ECE 276B
Project 4

Linear Quadratic Control & Reinforcement Learning

Due Date: March 24, 2018

Professor: Nikolay Atanasov

Chuqiao Song

A53239614

Introduction

Balancing the Acrobot Using Linear Quadratic Control:

In this part, I am trying to balance the acrobot system, a deterministic double pendulum system, by using linear quadratic control method in finite horizion. Previously, the single pendulum problem has been studied which is a low-dimensional model system. However, real world involves lot of complexity than that simple model. In general, by study this acrobot system, people can use the ideal behind it to build robotic arm with two degrees of freedom. In linear quadratic control, the dynamic system is linear in the control of u and state x, and the stage-cost is quadratic in the control of u and state x. By these two assumptions, the quadratic cost of a deterministic finite horizon linear system can be minimized by using Riccati differential equation.

Reinforcement Learning of Hill Automobile Problem:

In part two, I am trying to solve a hill automobile problem by using two model-free method, Monte Carlo and Temporal Difference method. The task is to find a control policy that takes advantage of the potential energy obtained by swing between two hills to let the car starting from valley with underpowered engine climb up the steep hill on the right. Previously, the model-based MDP (Markov Decision Process) problem are solving by value iteration and policy iteration. In reinforcement learning, we assume there is no knowledge of the MDP motion model or cost function, but it still can be solved by accessing to examples of system transitions and incurred costs with so called model-free method. In general, Monte Carlo method and Temporal Difference method are two commonly used reinforcement-learning algorithms, which I will talk in following sections.

Problem Formulation

1) Balancing the Acrobot Using Linear Quadratic Control:

Linearizing the acrobot dynamic system: Given the nonlinear acrobot system dynamics, the first thing is to linearize the dynamics around $x_0 = (0,0,0,0)$ to achieve $\dot{x} = A_0\, x + B_0\, u$, where u is a scalar.

Compute a quadratic stage cost approximation: Given the original stage cost, the second thing is to achieve a quadratic approximation of the original stage cost in the form of $g_0(x,u) = \frac{1}{2}\, x^T Q x + \frac{r}{2} u^2$.

Solving linear quadratic regulator problem: Given calculated linear system as well as quadratic cost, this LQR problem can be solved by using Riccati differential equation.

Test the policy in nonlinear system: By applying the policy calculated from LQR problem to original nonlinear system, whether this policy works well have to be checked.

2) Reinforcement Learning of Hill Automobile Problem:

Formulating MDP problem: Given the formulated hill automobile car problem, the states and control space has to be discretized to form MDP tuple (X, U, cost, discount factor)

Implementing Monte Carlo Method: Given the initial point and policy matrix for each loop, a complete episode should be generated. Also, the Q matrix should be updated (policy evaluation), where Q is state-action value. Then, given the updated Q matrix, the policy improvement should be done to update policy matrix.

Implementing Temporal Difference Method: Given the initial point and policy matrix for each subsection of the episode (an in complete episode with $S_t, A_t, R_t, S_{t+1}, A_{t+1}$ sequence) should be generated, and corresponding Q value should be updated. Also, for each incomplete episode, the corresponding policy matrix should be updated based on Q matrix.

Technical Approach

1) Balancing the Acrobot Using Linear Quadratic Control:

In general, to solve a dynamic system getting optimal control whether in linear or non-linear, we have to approximate the system in linear form and the cost function in quadratic form. By formulating the problem in this way, we can solve Finite-horizon, continuous-time LQR with Riccati differential equation (RDE), and solve Infinite-horizon, continuous-time LQR with algebraic Riccati equation (ARE). However, since the RDE is hard to solve, we are assuming it is an infinite horizon problem for this acrobot system, and use ARE to solve this problem. In general, to formulate whole system, I use Python library *sympy*, to do algebraic operation in symbolic level. The original dynamic system is shown below, where the ζ represents friction and g represents gravity acceleration.

$$x = (\theta_1, \quad \theta_2, \quad \dot{\theta}_1, \quad \dot{\theta}_2)^T$$

$$\text{moment of Inertia :} \quad M(x) = \begin{bmatrix} 3 + 2\cos(x_2) & 1 + \cos(x_2) \\ 1 + \cos(x_2) & 1 \end{bmatrix}$$

$$\text{coriolis, centripetal,} \quad c_1(x) = x_4(2x_3 + x_4)\sin(x_2) + 2g\sin(x_1) + g\sin(x_1 + x_2)$$

$$\text{and gravitational forces :} \quad c_2(x) = -x_3^2\sin(x_2) + g\sin(x_1 + x_2)$$

$$\text{passive dynamics :} \quad a(x) = \begin{bmatrix} I_{2\times 2} & 0 \\ 0 & M^{-1}(x) \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \\ c_1(x) - \zeta x_3 \\ c_2(x) - \zeta x_4 \end{bmatrix}$$

$$\text{control gain :} \quad B(x) = \begin{bmatrix} I_{2\times 2} & 0 \\ 0 & M^{-1}(x) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\text{complete system dynamics :} \quad \dot{x} = a(x) + B(x)u,$$

A. Linearizing the acrobot dynamic system:

After formulating the whole system with sympy, I linearized the system around $x_0 = (0, 0, 0, 0)^T$ by calculating $A_0 = \frac{\partial \dot{x}}{\partial x} \quad B_0 = \frac{\partial \dot{x}}{\partial u}$ using *sympy jacobian* function for partial derivative to generate the linear approximation as $\dot{x} = A_0\, x + B_0\, u$.

B. Compute a quadratic stage cost approximation

Again, using *sympy hessian* function to calculate Q in the form of $g_0(x,u) = \frac{1}{2}\, x^T Q x + \frac{r}{2} u^2$. Basically, for given stage cost equation, where r determines the relative importance of conserving control energy versus staying upright and k is the width of the position cost:

$$g(x,u) = 1 - e^{k\cos x_1 + k\cos x_2 - 2k} + \frac{r}{2}u^2$$

I can approximate it in quadratic form at x_0 by

$g_0(x,u) = g(x_0,u) + \left(\frac{\partial g}{\partial x}\right)^T (x - x_0) + \frac{1}{2}\left(\frac{\partial^2 g}{\partial^2 x}\right)(x - x_0)^2$, where $g(x_0,u) = \frac{r}{2}u^2$, $\left(\frac{\partial g}{\partial x}\right)^T (x - x_0) = 0$, and $\frac{1}{2}\left(\frac{\partial^2 g}{\partial^2 x}\right)(x - x_0)^2 = \frac{1}{2}\, x^T \left(\frac{\partial^2 g}{\partial^2 x}\right) x = \frac{1}{2}\, x^T Q x$.

C. Solving linear quadratic regulator problem

After formulating a linear system and a quadratic cost function, I can solve this continuous-time linear quadratic regulation(LQR) problem via the continuous Riccati equation.

$$A_0^T\, M(t) + M(t)A - M(t)B_0\, r^{-1} B_0^T\, M(t) + Q = 0$$

Therefore, we can achieve the control by $u = -r^{-1} B_0^T\, M(t)\, x(t)$. Therefore, by achieving this control policy, I can check the balancing performance of this linear acrobot model. To do this, I directly use $\dot{x} = A_0\, x + B_0\, u$ with calculated u, and use *scipy.integrate.odeint* to find this ode solution by integral.

D. Test the policy in nonlinear system

Similar to the linear system, but now I plug calculated control policy into $\dot{x} = a(x) + B(x)u$, and with *scipy.integrate.odeint* to find the ode solution x. In this way, I can test the performance of this LQR approximation for original dynamic system.

2) Reinforcement Learning of Hill Automobile Problem:

Here, I am trying to use model-free method to solve this hill automobile problem. In general, not matter it is model-free or model-based problem, we are all trying to solve MDP problem. Therefore, the first thing to do is to formulate this mountain car problem by MDP. Then, I use either Monte Carlo method or Temporal Difference method along with ϵ – greedy strategy to find an optimal policy to achieve the goal.

A. Formulating MDP problem:

$$v_{t+1} = v_t + \left(g\, m \cos(3x_t) + \frac{u_t}{m} - k\, v_t\right)\tau$$

$$x_{t+1} = x_t + v_{t+1}\tau.$$

In general, I am assuming I don't know the above model, where g = 9.8 m/s^2 is for gravity, and m = 0.2kg is for mass. For Monte Carlo method, I discretize the velocity space [-1.5, 1.5] m/s and position space [-1.2, 0.5] m each with 24 bins. For Temporal difference, I discretize the velocity [-1.5, 1.5] m/s and position [-1.2, 0.5] m each with 12 bins. What's more, I discretize the control space into [-0.2, 0, 0.2] for both methods. In the processing, I use index to represent the discretized value by applying a python numpy function digitize, and form two matrix with size (N_bins, N_bins, 3) for state action Q function, and size (N_bins, N_bins) for policy matrix. In general, I am trying to make these to matrix converge at the end of iteration. Additionally, the terminal cost for reaching the goal is -1, and there is a stage cost of 0.01 at each time step (we want to get there fast!). Therefore, I am trying to minimize the long-term cost to achieve the goal.

B. Implementing Monte Carlo Method

To implement this Monte Carlo method, I first have to generate a whole episode for each iteration with random initial state s_0 and a (s_0). The way to generate this episode is using the given policy matrix with the hill automobile update method. Additionally, I set the initial policy matrix randomly, Q matrix all for zero, and 100 update steps in each episode. if the car arriving goal before 100 steps, I directly stop the generating and process that current episode. Moreover, I initialize a visiting counter matrix used to update α, where α is $\frac{1}{\text{accumulate visiting times for a state +1}}$. After having an episode, I have to use it to update my Q matrix using first-visit method, which means I only calculate long term cost Gs_a, when I first meet that state and that control. Also, Gs_a = $\sum_i \gamma^i cost(s,a)$ where i is the index of the state 's' in that episode. Therefore, Q (s, a) can be update as Q (s, a) = Q (s, a) + α $(G - Q(s,a))$. The second thing here is to use the updated Q matrix to update the policy matrix for each state 's' in the episode by using $a(x) = argmin_a\, Q(x,a)$ along with ϵ – greedy strategy which I will talk later. Following is the algorithm I used in this part.

First-visit MC Policy Iteration with ϵ-Greedy Improvement

Algorithm 2 First-visit MC Policy Iteration with ϵ-Greedy Improvement

Init: $Q(x,u)$, $\pi(u|x)$ (ϵ-soft policy) for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$
loop
  Generate an episode $\rho := x_0, u_0, x_1, u_1, \ldots, x_{T-1}, u_{t-1}, x_T$ from π
  for each x, u in ρ **do**
    $G \leftarrow$ return following the first occurrence of x, u
    $Q(x,u) \leftarrow Q(x,u) + \alpha\,(G - Q(x,u))$
  for each x in ρ **do**
    $u^* \leftarrow \arg\max_u Q(x,u)$
    $\pi(u|x) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}(x)|} & \text{if } u = u^* \\ \frac{\epsilon}{|\mathcal{U}(x)|} & \text{if } u \neq u^* \end{cases}$

C. Implementing Temporal Difference Method

Another way to solve this mountain car problem is by using Temporal Difference method. The main difference between MC method and TD method is that TD method is an online method working with incomplete episode, and TD converges faster than MC method.

The main procedure is similar. For each iteration of an episode, I update 100 steps. However, I don't have to generate the whole episode at the begin of each episode. Instead, I update my Q matrix by the online method using incomplete episode of every St, At, Rt, St+1, At+1.

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha\left[g(x_t, u_t) + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t)\right]$$

, where $\alpha = 0.001$ for learning rate, $\gamma = 0.999$ for discount factor.

After geting the update Q value for this state and this action, I can update the policy at this state based on update Q value using ϵ – greedy strategy. Following is the algorithm I used in this part.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Figure 6.9: Sarsa: An on-policy TD control algorithm.

D. ϵ – Greedy Strategy

This is the alternative to exploring starts, this method can ensure continual exploration which means it can encounter all the actions for a specific state 's' with non-zero probability. It is a stochastic policy that picks the best control according to Q (s, a) in the policy improvement step and ensure all other controls are selected with a small (non-zero) probability.

$$\pi(u \mid x) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}(x)|} & \text{if } u = \underset{u' \in \mathcal{U}(x)}{\arg\max}\, Q(x, u') \\ \frac{\epsilon}{|\mathcal{U}(x)|} & \text{otherwise} \end{cases}$$

Results and Discussion

1) Balancing the Acrobot Using Linear Quadratic Control:

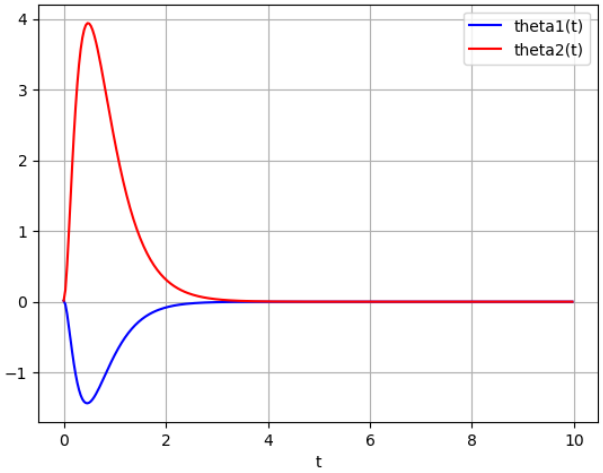

Figure1: linearized model

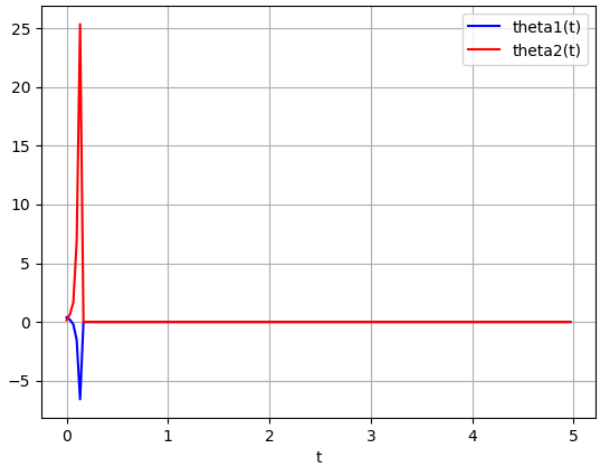

Figure2: non-linearized model

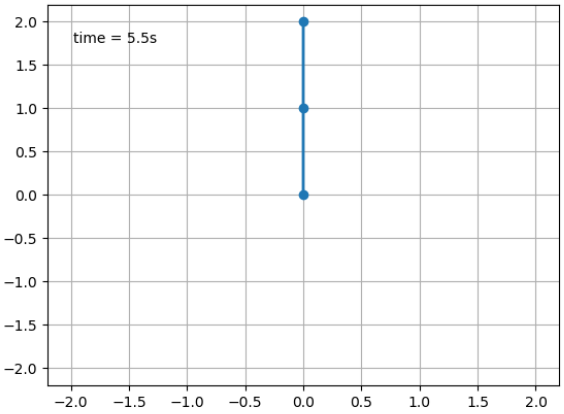

Figure3: linearized standing

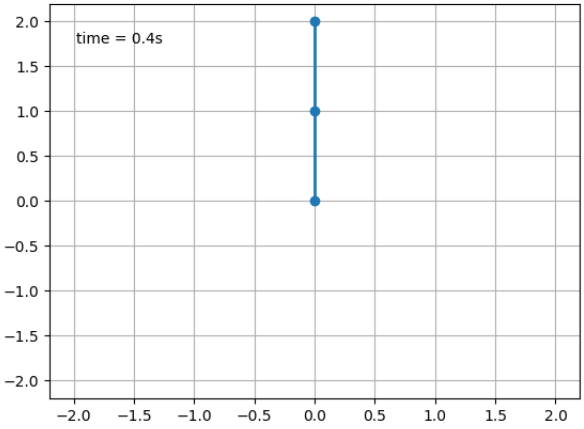

Figure4: non-linearized standing

The above 4 graphs show result of the linear model and nonlinear for initial point (0.01511932, 0.01609094, 0.11496859, 0.08131474) and (0.39118467, 0.15678339, -0.63083821, 0.53322947) respectively. For this acrobot system, I choose r = 1, and k =1 to achieve good performance. In general, both nonlinear model and linearized model perform well with the initial state near the linearizing point $x_0 = (0, 0, 0, 0)^T$. In general, by moving the initial state away from x_0 the linearized system can still balance inverted, and the non-linearized system can never go back inverted when initial state is too far away. In my opinion, this linearized system is globally controllable, and the non-linearized system is locally controllable. However, for my project I did not find that threshold point for nonlinear case. Also, as shown in figure1 and figure2, the nonlinear system goes back to stable is much faster than linear model with the LQR policy. Therefore, LQR method is a good approximation.

2) Reinforcement Learning of Hill Automobile Problem

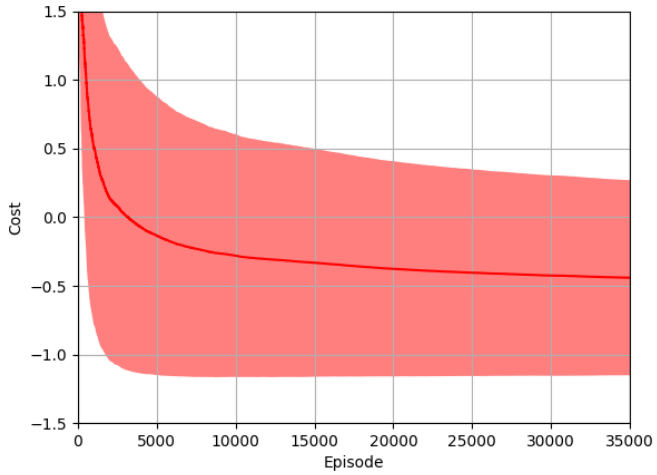

Figure5: MC method cost

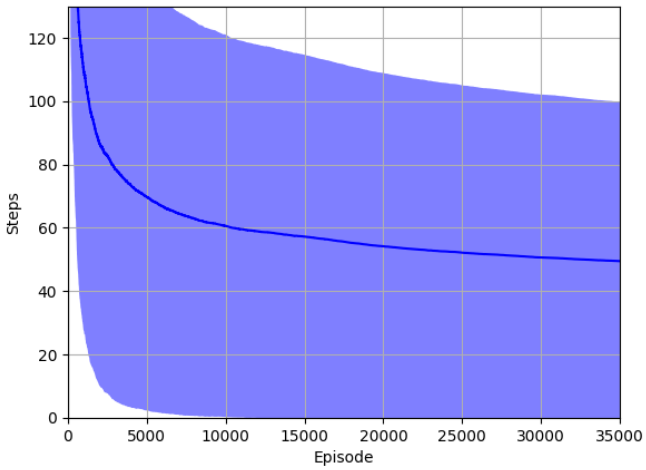

Figure6: MC method step

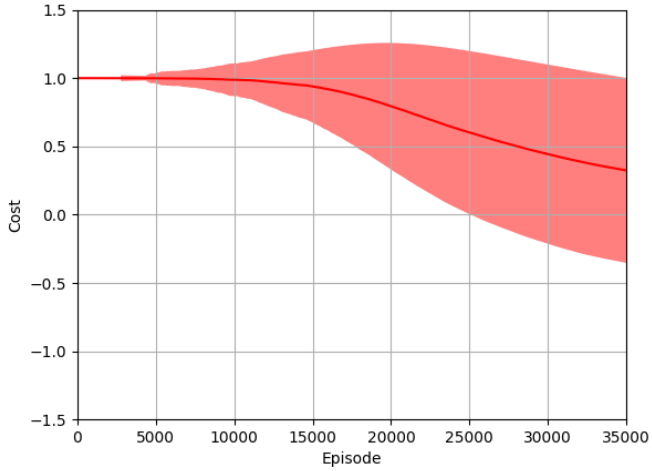

Figure7: TD method cost

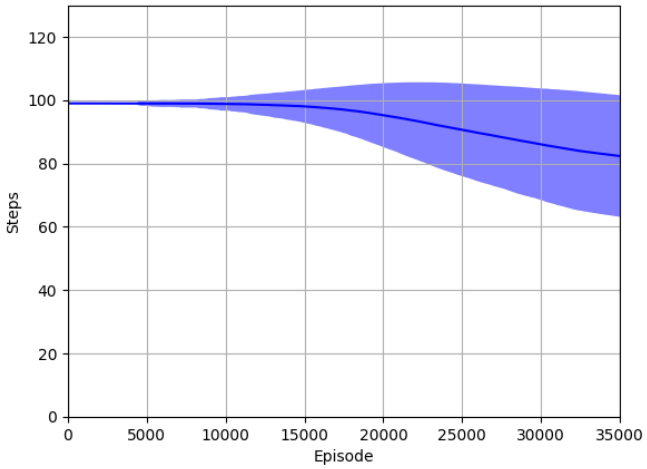

Figure8: TD method step

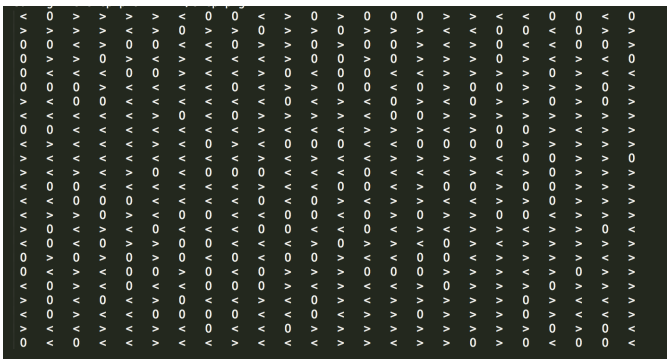

Figure 9: MC policy matrix

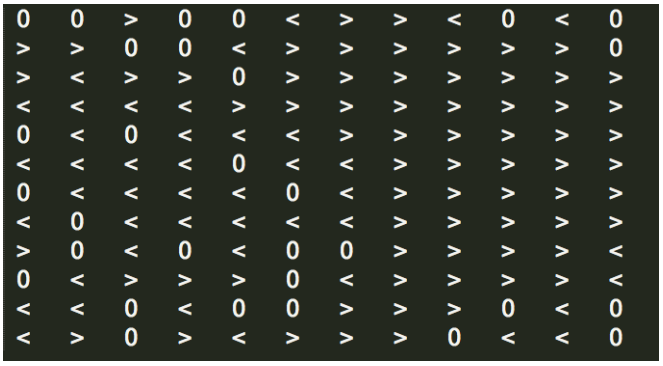

Figure 10: TD policy matrix

The above 6 graphs, figure5 to figure 10, show the results of Monte Carlo method and Temporal Difference method. For the MC method, I set discretizing specs number of actions =3, number of bins for state of each

dimension = 24, and epsilon for greedy strategy = 0.3, discount factor for long term cost = 0.995, alpha is a stochastic value, total episode = 35000, and 200 update steps for each episode.

For the TD method, I set discretizing specs number of actions =3, number of bins for state of each dimension = 12, and epsilon for greedy strategy = 0.1, discount factor for long term cost = 0.9999, learning rate alpha = 0.001, total episode = 35000, and 100 update steps for each episode. Also, the arriving time for MC is 6.1s, and TD is 5.4s for above graphs.

In general, the TD method converges faster than MC method with lower variance but with high bias, and MC is verse vice. Therefore, for MC method, it sometimes takes much long time to arrive the goal, and sometimes takes much fast time to arrive the goal as high variance for MC method as shown in figure 6, and it is the reason I set my update steps for MC as 200. For the TD method, the arriving time do not have much variation, but it has high bias, so the mean arriving time for TD is large than MC. The reasoning for cost comparison is similar to the step for TD and MC.

The figure 9 and figure10 show the visualization of MC and TD method respectively. The column represents the velocity space, and row represents position space. ‘<’ represents -0.2 control, ‘0’ represent 0 control, and ‘>’ represents 0.2 control. In general, the two policy matrixes should be similar, when both are near converging, even if MC method I have denser discretizing than TD method.