Balancing Inverted Pendulum Using MDP Approximation

Jingpei Lu JIL360@UCSD.EDU
Department of Electrical & Computer Engineering
University of California, San Diego

1 Problem Formulation

1.1 Problem and Parameter Setup

In this problem, we will solve the optimal control problem of balancing an inverted pendulum. Consider the pendulum state $\boldsymbol{x} = (x_1, x_2)^T$. The dynamical system is:

$$dx = f(\boldsymbol{x}, u)dt + \sigma d\omega \tag{1}$$

$$f(\boldsymbol{x}, u) := \begin{bmatrix} x_2 \\ (asinx_1 - bx_2 + u) \end{bmatrix} \tag{2}$$

$x_1 \in \theta \coloneqq [-\pi, \pi]$ is the angle of the pendulum, $x_2 \in \dot{\theta} \coloneqq [-8,8]$ is the angular velocity, $u \in [-2,2]$ is the control, ω is Brownian motion. The cost function is

$$l(\boldsymbol{x}, u) = 1 - \exp(k * cosx_1 - k) + \frac{r}{2}u^2 \tag{3}$$

the parameters $(a, b, \sigma, k, r, \gamma)$ is set to be (1, 0.5, 0.001, 2, 1e-6, 0.9). We want to solve it using value iteration and policy iteration.

Since the control and state spaces are continuous, we need to discretize them. First, we choose the time step $\delta t = 0.05$, maximum velocity $x_{2,\max} = 8$ and maximum control $u_{\max} = 2$. We discretize the state into (40, 10) number of grid points and discretize the control space into 17 discrete controls. So, for a given state $(x_{1,c}, x_{2,c})$ in continuous space, we can discretize it as

$$x_1 = int\left(\frac{(x_{1,c} - x_{1,\min})}{stride_1}\right) \qquad x_2 = int\left(\frac{(x_{2,c} - x_{2,\min})}{stride_2}\right)$$

$$stride_1 = \frac{x_{1,max} - x_{1,min}}{40} \qquad stride_2 = \frac{x_{2,max} - x_{2,min}}{10}$$

1.2 Markov Decision Process (MDP) Formulation

After discretizing the continuous states into sets of discrete state θ and $\dot{\theta}$, we can formulate this problem as a finite-state MDP as:

- The state space is $\boldsymbol{x} \in X \coloneqq \{\theta \times \dot{\theta}\}$, which is the cartesian product between angle and angular velocity. And $\boldsymbol{x} = (x_1, x_2) \in \Re^2$, where $x_1 \in \theta$ and $x_2 \in \dot{\theta}$.
- The control space is $u \in \mathcal{U} \coloneqq [-2,2]$, which is the torque applied to the pendulum
- The initial state x_0 can be defined randomly as $x_0 = (x_{1,0}, x_{2,0})$, where $x_{1,0} \in \theta$ and $x_{2,0} \in \dot{\theta}$
- The motion model $p_f(x_{t+1}|x_t, u_t)$ is Gaussian with mean $\boldsymbol{x} + f(\boldsymbol{x}, u)\delta t$ and covariance $\sigma\sigma^T\delta t$, where $f(.)$ is defined in equation (2).
- The stage cost is $l(\boldsymbol{x}, u)\delta t$ for $\boldsymbol{x} \in X$, where $l(\boldsymbol{x}, u)$ is defined in equation (3)

1.3 Optimal Control Problem

Given this MDP formulation, our goal is to find the optimal policy that minimizes the long-term cost for every state by solving the Bellman Equation:

$$V^*(x) = \min_{u \in \mathcal{U}(x)} (l(x, u) + \gamma \sum_{x' \in X} p_f(x'|x, u)V^*(x')) \qquad \forall x \in X \tag{4}$$

1.4 Interpolation Problem

Since we solve this problem by first formulate it as a discrete MDP problem, we need to approximate the solution of the original continuous problem by interpolating the resulting discrete space optimal policy into a continuous space policy, so that we can obtain the optimal policy π for any state $x_c \in \mathcal{R}^2$ in continuous state space X_c.

2 Technical Approach

2.1 Value Iteration Algorithm

Value iteration is a method of computing an optimal MDP policy and its value. My implementation of the Value Iteration Algorithm is described as below.

Algorithm 1: Value Iteration Algorithm

1: Initialize $V(x) = 0$, for $\forall x \in X$
2: Loop for episode 1, 2, … 50:
3: $\quad \delta = 0$
4: $\quad$ For each $x \in X$:
5: $\quad\quad v \leftarrow V(x)$
6: $\quad\quad V(x) \leftarrow \min_{u \in \mathcal{U}(x)} (l(x,u) + \gamma \sum_{x' \in X} p_f(x'|x,u) V^*(x'))$
7: $\quad\quad \delta \leftarrow \max(\delta, |v \leftarrow V(x)|)$
8: $\quad$ If $\delta <$ some threshold: break loop
9: For each $x \in X$:
10: $\quad \pi(x) \leftarrow \underset{u \in \mathcal{U}(x)}{\operatorname{argmin}} (l(x,u) + \gamma \sum_{x' \in X} p_f(x'|x,u) V^*(x'))$

Figure 1: Value Iteration Algorithm

2.2 Policy Iteration Algorithm

While value-iteration algorithm keeps improving the value function at each iteration until the value-function converges, policy-iteration instead of repeated improving the value-function estimate, it will re-define the policy at each step and compute the value according to this new policy until the policy converges. My implementation of the Policy Iteration Algorithm is described as below.

Algorithm 2: Policy evaluation

1: For given policy π
2: Initialize $V(x) = 0$, for $\forall x \in X$
3: Loop for episode 1, 2, … 50:
4: $\quad$ Loop for each $x \in X$:
5: $\quad\quad V(x) = \sum_{x'} p\big(x' \big| x, \pi(x)\big)[l\big(x, \pi(x)\big) + \gamma V(x')]$

Algorithm 3: Policy improvement

1: For given value function V
2: Loop for each $x \in X$:
3: $\quad \pi(x) = argmin_u \sum_{x'} p(x'|x,u)[l(x,u) + \gamma V(x')]$

Algorithm 4: Policy Iteration Algorithm

1: Initialize $V(x) = 0$ and $\pi(x) \in \mathcal{U}$ randomly, for $\forall x \in X$
2: Loop for episode 1, 2, … 50:
3: Given $\pi(x)$, running Policy Evaluation to estimate $V^{\pi}(x)$ for $\forall x \in X$
4: Given $V^{\pi}(x)$, obtain $\pi'(x)$ by running Policy Improvement one time.
5: $\pi(x) = \pi'(x)$ for $\forall x \in X$

Figure 2: Policy Iteration Algorithm

2.3 Policy Interpolation

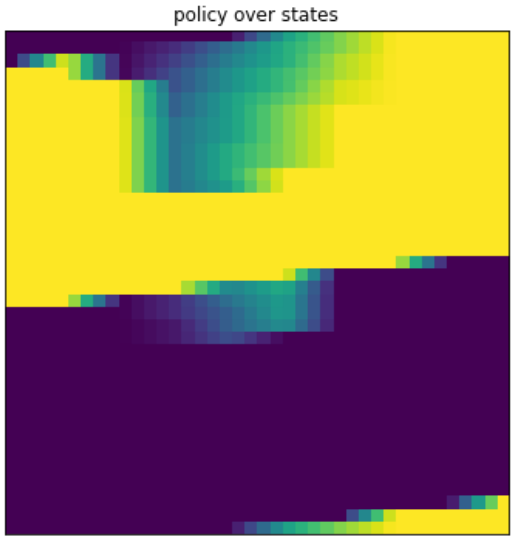
After obtaining the π^* in discrete state space X, we can approximate the policy π^* to continuous state space X_c using linear interpolation. For given state $x_c \in X_c$, we approximate $\pi(x_c)$ by first finding its two nearest state in discrete state space $x_d, x_{d+1} \in X$. Then we can estimate the $\pi(x_c)$ as

$$\pi(x_c) = \pi(x_d) + \big(\pi(x_{d+1}) - \pi(x_d)\big)\frac{x_c - x_d}{x_{d+1} - x_d} \tag{5}$$

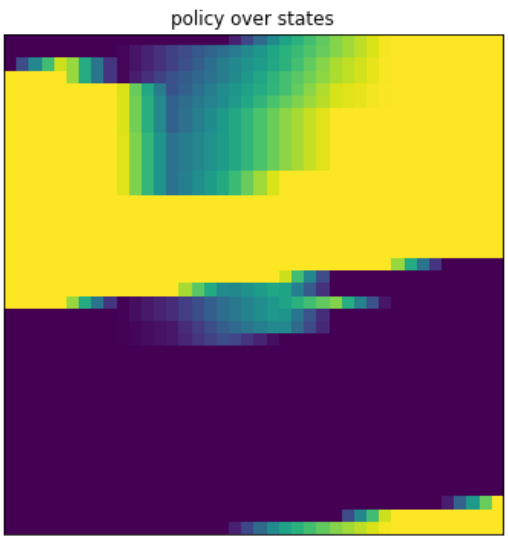
3 Results

3.1 The optimized policy over the state space

Left: value iteration



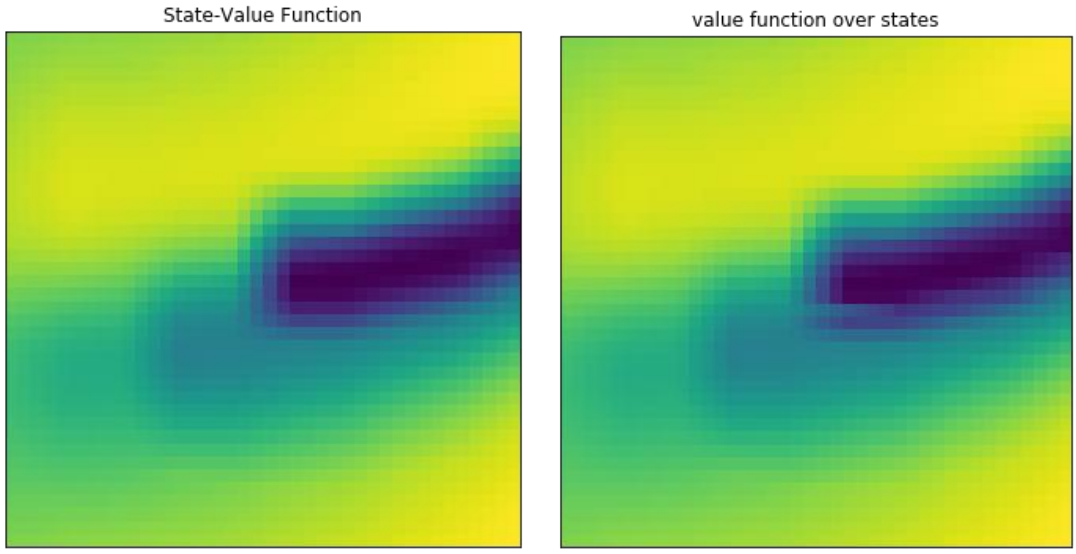
Right: policy iteration



From the figures above, we can see that the optimal policy over 50 episodes are very similar for these two algorithms.

3.2 The value function over the state space

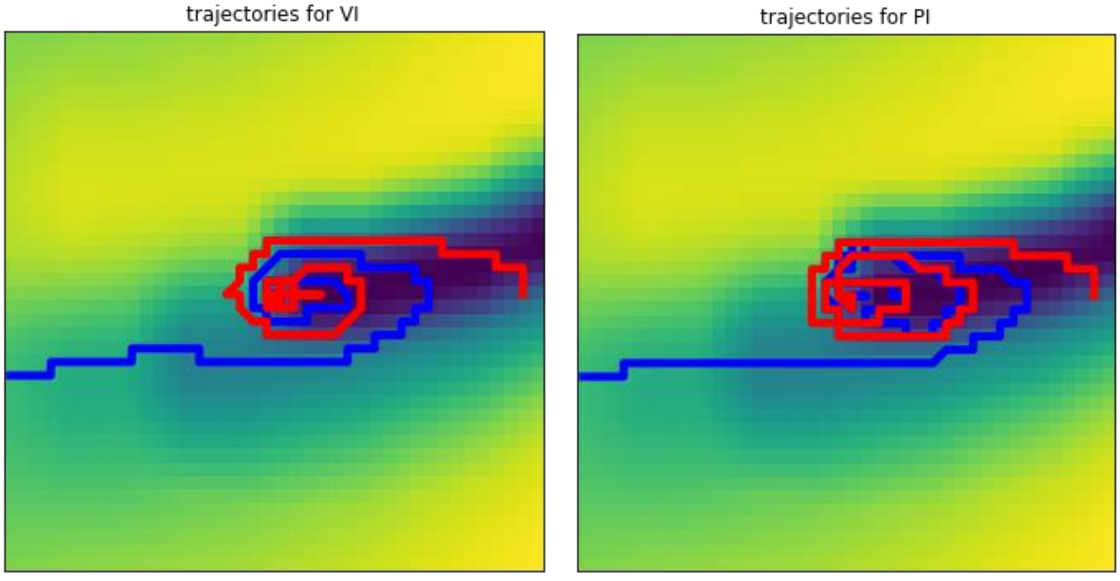
Left: value iteration Right: policy iteration



From the figures above, we can see that the final value functions over 50 episodes are very similar for these two algorithm.

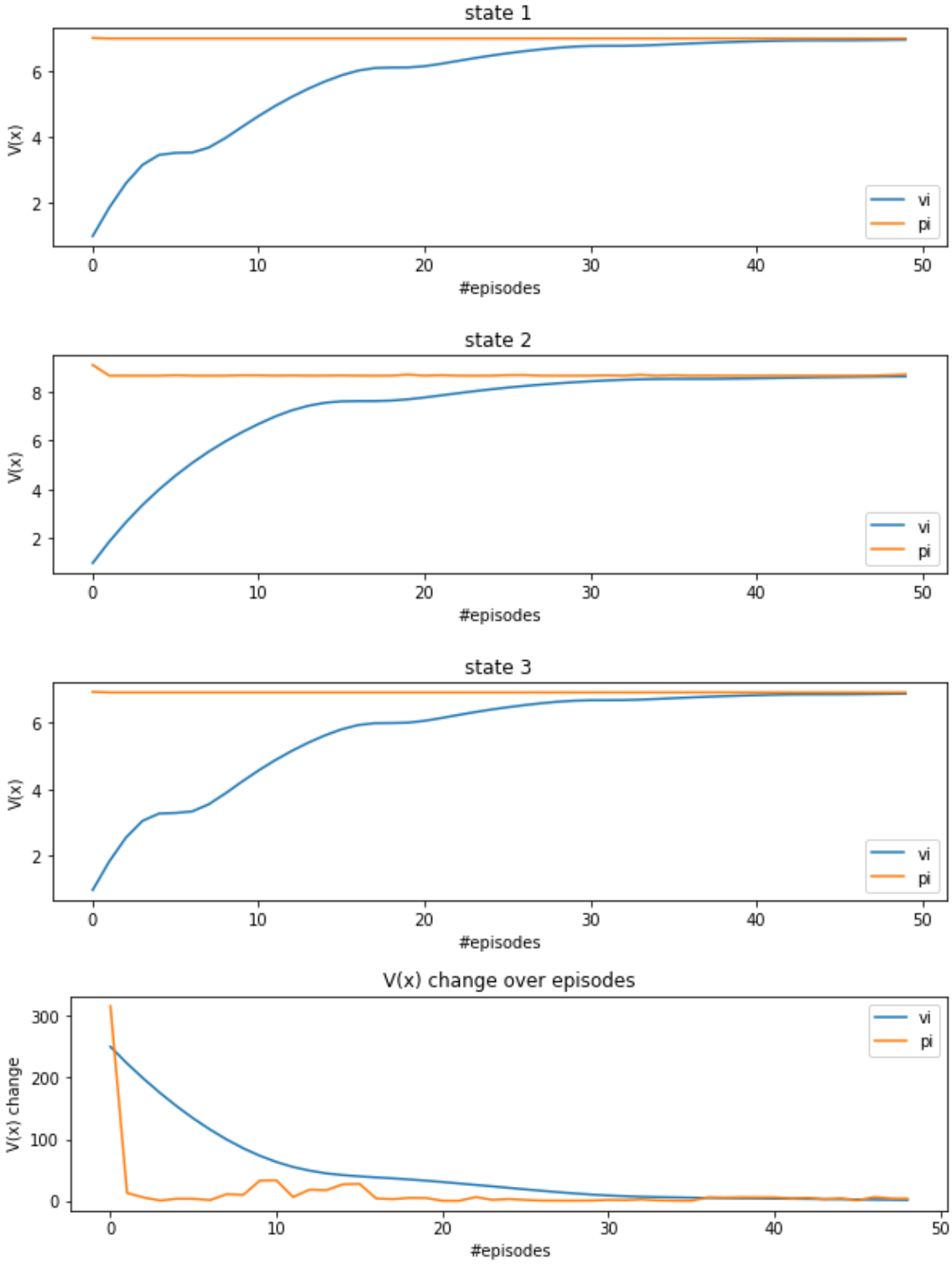
3.3 Trajectory simulation

Left: value iteration Right: policy iteration



From the figures above, we can see the trajectories of different starting points. Both algorithms converge to the center of the plot with slightly different trajectories.

3.4 V(x) over episodes between VI and PI for a set of states



From the figures above we can see that the Policy Iteration (PI) converges much faster than the Vale Iteration (VI). However, in practice, PI runs much slower than VI.