Learning to Control First Order Linear Systems with Discrete Time Reinforcement Learning

Eric Nelson
Department of Computer Science and Engineering
Texas A&M University
College Station, TX 77843, USA
ejn8411@tamu.edu

Thomas Ioerger
Department of Computer Science and Engineering
Texas A&M University
College Station, TX 77843, USA
ioerger@cs.tamu.edu

***Abstract*— Reinforcement learning (RL) is a powerful method for learning policies in environments with delayed feedback. It is typically used to learn a control policy on systems with an unknown model. Ideally, it would be desirable to apply RL to learning controllers for first-order linear systems (FOLS), which are used to model many processes in Cyber Physical Systems. However, a challenge in using RL techniques in FOLS is dealing with the mismatch between the continuous-time modeling in the linear-systems framework and the discrete-time perspective of classical RL. In this paper, we show that the optimal continuous-time value function can be approximated as a linear combination over a set of quadratic basis functions, the coefficients of which can be learned in a model-free way by methods such as Q-learning. In addition, we show that the performance of the learned controller converges to the performance of the optimal continuous-time controller as the step-size approaches zero.**

I. INTRODUCTION

The use of reinforcement learning is becoming an increasingly popular choice for learning to control arbitrary systems by minimizing the error of a given objective function. This is because near optimal control policies can be obtained without having to know the underlying system model. If the model of the system is known, the optimal control can sometimes be derived analytically or through the use of dynamic programming and other methods. However, the analytical solutions, even when they can be obtained, are often complicated. This is even more true in the case of higher order dynamical systems. Therefore, this work focuses only on first-order (FO) linear systems. In addition, it is possible for the dynamics of linear systems to vary with time. For example, an airplane's weight changing as the fuel it is carrying is burned. Solutions to these types of systems do exist however they are not the focus of this paper. We only consider the control of first-order linear time-invariant (FO LTI) systems.

The analytical control solutions of FO LTI systems are continuous-time functions that define an optimal trajectory through state-space. However, in many CPSs, control inputs cannot be performed in real-time. For instance, the controls performed by a digital control system are limited by the clock cycle time of the underlying hardware. We assume in this paper that for these discrete time systems a control is selected and is held for a minimum of Δt before another control can be selected where the time between successive control inputs is denoted as Δt.

Reinforcement learning (RL) is a method for learning a policy in a particular environment by exploring the state and action space and receiving rewards after transitioning to a new state. Some RL algorithms, such as value iteration, require knowing a-priori the reward functions and state-transition functions. To remedy this, Q-learning was proposed to be able to learn without explicit knowledge of the reward and state-transition functions [1]. In Q-learning, we attempt to learn a function of both a state and an action instead of attempting to learn a function that gives a value only based on state. The Q-learning algorithm was originally proposed to work on problems with discrete state spaces. Eventually, methods to adapt Q-learning to continuous state spaces were developed in which the state space can be represented by a set of weighted basis functions [2].

In this paper, we show that an optimal controller for a linear system can be learned through the use of reinforcement learning. In addition, we provide insight into how the selection of a Δt affects the policies learned by the reinforcement learner. It seems intuitive that a smaller Δt will lead to better control policies and reducing the cycle time of the underlying hardware is possible through engineering methods. However, this can significantly increase the cost of the design. The selection of a larger Δt will allow for costs to be decreased by allowing slower (cheaper) hardware to be used. Further, we show how the properties of linear systems can be exploited in order to decompose the description of the system into a transient response which affects how the system evolves shortly after a new control input is selected and a long-term response which specifies where the system will come to rest. In order to do so, we have to make a few assumptions. First and as discussed above, we are only concerned with LTI systems. In addition, we assume once a control input has been selected we must hold it for the entire Δt. We show the reinforcement learner can learn a weighted basis function representation of the value function that approximates it's optimal continuous time counterpart. Finally, we show empirically that control error shrinks with Δt and approaches optimal continuous time control in the limit (as $\Delta t \to 0$).

II. CONTROL OF LINEAR SYSTEMS

A. State-Space Representation

Modern control theory introduces the idea of the state of a system. In this representation, a set of state variables are chosen and the system is described by how these variables evolve over time. Any FO LTI system with n state variables, r input variables, and m output variable can be written in terms of change in state by the following simultaneous differential equations:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

where x is the state, y is the output and u is the control input. $\mathbf{A}$ is an $n \times n$ matrix that describes the system dynamics. $\mathbf{B}$ is an $n \times r$ matrix that describes how the inputs affect the change in state. $\mathbf{C}$ is an $m \times m$ matrix and $\mathbf{D}$ is an $m \times r$ matrix.

We can solve the differential equation for x by employing the Laplace transform.

$$\mathbf{X}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{x}(0) + (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s) \quad (1)$$

Taking the inverse Laplace transform we get:

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\, d\tau \quad (2)$$

Therefore, we can see that the internal behavior (state space trajectory) is not, in general, linear and instead the system evolves exponentially as a function of time. These trajectories have a transient phase of rapid response before slowing to a new equilibrium point which we call the long-term or asymptotic response. If we assume that $\mathbf{A}$ is negative definite then as a control is applied, the exponential terms will approach zero as time passes. Eventually, the system will converge to a new equilibrium point as the exponential terms approach zero.

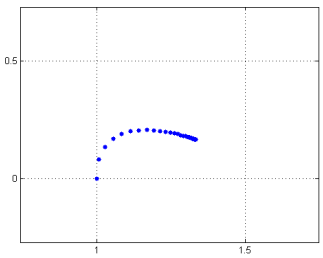


Fig. 1: State space response of the example system to unit step input. The state changes rapidly during the initial period (transient) after a new control input is applied and gradually converges to a new equilibrium point (long-term).

B. Example

Consider the following first-order linear system:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -1 & 2 & 0 \\ -1 & -4 & 1 \\ 0 & 0 & -1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{u}(t)$$

The solution can be derived as follows:

$$(s\mathbf{I} - \mathbf{A}) = \begin{bmatrix} s+1 & -2 & 0 \\ 1 & s+4 & -1 \\ 0 & 0 & s+1 \end{bmatrix}$$

$$(s\mathbf{I} - \mathbf{A})^{-1} =$$

$$\begin{bmatrix} \frac{2}{s+2} - \frac{1}{s+3} & \frac{2}{s+2} - \frac{2}{s+3} & -\frac{2}{s+2} + \frac{1}{s+3} + \frac{1}{s+1} \\ \frac{1}{s+3} - \frac{1}{s+2} & \frac{2}{s+3} - \frac{1}{s+2} & \frac{1}{s+2} - \frac{1}{s+3} \\ 0 & 0 & \frac{1}{s+1} \end{bmatrix}$$

To solve for $x(t)$ with

$$\mathbf{x}(0) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{u}(t) = \begin{bmatrix} U_1(t) \\ U_2(t) \end{bmatrix}$$

where $\mathbf{u}(t)$ is a step function. We now have the parts to evaluate Equation 1. Multiplying the matrices out we get:

$$(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{x}(0) = \begin{bmatrix} \frac{1}{s+1} \\ 0 \\ \frac{1}{s+1} \end{bmatrix}$$

and

$$(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s) =$$

$$\begin{bmatrix} \frac{1}{s+1} & \frac{2}{s+2} - \frac{2}{s+3} \\ 0 & \frac{2}{s+3} - \frac{1}{s+2} \\ \frac{1}{s+1} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{s} \\ \frac{1}{s} \end{bmatrix} = \begin{bmatrix} -\frac{1}{s+1} - \frac{1}{s+2} + \frac{2}{3(s+3)} + \frac{4}{3s} \\ \frac{1}{2(s+2)} - \frac{2}{3(s+3)} + \frac{1}{6s} \\ \frac{1}{s} - \frac{1}{s+1} \end{bmatrix}$$

Therefore,

$$\mathbf{X}(s) = \begin{bmatrix} \frac{1}{s+1} \\ 0 \\ \frac{1}{s+1} \end{bmatrix} + \begin{bmatrix} -\frac{1}{s+1} - \frac{1}{s+2} + \frac{2}{3(s+3)} + \frac{4}{3s} \\ \frac{1}{2(s+2)} - \frac{2}{3(s+3)} + \frac{1}{6s} \\ \frac{1}{s} - \frac{1}{s+1} \end{bmatrix}$$

so that

$$\mathbf{x}(t) = \begin{bmatrix} e^{-t} - e^{-t} - e^{-2t} + \frac{2}{3}e^{-3t} + \frac{4}{3} \\ \frac{1}{2}e^{-2t} - \frac{2}{3}e^{-3t} + \frac{1}{6} \\ e^{-t} + 1 - e^{-t} \end{bmatrix}$$

$$= \begin{bmatrix} -e^{-2t} + \frac{2}{3}e^{-3t} + \frac{4}{3} \\ \frac{1}{2}e^{-2t} - \frac{2}{3}e^{-3t} + \frac{1}{6} \\ 1 \end{bmatrix} \quad (3)$$

We will analyze the transient and long-term behavior of this system below.

C. Solving for the Optimal Continuous Time Trajectory

Next, we need to find a policy $\mathbf{u}$ that optimizes some objective. The calculus of variations will prove to be useful in finding such a policy.

Mathematically, we can describe the objective function as:

$$J = \int_a^b f(x, y, y')\ dx$$

where we want to find the function y that minimizes J over the range $[a, b]$ and satisfies initial and final boundary conditions. In this paper, we consider that the goal is to reach a target state x_d, and $f(\cdot)$ is the distance squared as a cost function, $f = \mid x_t - x_d \mid^2$ or, for generality, $f = \mathbf{x^T Q x}$ in matrix form, where $\mathbf{Q}$ is a matrix of weights. Using Calculus of Variations, a necessary condition for an extremum function is that it satisfies the Euler-Lagrange equation:

$$\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right) = 0$$

The optimal control for a given objective function can be found in a similar manner. This will correspond to how we find an optimal policy in the case of continuous state spaces. For example, take the objective function [3]:

$$J = \int_0^{t_f} \mathbf{x}(t)^T \mathbf{Q}\mathbf{x}(t) + \mathbf{u}(t)\mathbf{R}\mathbf{u}(t)\ dt$$

First we define the pre-Hamiltonian as:

$$\mathcal{H}(x, u, p, t) = L(x, u, t) + p^T(t) f(x, u, t)$$

$$\dot{p} = \frac{\partial \mathcal{H}}{\partial \mathbf{x}}$$

$$\mathcal{H} = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \mathbf{p}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})$$

Pontryagin's minimum principle specifies that the optimal policy is one that minimizes the Hamiltonian at every time t [3]. The Hamiltonian for this example is minimized by setting:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0 = 2\mathbf{R}\mathbf{u} + \mathbf{p}^T\mathbf{B}$$

$$\mathbf{u}^*(t) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{p}(t)$$

where $\mathbf{p}(t)$ is the co-state vector that can be solved for through the use of boundary conditions. This gives us the optimal continuous time policy $\mathbf{u}^*(t)$ that minimizes J. Furthermore, the optimal continuous-time trajectory can be derived as $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}^*(\tau)\ d\tau$.

D. Solving for the Optimal Continuous Time Value Function

One can solve for the optimal value function by solving the Hamilton-Jacobi-Bellman (HJB) equation [4]. If we define the value function as:

$$V = \int_0^\infty \mathbf{x}(t)^T\mathbf{Q}\mathbf{x}(t) + \mathbf{u}(t)^T\mathbf{R}\mathbf{u}(t)\ dt$$

where $\mathbf{Q}$ and $\mathbf{R}$ are positive definite matrices. Then we can define a differential equivalent [5]:

$$0 = r(x, u) + \nabla V^T \dot{x}$$

The RHS of this equation is the Hamiltonian of the system.

$$\mathcal{H}(x, u, \nabla V) = r(x, u) + \nabla V^T \dot{x}$$

We can now define the HJB equation as:

$$0 = \min_u \mathcal{H}(x, u, \nabla V)$$

If we assume that the value function is a quadratic function in the Euclidean distance to the goal then we can derive an analytical solution for the value function based on solving the Lyapunov equation $\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} = 0$ [5].

$$V(x) = \mathbf{x}^T\mathbf{P}\mathbf{x}$$

$$\mathbf{P} = \int_0^\infty e^{t\mathbf{A}^T}\mathbf{Q}e^{t\mathbf{A}}\ dt$$

where $\mathbf{P}$ is the solution to the Lyapunov equation. Note that the integral in the definition of $\mathbf{P}$ doesn't need to be solved directly. Instead, $\mathbf{P}$ is the solution of the $n(n+1)/2$ simultaneous equations defined by the Lyapunov equation. Also, note that these results only hold if the origin is the goal. Therefore, if the goal is to control the system to another point that is not at the origin then a change of variables where the new origin is defined as $x_n = x - x_d$ must be carried out.

III. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a method of machine learning in which an agent learns a control policy by attempting to maximize the returned rewards for taking an action in a given state [6]. It does this by exploring the state and action spaces and observing the rewards in order to approximate the *value function* which is the expectation of the sum of discounted rewards for starting in state s and following the policy π. An important assumption made by the RL framework is that state changes are discrete; when an action is taken, a transition to a new state occurs, and the reward is observed, instead of happening continuously in time. If the value function is known, the optimal policy π^* can be derived as:

$$\pi^*(s) = \arg\max_{s'} V(s')$$

A. Markov Decision Processes

Much of the theory of reinforcement learning is based on what are called *Markov Decision Processes* (MDPs) [7]. A MDP is defined on a set of states S and actions A available to an agent. The size of the state and action spaces are traditionally assumed to be finite. The reward function $R(s, a)$ defines the feedback given to the agent for being in state s, taking action a, and ending in state s'. Actions can either be deterministic or non-deterministic. If the current state depends only on the previous state and action taken then the problem is said to satisfy the *Markov property*. Given this definition of an MDP, we now define the value function for a given policy (mapping of states to actions) π as:

$$V^\pi(s) = E_\pi[R(s)] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{k+1}]$$

Where γ is a *discount factor* that tunes how much the agent should care about future rewards versus the current reward. In other words, the value of being in state s is equal to the expected value of sum of the current reward and all future rewards for being in state s and following π until termination.

If the reward and transition functions are known, an MDP can be solved by employing dynamic programming techniques [8]. Value iteration is one such technique in which one can iterate on the value function by taking the max over the actions $V(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$. The iterations continue until the LHS of the equation is equal to the RHS. The equation above is called the *Bellman equation*, which is based on the Bellman Principle of Optimality.

B. Q-learning

Some typical methods of reinforcement learning, such as the value iteration algorithm, require the learner to know some model of the underlying system (i.e. the reward function or state-transition function). However, it is often the case that one does not know these functions. Q-learning can be used to learn policies without having to know a-priori the reward or state-transition functions. Instead of a value function, we now define a Q-function that represents the value of being in state s and taking action a:

$$Q^\pi(s,a) = E_\pi[R(s,a)] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{k+1}]$$

The Q-function is typically represented as a table called a Q-table where the rows and columns are the finite states and actions. Q-learning can be extended to infinite state spaces by representing Q-functions as the sum of weighted basis functions:

$$\hat{Q}(s,a) = \sum_{i=1}^{N} w_i \phi_i(s,a)$$

This representation requires choosing a set of basis functions that can accurately represent the true value function for a given problem.

IV. CONNECTING REINFORCEMENT LEARNING AND CONTROL THEORY

Optimal control theory and reinforcement learning share the goal of obtaining an optimal control policy for a particular problem. The difference between the two is the knowledge of the model and assumptions about time. In the optimal control theory case, the control policies are derived exactly with full knowledge of the system and assume continuous time. In the case of reinforcement learning, the model of the system might not be fully or even partially known, and the policies are derived based on interaction with the environment over discrete periods of time, Δt. In this section, we describe the control policies obtained from both fields in order to derive some insight as to how the two different policies change with respect to Δt. The questions we address are: 1) Is the value function in the discrete case $V_{\Delta t}$ representable as a linear mapping? 2) How closely would it approximate the continuous time V^* for a finite Δt?

A. System Behavior Decomposition

In this section, we prove that a given linear system can be decomposed into a linear function of the transient and long term behaviors of the system.

Theorem 1: If $\mathbf{A}$ has both *real* and *distinct* eigenvalues, then each entry in the solution to the state equation (Eq. 2) can be written as the sum of exponentials and constants.

Recall the solution to the state space equations in the Laplace domain given by Equation 1. Note that $x(0)$ and $\mathbf{B}$ are both constant and $\mathbf{U}(s)$ is a step input. Therefore, let us focus on the $(s\mathbf{I} - \mathbf{A})^{-1}$ term first.

Using $|s\mathbf{I} - \mathbf{A}| = 0$, we can obtain the eigenvalues of $\mathbf{A}$, which we assume by the theorem are both real and distinct. We can write:

$$|s\mathbf{I} - \mathbf{A}| = (s - \lambda_1)(s - \lambda_2)...(s - \lambda_n)$$

which is a polynomial of degree n.

Let M_{ij} be the matrix obtained by eliminating the i^{th} row and j^{th} column from $\mathbf{A}$. $|M_{ij}|$ is called the minor of a_{ij} and has a degree $\leq (n-1)$. Let

$$A_{ij} = (-1)^{i+j} |M_{ij}|$$

be called the cofactor of a_{ij}. The adjoint of $\mathbf{A}$ is the matrix in which:

$$adj(A)_{ij} = A_{ji}$$

In other words, the adjoint of $\mathbf{A}$ is the transpose of the matrix of cofactors. Since

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{adj(s\mathbf{I} - \mathbf{A})}{|s\mathbf{I} - \mathbf{A}|}$$

We can use Eqs 1, 2, and 3 to write:

$$\frac{adj(s\mathbf{I} - \mathbf{A})_{ij}}{(s - \lambda_1)(s - \lambda_2)...(s - \lambda_n)}$$

$$= \frac{(s\mathbf{I} - \mathbf{A})_{ji}}{(s - \lambda_1)(s - \lambda_2)...(s - \lambda_n)}$$

$$= \frac{(-1)^{j+i} |M_{ji}|}{(s - \lambda_1)(s - \lambda_2)...(s - \lambda_n)}$$

Because the degree of $|M_{ji}|$ is at most $(n-1)$ the degree of the numerator is strictly less than the degree of the denominator.

Therefore, we can use partial fraction decomposition since there are no repeated roots in the denominator and the degree of the numerator is strictly less than the degree of the denominator to get the form:

$$\frac{k_1}{(s - \lambda_1)} + \frac{k_2}{(s - \lambda_2)} + ... + \frac{k_n}{(s - \lambda_n)}$$

Finally, taking the inverse Laplace transform we get:

$$k_1 e^{\lambda_1 t} + k_2 e^{\lambda_2 t} + ... + k_n e^{\lambda_n t}$$

for each entry in $(s\mathbf{I} - \mathbf{A})^{-1}$ as shown below.

$$(s\mathbf{I}-\mathbf{A})^{-1} = \begin{bmatrix} k_{1_{11}}e^{\lambda_{11}t} + ... + k_{n_{1n}}e^{\lambda_{1n}t} \\ k_{1_{21}}e^{\lambda_{21}t} + ... + k_{n_{2n}}e^{\lambda_{1n}t} \\ \vdots \\ k_{1_{n1}}e^{\lambda_{n1}t} + ... + k_{n_{nn}}e^{\lambda_{1n}t} \end{bmatrix}$$

Since $\mathbf{x}(0)$ is a vector of constants for the first half of equation 1 we get:

$$= \begin{bmatrix} c_1k_{1_{11}}e^{\lambda_{11}t} + ... + c_1k_{n_{1n}}e^{\lambda_{1n}t} \\ c_2k_{1_{11}}e^{\lambda_{11}t} + ... + c_2k_{n_{1n}}e^{\lambda_{1n}t} \\ \vdots \\ c_nk_{1_{11}}e^{\lambda_{11}t} + ... + c_nk_{n_{1n}}e^{\lambda_{1n}t} \end{bmatrix}$$

For the second half of equation 1, we have two parts:

$$(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} = \begin{bmatrix} b_1k_{1_{11}}e^{\lambda_{11}t} + ... + b_1k_{n_{1n}}e^{\lambda_{1n}t} \\ b_2k_{1_{11}}e^{\lambda_{11}t} + ... + b_2k_{n_{1n}}e^{\lambda_{1n}t} \\ \vdots \\ b_nk_{1_{11}}e^{\lambda_{11}t} + ... + b_nk_{n_{1n}}e^{\lambda_{1n}t} \end{bmatrix}$$

For the unit step in each dimension of $\mathbf{U}(s)$:

$$\mathbf{U}(s) = \frac{1}{s}\mathbf{K^T} = \begin{bmatrix} \frac{k_1}{s} & \frac{k_2}{s} & \dots & \frac{k_n}{s} \end{bmatrix}^T$$

we take the inverse Laplace transform to get:

$$\mathbf{U}(t) = \mathbf{U} = \mathbf{K^T}H(t)$$

where $H(t)$ is the Heaviside step function. So we can see that $(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)$ is equal to the matrix for $(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}$. Therefore,

$$\mathbf{x}(t) = \mathcal{L}^{-1}((s\mathbf{I}-\mathbf{A})^{-1}\mathbf{x}(0) + (s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s))$$
$$= \mathcal{L}^{-1}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{x}(0) + \mathcal{L}^{-1}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)) =$$
$$\begin{bmatrix} C_1k_{1_{11}}e^{\lambda_{11}t} + ... + C_1k_{n_{1n}}e^{\lambda_{1n}t} \\ C_2k_{1_{21}}e^{\lambda_{21}t} + ... + C_2k_{n_{2n}}e^{\lambda_{2n}t} \\ \vdots \\ C_nk_{1_{n1}}e^{\lambda_{n1}t} + ... + C_nk_{n_{nn}}e^{\lambda_{nn}t} \end{bmatrix}$$

where $C_i = k_i(b_i + c_i)$ and i is equal to the row index. Therefore, as long as the eigenvalues of the matrix $\mathbf{A}$ are both *real* and *distinct* the solution to the state equation can be written as a sum of exponential functions and constants.

If we group the exponential terms together, they form what we call the transient response matrix $\mathcal{T}$ of the system. The constants form the long term response matrix $\mathcal{N}$ of the system. The λ terms in the transient response matrix are the time constants of the system. We can now define:

$$x(t) = \mathcal{T}(t) + \mathcal{N}$$

For the example problem in section II, separating exponentials from constants in equation 3, we have:

$$\mathcal{T} = \begin{bmatrix} -e^{-2t} + \frac{2}{3}e^{-3t} \\ \frac{1}{2}e^{-2t} - \frac{2}{3}e^{-3t} \\ 0 \end{bmatrix}, \quad \mathcal{N} = \begin{bmatrix} \frac{4}{3} \\ \frac{1}{6} \\ 1 \end{bmatrix}$$

The smallest time constant is given by $\tau = \frac{1}{3}$. Thus, the system reaches about 37% of the way to the new equilibrium point $\begin{bmatrix} \frac{4}{3} & \frac{1}{6} & 1 \end{bmatrix}^T$ in about 1/3 sec.

B. Response LTI Systems to Actions

Unlike general applications of RL, we can derive a transition function from knowledge of a linear system. For any action held constant for Δt, we can predict the discrete response $x_t \rightarrow x_{t+1}$. The previous section allows us to see that given any step input we can decompose the system into a set of constants and exponential functions. In this section, we will show that we can derive the *state transition function* of an LTI system given the matrix A and a Δt over which a constant action u is held. In addition, we show that it is a linear transformation in the state space of the system. We start by defining Δx as the difference in state at time t from the initial state.

$$\Delta\mathbf{x} = \mathbf{x}(t) - \mathbf{x}(0)$$

We then plug in the solution for $\mathbf{x}(t)$.

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \mathcal{L}^{-1}((s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s))$$
$$\Delta\mathbf{x} = e^{\mathbf{A}t}\mathbf{x}(0) + \mathcal{L}^{-1}((s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)) - \mathbf{x}(0)$$
$$\Delta\mathbf{x} = (e^{\mathbf{A}t} - \mathbf{I})\mathbf{x}(0) + \mathcal{L}^{-1}((s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s))$$

Since $\mathbf{U}$ is a step input we can represent it as a constant $\mathbf{U}$.

$$\Delta x = \mathbf{M}(\Delta t)\mathbf{x}(0) + \mathbf{L}$$

where

$$\mathbf{M}(\Delta t) = (e^{\mathbf{A}\Delta t} - \mathbf{I})$$

and

$$\mathbf{L} = \mathcal{L}^{-1}((s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U})$$

Therefore, for a given Δt the matrix $\mathbf{M}$ and the vector $\mathbf{L}$ are constants, meaning we can predict, for a given state, Δt and control input, our next state discrete x_{t+1}. We can also see how this transformation is a simple linear transformation of coordinates in state space (given a fixed Δt) as shown in Figure 2.

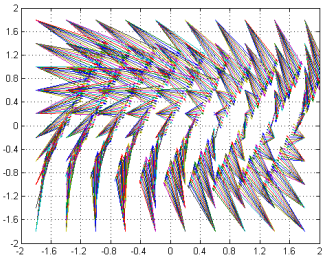


Fig. 2: A depiction of the linear mapping for an input action for the example problem. Each vector shows the result of taking the same action from different coordinates in state space and holding it for $\Delta t = 0.2$.

So in theory, since we can extract the transition function from knowledge of the dynamics, then it should be possible

learn a control policy (for a given reward function) using a dynamic programming technique such as value iteration. However, our goal is to show that a controller can be learned for a linear system without requiring the model (i.e. using Reinforcement Learning).

C. Approximating the continuous-time value function

Although reinforcement learning is often presented in terms of discrete state spaces, it can be extended to continuous state spaces by representing the value function as a linear combination over a set of continuous basis functions [2]. That is, given a basis set $\phi_1(s)$ to $\phi_n(s)$.

$$V = \sum_{i=1}^{n} w\phi_i(s)$$

where w_i is a weight for each ϕ_i. Two important questions in applying discrete-time RL to learning control of a continuous-time LS are: 1) can the continuous-time value function be approximated as a linear combination?, and 2) over what basis set? Since the Lyapunov equation (above) shows that if the reward is quadratic ($r = \mathbf{x^T Q x}$, then the optimal continuous-time value function can also be expressed as a quadratic function $V^* = \mathbf{x^T P x}$. Expanding this matrix expression out as a polynomial, $V^* = \Sigma_i \Sigma_i P_{ij} x_i x_j$. Therefore, a natural set of basis functions is the $n(n+1)/2$ combinations of products of the state space variables (up to second-order). For the example in the previous section, we would choose: $[\phi_1 ... \phi_n] = [x_1^2,\ x_1x_2,\ x_1x_3,\ x_2^2,\ x_2x_3,\ x_3^2]$

Theorem 2 (Representation Theorem): The optimal continuous-time value function V^* is representable as linear combination over a quadratic basis set, $V^* = \sum_{i=1}^{n(n+1)/2} w_i\phi_i(s)$ where $[\phi_1 ... \phi_{n(n+1)/2}] = [x_1^2, x_1x_2 ..., x_ix_j, ...x_n^2]$.

We show below that the optimal discrete-time value function $V_{\Delta t}$ can be made to approximate V^*. Thus the Representation Theorem shows that V^* can be learned as a target by using discrete-time methods (e.g. Reinforcement Learning) by learning weights over a quadratic basis set. However, as shown in the next section, the accuracy of approximation will depend on the size of the time step, Δt.

D. Effect of Δt

If we assume continuous control inputs that are bounded by some constant μ (i.e. $|U| \leq \mu$) then a *reachable* region from the current state is defined given a Δt. This is illustrated in Figure 3 as the light gray region. A state x_2 is said to be *reachable* from x_1 if there exists a control input $u(t)$ such that $|u(t)| < \mu$ that can transfer the system from x_1 to x_2 in finite amount of time. It should be noted that the shape of the region is dependent on the dynamics of the system and is not necessarily circular. The optimal trajectory, $x^*(t)$, is represented as the solid black line in the figure and a constant input trajectory for a given action is shown as a dotted line. The dotted line represents choosing an action and holding it for Δt. It is curved because of the non-linear, transient response of the system.

Theorem 3: Assuming bounded, continuous control inputs, there exists an action that intersects with either the optimal trajectory along the boundary of the reachable region or the goal if it is inside the reachable region and this action belongs to the optimal policy.

This can be proven by contradiction. The policy we learn, once converged, has the property that the action chosen for a given state maximizes the value function. That is

$$\pi^*(s) = \arg\max_{s'} V(s')$$

which means that $V(s') \geq V(s)\ \forall s \in S$ where S is the full state space. In addition, the optimal trajectory has the property that it maximizes the optimal value function for every point along optimal trajectory. That is: $V^*(s) \geq V(s')\ s' \neq s$. Therefore, if we choose an action that leads to a state not at the intersection of the optimal trajectory and the reachable region's boundary then its value is less than the optimal value. This violates our assumption that $V(s') \geq V(s)\ \forall s \in S$ because we could choose an action with a higher value by choosing the action at leads to the intersection point.

Next, we show that as $\Delta t \to 0$, we get better control policies. For a smaller Δt, the reachable region is smaller which implies the maximum distance away from the optimal trajectory is also smaller. The error of the policy for a given Δt is defined by the difference of the integral of the distance to the goal for the trajectories produced by both policies.

An important goal is to show that the value function learned by reinforcement learning will converge in the limit (as $\Delta t \to 0$) to the optimal value function for continuous time control,

$$V^* = \int_0^\infty (\mathbf{x(t)} - \mathbf{x_d})^2\ dt$$

Although the discrete-time value function is conventionally defined $V = \Sigma \gamma^i r_t$, in order to get it to converge to V^*, we modify by setting $\gamma = 1$ and multiply by Δt:

$$V_{\Delta t} = \Delta t\ \Sigma_{t=0}^{\infty}\ r_t = \Delta t\ \Sigma_{t=0}^{\infty}\ (\mathbf{x(t)} - \mathbf{x_d})^2$$

Since for any time step Δt, there is always a constant action that can move the system to the same point on the continuous time trajectory, as argued above, then the rewards $r_t = (\mathbf{x(t)} - \mathbf{x_d})^2$ will match at these discrete points. Thus, the discrete-time value function is simply the numerical approximation of the continuous-time value function, V^*.

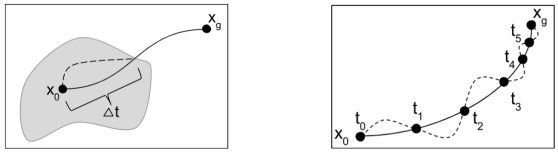
$$V^* = \int_0^\infty (\mathbf{x(t)} - \mathbf{x_d})^2\ dt \approx \Sigma_{t=0}^{\infty}\ \Delta t (\mathbf{x(t)} - \mathbf{x_d})^2$$

$$\lim_{\Delta t \to 0} V_{\Delta t} \to V^*$$

Furthermore, we can we put a bound on the error in the value function. The error is a function of the time step size (as with any numerical quadrature), as well as the maximum deviation between the optimal continuous-time trajectory and the constant-input trajectory between the discrete time points, where the coordinates coincide. This is bounded because the

amount the constant-input trajectory can diverge from the continuous time path within Δt is limited by the dynamics of the system, which will force them to follow similar paths.

The response to a given input, and therefore the error, is dependent on the underlying system dynamics and it is also dependent on how quickly a system responds to a new input. The smallest system time constant, τ, defines how rapidly the system responds to a new control input. Therefore, we propose that a requirement for the selection of a Δt is that it be less than τ in order to ensure that these rapid responses can be controlled. At $\Delta t = dt$ the optimal trajectory will equal the approximate trajectory through state space.



(a) (b)

Fig. 3: Differences between the optimal continuous-time trajectory and the trajectory produced by a sequence of discrete inputs. Solid line shows continuous-time trajectory. Shaded region shows the reachable space within Δt. Dashed line shows trajectory by a sequence of constant inputs.

V. EXPERIMENTS

We analyzed and tested a stable, 2-dimensional system with the dynamics defined by the matrices $\mathbf{A} = [[-1\ \ 2], [-1\ \ -4]]$ and $\mathbf{B} = [1\ \ 1]^T$, and evaluated the effect of Δt on policies learned by Q-learning. This system is stable because $\mathbf{A}$ is negative definite. The time constants of the system are: $\tau_1 = \frac{1}{3}$ and $\tau_2 = \frac{1}{2}$. The system was implemented in MATLAB using fourth order Runge-Kutta methods to simulate the system dynamics with a time step of $\Delta t = 0.001$ seconds. The goal is to show that the accuracy of the learned controller increases as $\Delta t \to 0$.

We used the Q-learning algorithm to learn the weights of the quadratic functions of the state variables. The system is set in a random initial location for an episode and uses an ϵ-greedy action selection mechanism [9] to explore the state space. The actions available to the learner were those between -10 and 10 with steps of 0.1 and the goal was to reach (0,0).

As shown in Figure 4, as $\Delta t \to 0$ the value estimate increases which represents a better control policy. This figure was generated by, first, learning for each Δt until the Q-function converged which we defined as 10000 weight updates with maximum weight changes of less than 10^{-12} or 5000 episodes completed. Second, we tested each policy by setting $x_0 = (2,2)$ and running the system until the goal is reached. The squared distance to the goal accumulated along the trajectory is the value presented in the graph. Note that $V_{\Delta t}$ plateaus below the lowest time constant of the system. This suggests that there are diminishing returns to reducing Δt further than the start of the plateau and that a Δt of 0.01 would be adequate to learn a near-optimal controller given the dynamics of the system.

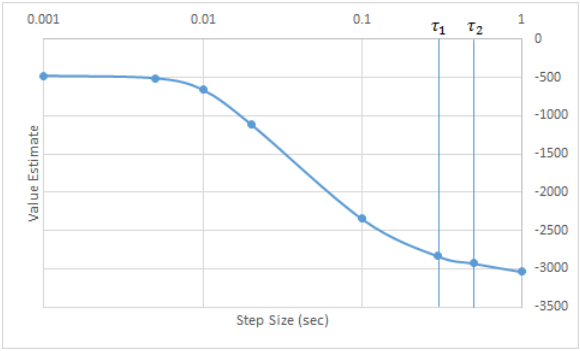


Fig. 4: The value estimates from the learned policies as a function of Δt.

VI. DISCUSSION

In this paper, we have discussed how it is possible to decompose a system into a transient response and a long term response. The decomposition provides some insight as to the selection of a control time for a given learner and enables one to use methods such as value iteration to find an optimal value function $V_{\Delta t}$. We have shown that for a quadratic cost (or reward) function the optimal value function is also quadratic and it is representable by a weighted linear combination of a quadratic basis set. Finally, we have shown that as $\Delta t \to 0$ the approximation to the optimal value function is more accurate and therefore we get better control policies. This idea was corroborated by the tests on an example dyanmical system where we see that the Δt should be less than the smallest time constant of the system in order to generate a near-optimal policy.

REFERENCES

[1] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[2] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=945365.964290

[3] W. L. Brogan, *Modern Control Theory (3rd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[4] M. Johnson, R. Kamalapurkar, S. Bhasin, and W. E. Dixon, "Approximate-player nonzero-sum game solution for an uncertain continuous nonlinear system," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 8, pp. 1645–1658, 2015.

[5] F. L. Lewis, D. L. Vrabie, and V. L. Syrmos, *Reinforcement Learning and Optimal Adaptive Control*. John Wiley & Sons, Inc., 2012, pp. 461–517. [Online]. Available: http://dx.doi.org/10.1002/9781118122631.ch11

[6] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996. [Online]. Available: http://people.csail.mit.edu/lpk/papers/rl-survey.ps

[7] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[8] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[9] M. Tokic and G. Palm, *KI 2011: Advances in Artificial Intelligence: 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax, pp. 335–346. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24455-1_33