

1. In this exercise, we will apply value iteration and policy iteration to solve the fixed maze example (maze-sample-10x10-v0) provided by the TA.

(a) Define the dynamic system model.

Let the overall cost function, per-time step cost function and the state function denoted with J , L and f , respectively. Therefore, optimal control policy can be found by solving the below equation Where x_k and u_k are system state and taken control, respectively at time k :

$$J(0, x_0, u) = \mathbb{E} \left[\sum_{k=0}^{\infty} L(x_k, u_k, b_k) \middle| x_0 \right] \quad (1)$$

s.t. $x_{k+1} = f(x_k, u_k, b_k)$

The dynamic system (i.e. the maze) is modeled as a 2D $n \times n$ matrix where a single object, called an agent, moves in it. The position of the agent (denoted with p) is its Cartesian coordinate and the maze's barriers location (denoted with b) is detriment using two tuples each represent a Cartesian coordinate. The boundaries of the maze are also modeled as barriers. Fig. 1 shows an example of p and b values where $p = (0, 0)$ and the vertical barrier $b = ((1, 0), (1, 1))$. Therefore, $x_k = p_k$. There are only four possible values for u_k , up, down, right and left where it is mathematically formulated as $u_k \in \{(-1, 0), (1, 0), (0, 1), (0, -1)\}$, however if the agent faces a barrier it will stay in its state. The per-step cost function is defined in eq. 2.


 (0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

Figure 1: A sample of a two by three maze indicating the agent, barriers and each cell Cartesian coordinate.

$$L(x_k, u_k) = a + ch_k \quad (2)$$

a and c are tunable parameters to control the importance of each term where in this HW they were set to -1 and -0.5, respectively. The first term tries to minimize number of taken steps by the agent till it reaches the final destination. The second term tries to minimize

number of times the agent hits the walls. The terms are mathematically further formulated as $a, c \in \mathbb{R}$ where $a, c \in [-1, 0]$ and $h_k \in \{0, 1\}$. The state equation is defined in 3.

$$f(x_k, u_k, b_k) = \begin{cases} x_k, & \text{both } x_k, u_k \in b_k \\ x_k + u_k, & \text{otherwise} \end{cases} \quad (3)$$

2. **Use value iteration to find a solution. Start with the initial value $V_o = 0$.**

The algorithm of value and policy iteration was adapted from Chapter 4 of the book 'Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.'. The code follows the algorithm shown in Fig. 2. Although, the ECE372 course's slides try to minimize the cost function and Fig. 2 tries to maximize, both are optimizing the control policy but with different cost functions. A positive cost function is defined in the slides but a negative one defined in the book, hence, a minimization should be applied in the former and a maximization is needed for the latter. **A negative cost function is defined in this HW and r shown in Fig. 2 is defined as $r = L(x_k, u_k)$.** The

```

Initialize array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in \mathcal{S}^+$ )

Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)

Output a deterministic policy,  $\pi$ , such that
 $\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$ 

```

Figure 2: Value iteration algorithm taken from chapter 4 of Sutton & Barto.

algorithm was coded using python and the value function converged after 64 iterations. The value function generated after each iteration was used to develop a policy the agent would follow to start with initial state until the final stage with a $cost = \sum_{x_k \notin s_{destination}} L(x_k, u_k)$. The per iteration cost is shown in Fig. 3.

Some value functions produce an extremely bad policy yielding very high costs, hence, they were truncated. Indeed the plot shows that the value function converges to the optimal one where that can be clearly seen starting at iteration 55.

3. **Use policy iteration to find a solution. Begin with the initial policy $\pi_o = 0$.**

Similar to part 2, the policy iteration was found but using the algorithm shown in Fig. 4. $\pi_o = 0$ was encoded as going north; i.e. $(\pi_o(s) = go\ north \forall s \in \mathcal{S})$.

The algorithm was coded using python and the policy function converged after 39 iterations. Policy iteration needed less number of iterations, however, its computation complexity is about $10 \times$ more than value iteration. Nevertheless, policy iteration provides a great

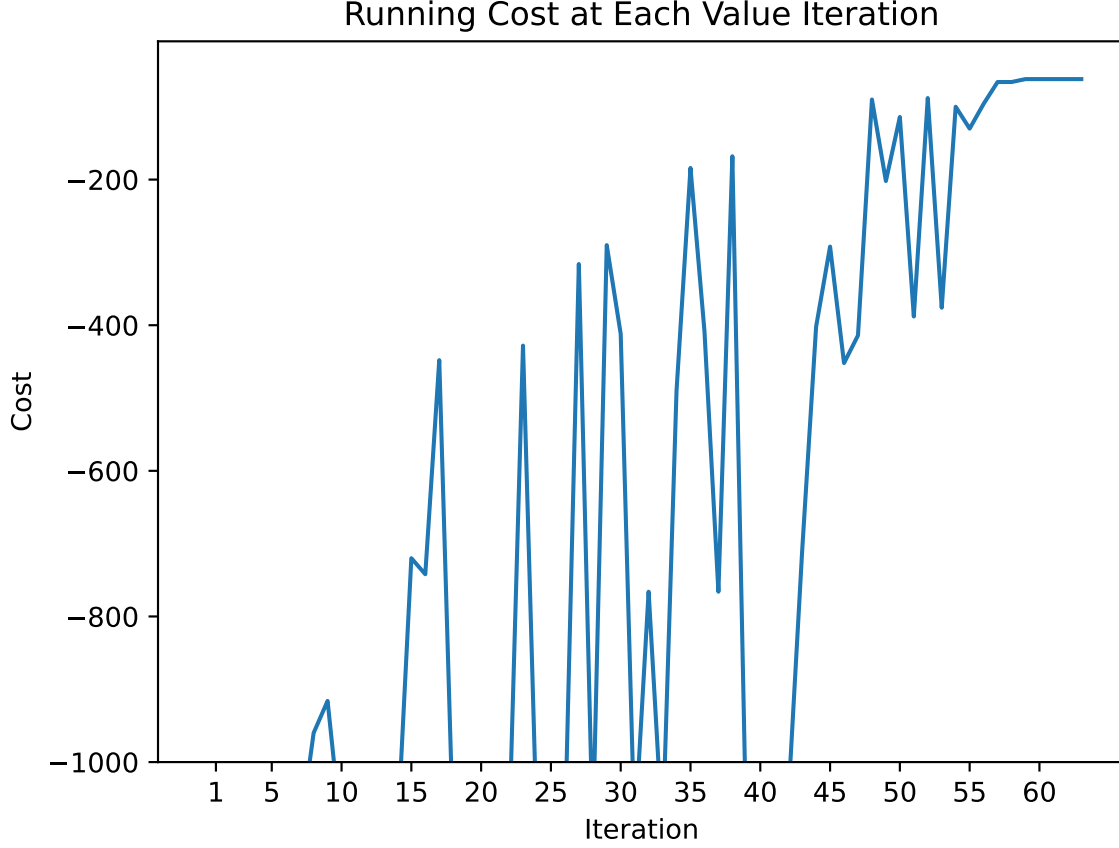


Figure 3: Plot of the cost function when the agent uses the policy found from per iteration value function.

convenience that the system designed do not need to define neither a value function nor a policy for the control system; rather policy iteration will find both!

The policy function generated after each iteration was used to control the agent from the initial state until the final stage with a $cost = \sum_{x_k \notin s_{destination}} L(x_k, u_k)$. The per iteration cost is shown in Fig. 5.

Some policy functions produce an extremely bad policy yielding very high costs, hence, they were truncated. Indeed the plot shows that the policy function converges to the optimal one where that can be clearly seen starting at iteration 33.

4. **Now, repeat (b) and (c) with different initial values V_o and π_o . Suppose we know the optimal path for the maze problem. Can you devise a way to embed this path into V_o and π_o ?**

One way to embed the optimal solution into V_o and π_o is to modify both of them according to the optimal path. $V_o(s) = \min value \forall s \notin \mathcal{S}_{opt}$, $V_o(s_o) = \min value + \zeta$ and $V_o(s_{destination}) = 0$. All other values of $V_o(s)$ for $s \in \mathcal{S}_{opt}$ are greater than $\min value$ and less than 0 in an increasing order. V_o and π_o are shown for the 10x10 maze in Figs. 6 and 7. The plots shows a clear convergence that is much much faster the convergence when both v_o and π_o were equal to zero. The convergence occurred much much faster and the plots

```

1. Initialization
    $V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ 

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$ 
     For each  $s \in \mathcal{S}$ :
        $v \leftarrow V(s)$ 
        $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$ 
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
   until  $\Delta < \theta$  (a small positive number)

3. Policy Improvement
   policy-stable  $\leftarrow$  true
   For each  $s \in \mathcal{S}$ :
      $a \leftarrow \pi(s)$ 
      $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$ 
     If  $a \neq \pi(s)$ , then policy-stable  $\leftarrow$  false
   If policy-stable, then stop and return  $V$  and  $\pi$ ; else go to 2

```

Figure 4: Policy iteration algorithm taken from chapter 4 of Sutton & Barto.

are shown in Figs. 8 and 9.

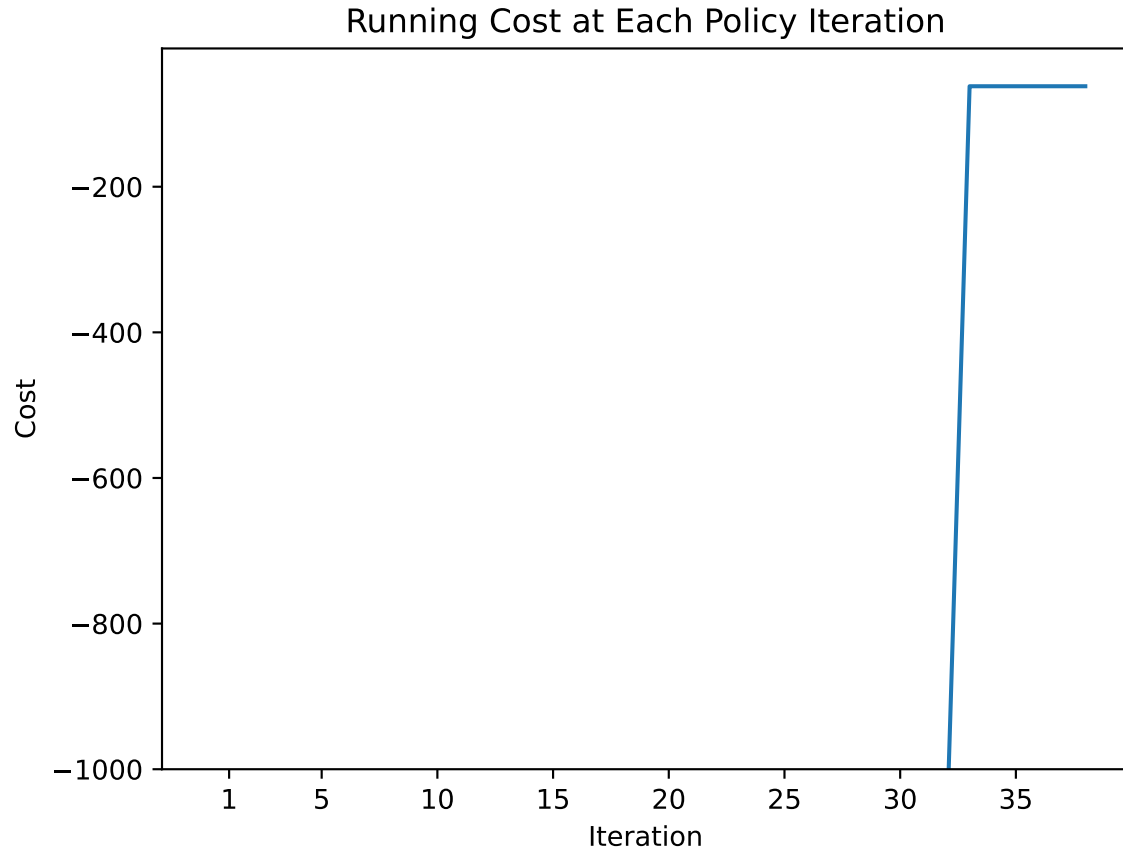


Figure 5: Plot of the cost function when the agent uses the policy found from per iteration policy function.

```
[[-62. -61. -100. -100. -28. -27. -26. -25. -24. -100.]
 [-59. -60. -100. -100. -29. -30. -31. -32. -23. -22.]
 [-58. -100. -100. -100. -100. -35. -34. -33. -20. -21.]
 [-57. -100. -39. -38. -37. -36. -100. -100. -19. -18.]
 [-56. -100. -40. -41. -100. -100. -14. -15. -16. -17.]
 [-55. -100. -100. -42. -43. -100. -13. -100. -100. -100.]
 [-54. -53. -52. -51. -44. -100. -12. -7. -6. -5.]
 [-100. -100. -100. -50. -45. -46. -11. -8. -3. -4.]
 [-100. -100. -100. -49. -48. -47. -10. -9. -2. -100.]
 [-100. -100. -100. -100. -100. -100. -100. -100. -1. 0.]]
```

Figure 6: A representation for V_o if optimal path was known.

```

[[2. 1. 0. 0. 2. 2. 2. 2. 1. 0.]
 [1. 3. 0. 0. 0. 3. 3. 3. 2. 1.]
 [1. 0. 0. 0. 0. 2. 2. 0. 1. 3.]
 [1. 0. 2. 2. 2. 0. 0. 0. 2. 1.]
 [1. 0. 0. 3. 0. 0. 1. 3. 3. 3.]
 [1. 0. 0. 0. 3. 0. 1. 0. 0. 0.]
 [2. 2. 2. 1. 0. 0. 1. 2. 2. 1.]
 [0. 0. 0. 1. 0. 3. 1. 0. 1. 3.]
 [0. 0. 0. 2. 2. 0. 2. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 2. 0.]]

```

Figure 7: A representation for π_o if optimal path was known.

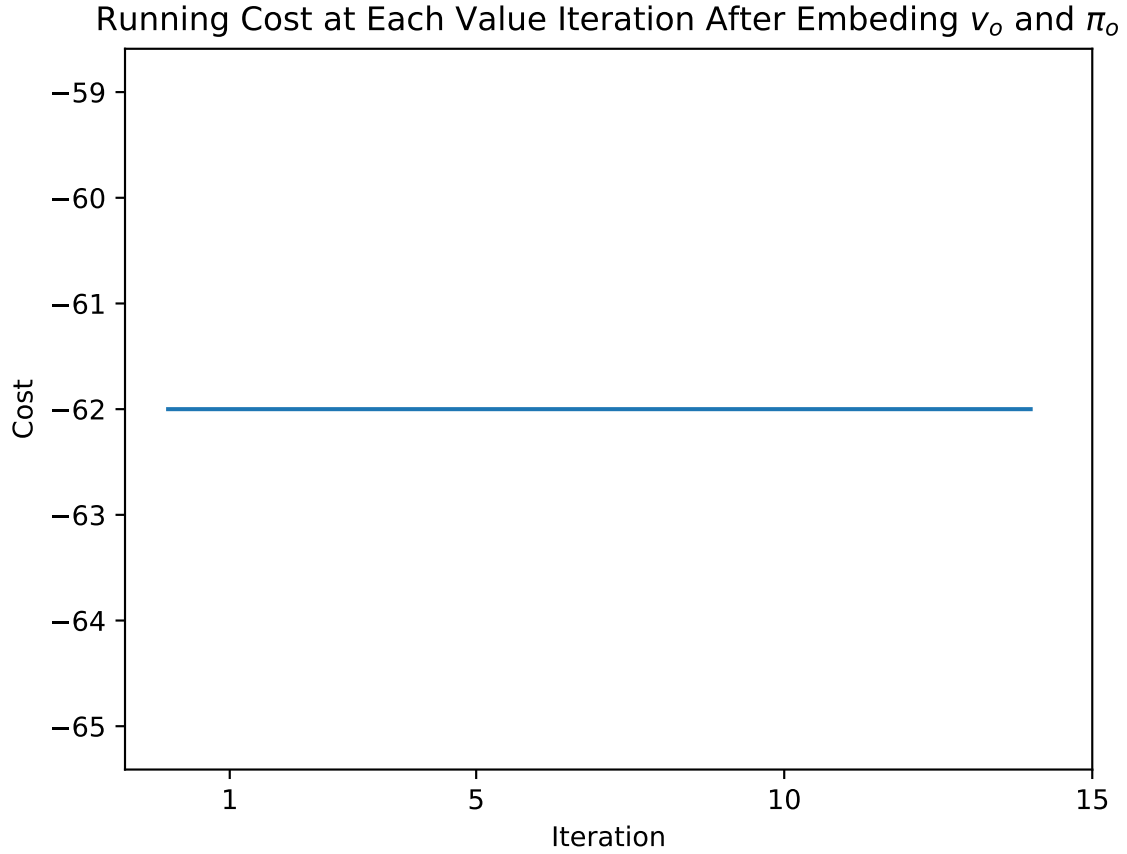


Figure 8: Plot of the cost function when the agent uses the policy found from per iteration value function after embedding the optimal path.

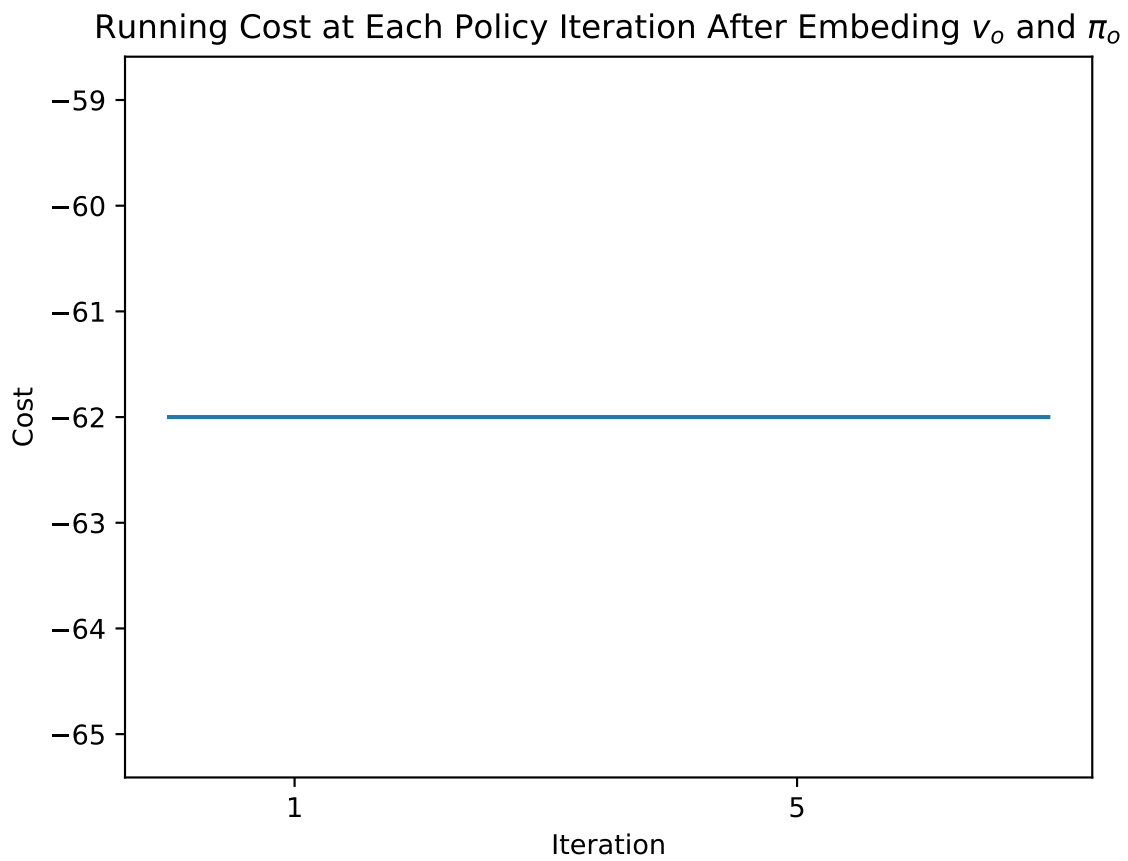


Figure 9: Plot of the cost function when the agent uses the policy found from per iteration policy function after embedding the optimal path.