q, R)$

3.2.3 Results and Analysis

To show the importance of the number of episodes ran, we plot the absolute error of estimated L_{est} with respect to the number of episodes ran (for the same length of trials chosen). We run this open-loop learning algorithm on state x with dimension $n = 6$ and control with dimension $r = 1$ for 300 episodes, and on $n = 10, r = 3$, the changes in absolute error is then plotted in Figure 4

below. To have a better sense of how well the algorithm is doing, we also plotted absolute error of estimated A_{est} and B_{est}, shown in Figure 5 below.

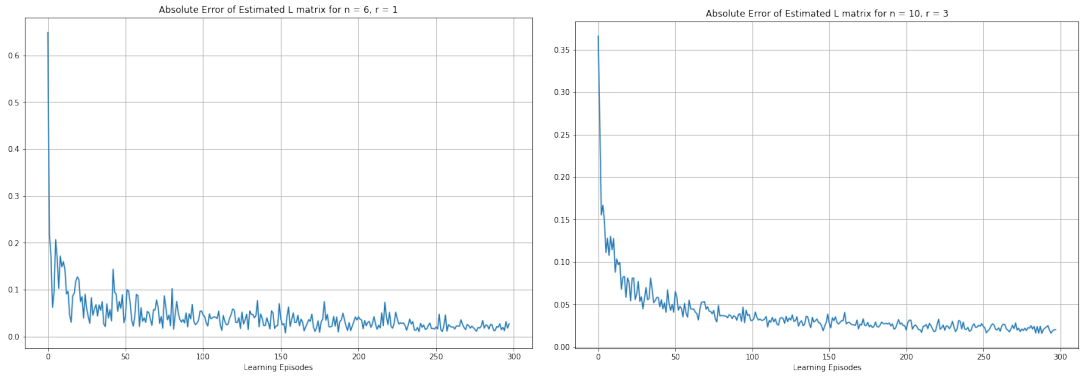

Figure 4: Absolute Error of estimated L matrix for PAC learning algorithm for (left) x of size 6 and control u of size 1 and (right) x of size 10 and control u of size 3.

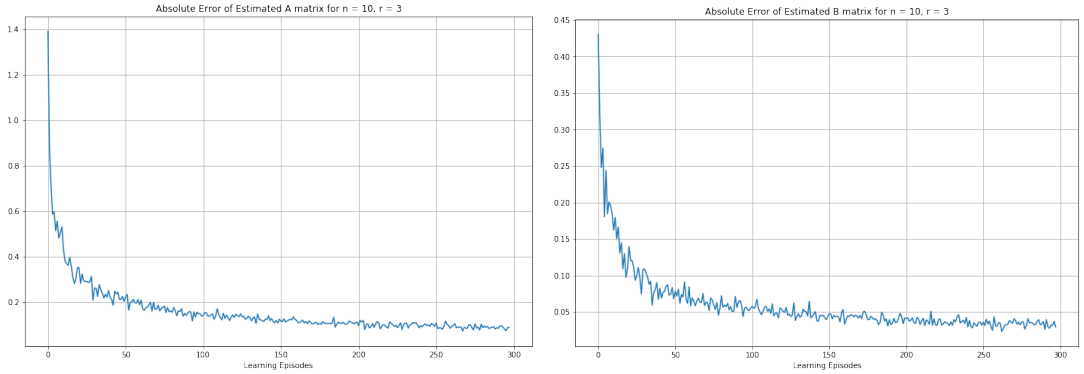

Figure 5: Absolute Error of estimated A and B matrix for PAC learning algorithm for x of size 10 and control u of size 3.

We see clearly from the graphs that the algorithm converged quickly for both dimensions of state and input and for both A and B. There was slightly more variance in absolute error for case of $n = 6, r = 1$ at the beginning of the learning episodes, but it quickly died down to converge to close to 0. The resulting plots are smooth and stable.

Therefore, we conclude that the PAC open-loop learning algorithm works well in learning unknown parameters in LQR system. It achieves higher performance than our basic close-loop learning algorithm by actively searching for control vectors to produce observations that are linearly independent and thus span the entire observation as much as possible. This way the agent gathers better understanding of the environment gives better estimates of parameters. Since PAC learning algorithm is open-loop, it essentially works with any types of control.

4 Linear Quadratic System with Imperfect State Information

We are in this section considering a linear quadratic model with imperfect information. In this case, the Kalman filter can be implemented if the initial guesses and the noises are all assumed to have Gaussian distributions. We will in the rest of this section assume the initial guesses and the noises follow such Gaussian distributions, and show how the Kalman filter can be used to estimate random parameters. Sources [2] to [6] were used for this section.

4.1 Kalman Filter Without Control

4.1.1 Method

Let us first consider a simple model without control, driven by the following equations, and by the unknown parameter θ:

$$\begin{aligned} x_{k+1} &= f(x_k, \theta) + w_k \\ z_k &= Cx_k + e_k \\ J &= E\left(\tfrac{1}{2}\sum_{k=0}^{n}(x_k^T x_k)\right) \end{aligned}$$

Where z_k is an imperfect observation of x_k at the end of each step and where θ is an unknown parameter driving A and B. We will consider that the matrix C is known, that the parameter θ is normally distributed with mean θ_m and covariance Σ_θ and that x_0 is normally distributed with mean x_m and covariance Σ_m.
In order to estimate the unknown parameter θ, we expand the state space to contain the parameter we want to estimate. We will in this case consider the parameter θ as constant over time, and create the new state y_k, defined as follows:

$$y_k = \begin{pmatrix} x_k \\ \theta \end{pmatrix}$$

Our new state y_k and observation z_k are driven by the following dynamic equations:

$$\begin{aligned} y_{k+1} = \begin{pmatrix} x_{k+1} \\ \theta \end{pmatrix} = \begin{pmatrix} f(x_k,\theta) \\ \theta \end{pmatrix} + \begin{pmatrix} w_k \\ 0 \end{pmatrix} = g(x_k,\theta) + \begin{pmatrix} w_k \\ 0 \end{pmatrix} = \hat{g}(y_k) + \begin{pmatrix} w_k \\ 0 \end{pmatrix} \\ z_k = Cx_k + e_k = \begin{pmatrix} C & 0 \end{pmatrix}\begin{pmatrix} x_k \\ \theta \end{pmatrix} + e_k = \begin{pmatrix} C & 0 \end{pmatrix} y_k + e_k \end{aligned}$$

Where $\begin{pmatrix} w_k \\ 0 \end{pmatrix}$ is a normally distributed noise, with covariance M_k and such that no noise is applied to θ because we assume it is constant over time. e_k is normally distributed with covariance N_k. The Kalman filter works as follows:

- Our initial guess for y_0 is $\begin{pmatrix} y_m \\ \theta_m \end{pmatrix}$

- Our initial guess for the covariance is Σ_0 is $\begin{pmatrix} \Sigma_m & 0 \\ 0 & \Sigma_\theta \end{pmatrix}$
- With every new observation z_k, the estimate state and covariance $y_{k|k-1}$ and $\Sigma_{k|k-1}$ are updated as follows (where the initial guesses are y_0 and Σ_0)
  - $y_{k|k} = y_{k|k-1} + \Sigma_{k|k-1} \begin{pmatrix} C \\ 0 \end{pmatrix} \left(\begin{pmatrix} C & 0 \end{pmatrix} \Sigma_{k|k-1} \begin{pmatrix} C \\ 0 \end{pmatrix} + N_k \right)^{-1} \left(z_k - \begin{pmatrix} C & 0 \end{pmatrix} y_{k|k-1} \right)$
  - $\Sigma_{k|k} = y_{k|k-1} + \Sigma_{k|k-1} \begin{pmatrix} C \\ 0 \end{pmatrix} \left(\begin{pmatrix} C & 0 \end{pmatrix} \Sigma_{k|k-1} \begin{pmatrix} C \\ 0 \end{pmatrix} + N_k \right)^{-1} \begin{pmatrix} C & 0 \end{pmatrix} \Sigma_{k|k-1}$
- Then the estimates are moved one step forward using the following equations:
  - $y_{k+1|k} = \begin{pmatrix} f(x_{k|k}, \theta_{k|k}) \\ \theta_{k|k} \end{pmatrix} = g(x_{k|k}, \theta_{k|k}) = \hat{g}(y_{k|k})$
  - $\Sigma_{k+1|k} = \begin{pmatrix} \frac{\partial f}{\partial x}(x_{k|k}, \theta_{k|k}) & \frac{\partial f}{\partial \theta}(x_{k|k}, \theta_{k|k}) \\ 0 & I \end{pmatrix} \Sigma_{k|k} \begin{pmatrix} \frac{\partial f}{\partial x}(x_{k|k}, \theta_{k|k}) & \frac{\partial f}{\partial \theta}(x_{k|k}, \theta_{k|k}) \\ 0 & I \end{pmatrix}^T + M_k$

Using this method, we update at every step our belief regarding the unknown parameter θ, and after a certain number of steps have a reasonable estimate of θ, which means we have a reasonable estimate of what out dynamic equation is.

4.1.2 Results

The section below shows results of the algorithm implemented for a very simple system where n=5, $C = I_n$, θ is a scalar parameter, $A(\theta) = A + \theta I_n$ (A is a known matrix, only the scalar parameter θ is unknown). With such approximations, we can use in the equations above:

- $f(x_k, \theta) = (A + \theta I_n) x_k$
- $\frac{\partial f}{\partial x}(x_{k|k}, \theta_{k|k}) = A + \theta_{k|k} I_n$
- $\frac{\partial f}{\partial \theta}(x_{k|k}, \theta_{k|k}) = x_{k|k}$

Figure 5 shows the evolution of the absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of the number of observations for different distributions of θ and for 5 trials. Here there is no control, and we see that the algorithm converges towards the correct parameter in less than 100 steps. We also observe that convergence is slower, and more unstable, if the variance of the parameter to estimate is important (for $\theta \sim \mathcal{N}(0, 10)$, 1 of the 5 trials did not converge).

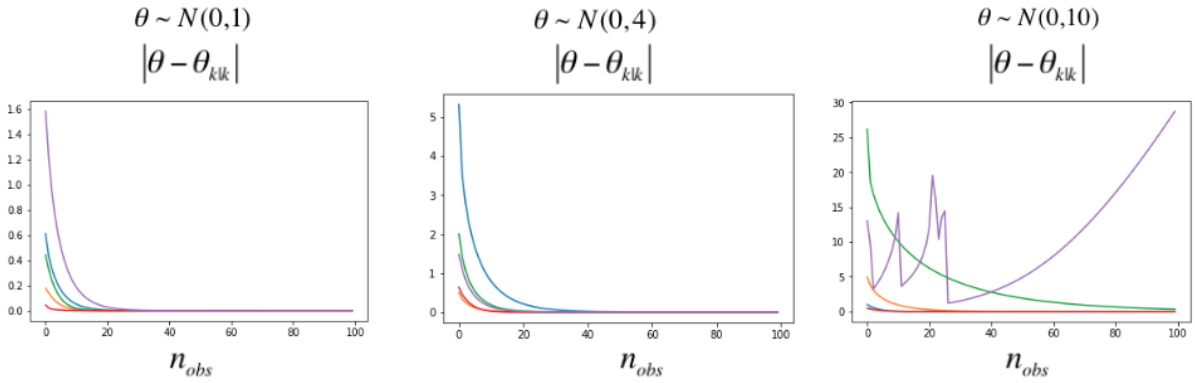

Figure 6: Absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of n_{obs} for different distributions of θ and for 5 trials

4.2 Kalman Filter With Control

In this section we will add a control to the problem presented in the previous paragraph. The new problem becomes:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, \theta) + w_k \\ z_k &= Cx_k + e_k \\ J &= E\left(\frac{x_n^T x_n}{2} + \frac{1}{2}\sum_{k=0}^{n-1}(x_k^T x_k + u_k^T u_k)\right) \end{aligned}$$

We expand the state space in the same way, creating the vector y_k such that:

$$y_{k+1} = \begin{pmatrix} x_{k+1} \\ \theta \end{pmatrix} = \begin{pmatrix} f(x_k, u_k, \theta) \\ \theta \end{pmatrix} + \begin{pmatrix} w_k \\ 0 \end{pmatrix} = g(x_k, u_k, \theta) + \begin{pmatrix} w_k \\ 0 \end{pmatrix} = \hat{g}(y_k, u_k) + \begin{pmatrix} w_k \\ 0 \end{pmatrix}$$

The Kalman filter works in the same way, where the initial estimates for y_0 and Σ_0 are defined in the same way, where the updates when observing new information $y_{k|k}$ and $\Sigma_{k|k}$ follow the same formulas. The formulas to move one step forward, however, are different, and become:

- $y_{k+1|k} = \begin{pmatrix} f(x_{k|k}, u_k, \theta_{k|k}) \\ \theta_{k|k} \end{pmatrix} = g(x_{k|k}, u_k, \theta_{k|k}) = \hat{g}(y_{k|k}, u_k)$
- $\Sigma_{k+1|k} = \begin{pmatrix} \frac{\partial f}{\partial x}(x_{k|k}, u_k, \theta_{k|k}) & \frac{\partial f}{\partial \theta}(x_{k|k}, u_k, \theta_{k|k}) \\ 0 & I \end{pmatrix} \Sigma_{k|k} \begin{pmatrix} \frac{\partial f}{\partial x}(x_{k|k}, u_k, \theta_{k|k}) & \frac{\partial f}{\partial \theta}(x_{k|k}, u_k, \theta_{k|k}) \\ 0 & I \end{pmatrix}^T + M_k$

The decision to make here is which set of controls u_k to use during the learning period. For the purpose of this project, we studied to types of control. The first control law we could choose would be to implement at every step the optimal control given our current estimates of A and B, which would mean that at step k we implement $u_k = L_{k|k}x_k$ where $L_{k|k} = riccati(A(\theta_{k|k}), B(\theta_{k|k}), I_n, I_r)$. The second option is to choose the same method as the PAC learning algorithm and to use an open-loop control law covering the canonical base of $\mathbb{R}^r$. Both of those methods are implemented in the following sections.

The two sections below shows results of the algorithm implemented for a very simple system where n=5, $C = I_n$, θ is a scalar parameter, $A(\theta) = A + \theta I_n$, $B(\theta) = B + \theta I_{nr}$ (A and B are known matrices, only the scalar parameter θ is unknown). With such approximations, we can use in the equations above:

- $f(x_k, u_k, \theta) = (A + \theta I_n)x_k + (B + \theta I_{nr})u_k$
- $\frac{\partial f}{\partial x}(x_{k|k}, u_k, \theta_{k|k}) = A + \theta_{k|k} I_n$
- $\frac{\partial f}{\partial \theta}(x_{k|k}, u_k, \theta_{k|k}) = x_{k|k} + I_{nr} u_k$ (note that this is the only major change in the formulas)

4.2.1 Kalman Filter With Optimal Control

Figure 6 shows the evolution of the absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of the number of observations for different distributions of θ and for 5 trials. In this case the optimal control was computed at every step by solving the Riccati equation using the estimate for θ at each step. We see that the algorithm performs well, and that after less than 40 observations we have an accurate estimate of θ. The plots also show that as the variance Σ_θ of our initial parameter increases, the number of observations required to learn accurately increases, which is intuitive. The last plot, where $\theta \sim \mathcal{N}(0, 10)$ shows that if the variance of the parameter to estimate is very important, convergence takes longer and is not guaranteed (here 1 out of the 5 trials was unstable)

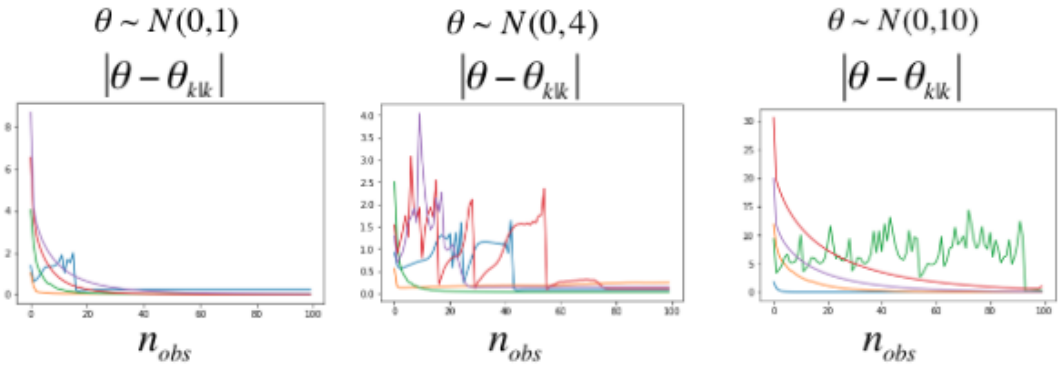

Figure 7: Absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of n_{obs} for different distributions of θ and for 5 trials

4.2.2 Kalman Filter With Open-Loop Canonical Control

Figure 7 shows the evolution of the absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of the number of observations for different distributions of θ and for 5 trials. Here the control law is an open loop law, so the control was chosen beforehand and spans the canonical base.

We see that the algorithm performs better than that with optimal control from previous section, and that after less than 40 observations we have an accurate estimate of θ. Similar with the

optimal control case that as the variance Σ_θ of our initial parameter increases, learning becomes more unstable. However in this case, convergence on open-loop control is more stable than that with optimal control even with higher variance in initialization of θ. In the case with really large variance($N\ (0, 10)$) however, some cases of open-loop control still converges sub-optimally but the chance of that is significantly lower that the case with close-loop optimal control. These results show that (1) open-loop control remains a more promising method to choose control vectors to gather observations in the LQR system with imperfect state information. (2) convergence rate of the model depends on variance of the parameter, convergence takes longer and is not guaranteed for problems with large variance in its initializations (here 1 out of the 5 trials was unstable).

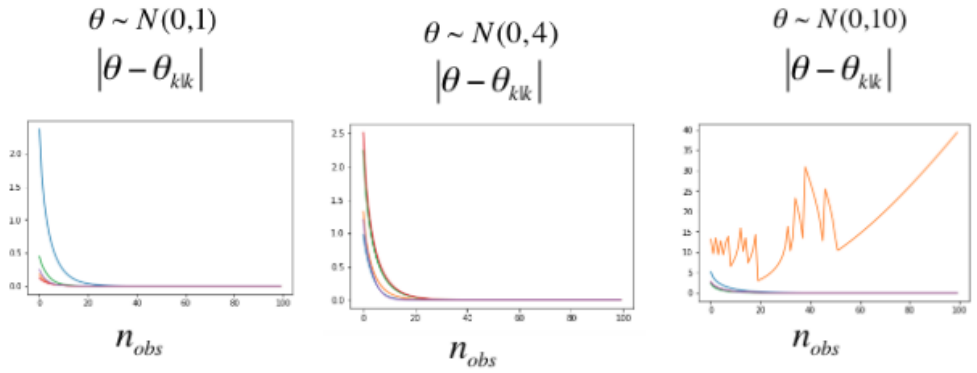

Figure 8: Absolute error in estimation $|\theta - \theta_{k|k}|$ as a function of n_{obs} for different distributions of θ and for 5 trials

5 Conclusion

In this writeup we present our solutions to two types of linear quadratic regulator (LQR) parameter estimation problems. We first focused on LQR with perfect state information system. We use linear squares estimator to compute estimated A_{est} and B_{est} using data on observed control and states stored as $\phi = \begin{pmatrix} u_0^T & x_0^T \\ \vdots & \vdots \\ u_m^T & x_m^T \end{pmatrix}$.

In order to calculate estimates based on linear square estimator equations, we need to collect data on system updates at each time step. We then make decisions to how we want to apply control to update the state estimates. We presented solutions from both updating the system using close-loop optimal control and open-loop control canonical base separately. The results show that even though the basic greedy strategy in choosing optimal control to update state estimates could converge to optimal estimates of A and B, the majority of the times the convergence is very slow and unstable. On the other hand, applying open-loop canonical control base to the system worked exceptionally well in estimating unknown parameters, and the convergence was fast and smooth.

In the second part of the project we tackled a second type of LQR parameter estimation problem with imperfect state information, where we have an observation error $z_k = Cx_k + e_k$. We solve this type of LQR problem using Kalman Filter update equations, which works under Gaussian distributions assumption (for noises and initial guesses). In updating the system state information during the learning steps, we considered different types of controls: (1) no control; (2) optimal control; (3) open-loop canonical base control. Our results show that the Kalman Filter update methods with all three types of control converge to the real value of the unknown parameter in the end. However, the convergence performance from applying open-loop canonical base control turns out to be more stable.

Our conclusion is that for LQR parameter estimation problem with perfect or imperfect state information systems, picking the right set of control vectors to the update system equation in gathering data points about u and x is the key to success for the agent to learn the environment. Actively picking controls that look suboptimal but span the horizon of possible controls performs better than greedily applying optimal control. Moreover, picking canonical bases of control vectors helps system generate observations that are linearly independent and thus provide better estimates of parameters in both perfect and imperfect state systems.

References

[1] *PAC Adaptive Control of Linear Systems*, Claude-Nicolas Fiechter, Department of Computer Science, University of Pittsburgh

[2] MS&E 251 Lecture Notes (Lectures 9,11,12 and 13)

[3] *Parameter Estimation Method Using A Kalman Filter* - Emmanuel D. Blanchard, Adrian Sandu, Corina Sandu - 2007 - University of Wollongong

[4] *Extended Kalman Filter for Estimation of Parameters in Nonlinear State*, Space Models of Biochemical Networks Xiaodian Sun, Li Jin, Momiao Xiong, 2008, PLoS ONE

[5] Lecture Notes from Polytechnique Montreal `http://www.professeurs.polymtl.ca/`

[6] Lecture Notes from Richard M. Murray - CalTech - *Optimisation Based Control* `http://www.cds.caltech.edu/`