EL2805 Reinforcement Learning

Computer Lab 1

Mohammed Akif (990123-4493)
Kevin Armbruster (930519-T711)

November 11, 2022

1 Maze and Minotaur

1.1 MDP - Simultaneous movement

The state space S is modeled to be the set of all possible positions of the player (x, y) and the Minotaur (yM, xM). The set of possible positions is defined to be the coordinates inside the maze boundaries. Since the Minotaur can walk inside the walls, we do exclude these coordinates only for the player position.

(i) $S = \{((y, x), (yM, xM))\}$, where maze(y,x) != wall. Similarly, maze(y,x) and maze(yM,xM) need to be defined, i.e. inside the maze.

The action space A is the union of actions A_s available to the player in each state s:

(i) $A = \{stay, up, down, left, right\}$

Additionally, to allow a concise further description the sets of A_{valid} and $A_{invalid}$ are introduced, which would result in a player position that respectively is or is not defined in the set of states S.

The transition probabilities P derive from the deterministic actions of the player, the i.i.d. action-selection-behavior of the Minotaur and the game rules:

(i) $P(s'|s, a \in A_{valid}) = \frac{1}{|MA|}$, where |MA| denotes the size of the set of possible actions by the Minotaur. The positions of both player and Minotaur are updated accordingly.

(ii) $P(s' = (y, x, yM', xM')|s = (y, x, yM, xM), a \in A_{invalid}) = \frac{1}{|MA|}$. The players position remains unchanged, if the player takes any action which would result in a position outside the maze or inside a wall.

(iii) Undefined transitions have probability 0.

The reward R is stationary, only dependent on the state, to achieve the objective of maximizing the probability of reaching the goal B.

(i) $R = -\infty$, if the player position and Minotaur position coincide.

(ii) $R = 0$, if the player reaches the goal.

(iii) $R = -1$, for each valid action the player takes in the maze.

(iv) $R = -100$, for each invalid action the player takes in the maze.

1.2 MDP - Alternating movement

We extend **the state space** from our previous model to $S = \{(b, (y, x), (yM, xM))\}$ by a binary turn variable b, which indicates that if $b = True$ the player can move, otherwise the Minotaur.

The action space stays the same, but for all states with the turn variable $b = False$, the player can only choose the action stay, and vice versa for the Minotaur.

The transition probabilities have to be extended accordingly for the new state space:

(i) $P(!b|b, a \in A) = 1$, the turn variable flips with certainty after each transition.

(ii) Otherwise the same transition rules apply.

This definition would increase the action space of the Minotaur, which might not be possible. In this case the transition probabilities are adapted to fail with certainty, if either the player or Minotaur is not in turn, which basically corresponds to the stay action.

With alternating movement, it becomes more likely for the Minotaur to catch the player, because the player can no longer step through the Minotaur, because the Minotaur will not leave the cell. Especially in hallways, this blocks the player from slipping through in direction of the goal and forces him to dodge and circumvent the Minotaur. Similarly, the player cannot take a step that would lead to an adjacent cell to the Minotaur, because the player would die w.p. $\frac{1}{|MA|}$, where $|MA|$ denotes the number of possible actions the Minotaur can take.

1.3 Dynamic Programming

The figures illustrate the change in policy near a Minotaur. In the case where the Minotaur cannot stay, the agent walks on his spot, since it is safe, while it becomes unsafe in case the Minotaur can stay. It is a similar setup to alternating movement.

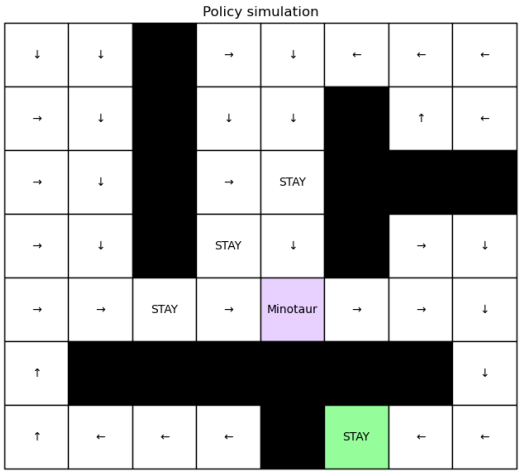

Figure 1: Policy of Dynamic Programming with Minotaur NOT capable of standing still

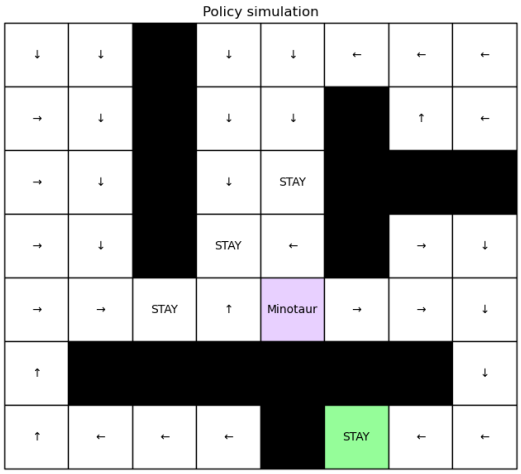

Figure 2: Policy of Dynamic Programming with Minotaur capable of standing still

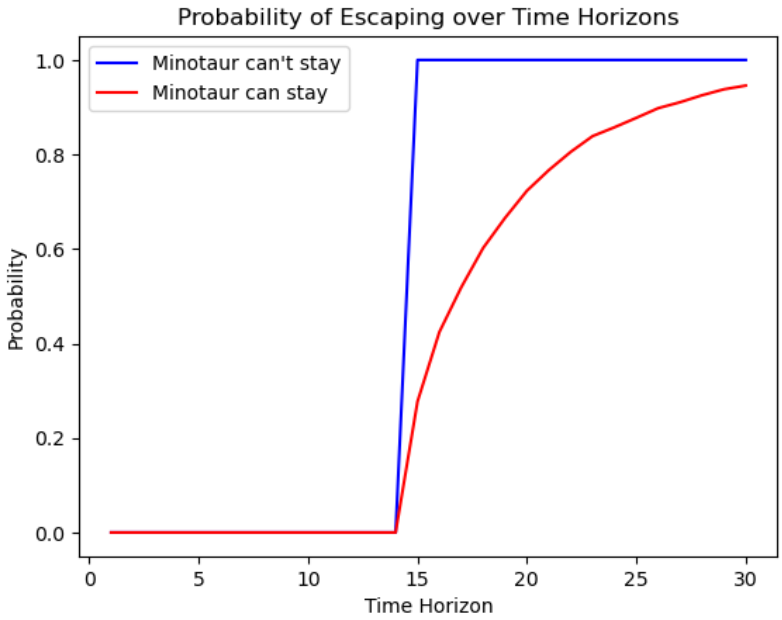

Figure 3: Probability comparison

1.4 Value Iteration

To model the geometrically distributed life with mean 30 we used a discount factor and solved it using the Value Iteration method. One of the possible interpretations of the discount factor was the random time horizon which was geometrically distributed. Adapting the formula from the lecture yields:
$\mathbb{E}[T] = 1/(1-\lambda)$, with $\mathbb{E}[T] = 30$ from the exercise description.
$\lambda = \frac{29}{30}$

The resulting visualizations of the policies look identical to the same DP case. The probability described in f) is $P(alive) = 0.75$.

1.5 Theoretical questions

1.5.1 On- vs Off-policy learning

1. Off-policy learning collects experiences under an independent behavior policy.
2. On-policy learning collects experiences under the policy that we want to improve.

In both cases, the gathered data is then used to learn the optimal policy, i.e. improve the current policy.

1.5.2 Convergence conditions for Q-learning and SARSA

1. The step size has to satisfy $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$
   This is fulfilled if $\alpha_t = \frac{1}{t+1}$
2. The policy in off-policy learning has to explore every (state, action) pair infinitely often.
   In off-policy learning, this has to be ensured by the behavior policy in producing a irreducible Markov chain.
   In on-policy learning ϵ-soft policies are applied, that use uniform random actions w.p. ϵ, else $a = \arg\max_{a \in A_{s_t}} Q(s_t, a)$ w.p. 1-ϵ.

1.5.3 MDP with life and keys

We model it as an infinite MDP with discount factor $\lambda = 49/50$ due to the geometrically distributed life with mean 50, similar to e).

The state space S is extended by another boolean variable for the possession of the key:

(i) $S = \{((y, x), (yM, xM), k)\}$.

The action space A formulation does not change. I assume that the keys are picked up my moving over the respective cell, based on the formulation of the task.

The transition probabilities P derive from the deterministic actions of the player, the i.i.d. action-selection-behavior of the Minotaur and the game rules:

(i) $P(s'|s, a \in A_{valid}) = 0.65 * \frac{1}{|MA|}$, where |MA| denotes the size of the set of possible actions by the Minotaur. The positions of both player and Minotaur are updated accordingly.

(ii) $P(s' = (y, x, yM', xM')|s = (y, x, yM, xM), a \in A_{invalid}) = 0.65 * \frac{1}{|MA|}$. The players position remains unchanged, if the player takes any action which would result in a position outside the maze or inside a wall.

(iii) $P(s' = (y', x', yM', xM')|s = (y, x, yM, xM), a \in A) = 0.35 * 1$, where $|(y, x) - (yM', xM')| < |(y, x) - (yM, xM)|$, i.e. the Minotaur moves into the direction of the player position.

(iv) Undefined transitions have probability 0.

The reward R formulation does not change.
That is what we wrote before trying the bonus exercises, but during them we came to the conclusion that we had to change the formulation to the following:

(i) $R = 0$, if the agent reaches the goal or picks up the key.

(ii) $R = -1$, if the agent takes a valid step.

(iii) $R = -2$, if the agent stayed still.

(iv) $R = -100$, if the agent takes an invalid step.

(v) $R = -\infty$, if the agent dies.

We introduced the reward for the key to split the problem into two subproblems and the agent can find the path to the key more reliably in order to focus searching the goal afterwards. This helps due to the fact that the agent only starts propagating the "good" value back from the end, when he reaches a goal that gives such a desirable reward. While it works without this setup, the convergence time decreases notably.

1.6 Episodic online Q-learning

Algorithm 1: Q-learning

Input : Initial state s, Number of episodes N, Discount factor λ, Step size exponent α, Exploration probability ϵ (, Exploration decay δ)

Output: policy π

Initialize $Q^{(0)}$ with zeros
Initialize $N^{(0)}$ with zeros
for *Episodes k = 1,2,...,N* **do**
    $s \leftarrow$ Initial state
    while *last state != current state (alt.: Agent alive and Agent in maze)* **do**
        Action from policy: $a \leftarrow \epsilon$-greedy policy given state s
        Observe: Next state s', Reward $r \leftarrow$ simulate step given (s, a)
        Count visit: $N(s, a) += 1$
        Calculate step size: $c \leftarrow 1/N(s, a)^{\alpha}$
        Update current Q according to TD error using best action in next state:
        $Q(s, a) = Q(s, a) + c * (r + \lambda * \max_b(Q(s', b)) - Q(s, a))$
        $s \leftarrow s'$
    end
end
return $\arg\max_a(Q)$

In the following figures we see the convergence of the initial state value under varying exploration (4) and step size exponent (5). Sometimes the figures showed a small bent going upwards again, other times they looked like depicted. We think that happens if the agent finds the goal and propagates the value back to the initial state and depending on the rewards, it starts to rise again (Goal Reward > 0). In 4 we see that a too high exploration leads to a later convergence in this case. In 5 we see that a bigger step size leads to an earlier convergence. In both cases there is a trade-off, while exploration is necessary for us to find the goal path in the beginning, a too high exploration leads to worse results. The same applies for the step size. In figure 5 we see that the faster decaying learning rate needs more steps to actually find the goal and bounce back gaining value as it is propagated through the states. Additionally, one could experiment with a decaying exploration probability similar to the one used in SARSA.

We found that under specific configurations the agent learns to go towards the Minotaur in order to end the game as early as possible to avoid collecting negative rewards. Similarly, the agent might choose to not learn to reach the goal but only stay alive inside the maze. These can be fixed with fine tuning the number of episodes, rewards, and the previously discussed exploration and step size. Especially these behaviors led to the aforementioned adaptations in our reward definitions.

I think that a proper initialization of the Q-values that signifies where the correct path could be and the agent does not have to search from scratch tremendously speeds up the convergence. It would help the agent finding the key and goal faster, which is the only way for the agent to learn positive values for the path.

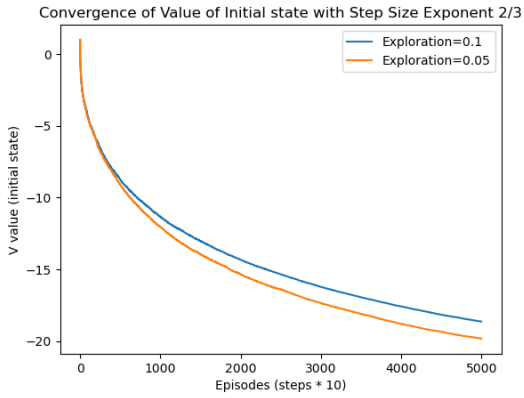

Figure 4: QLearning Convergence under variable Exploration. Episodes times 10.

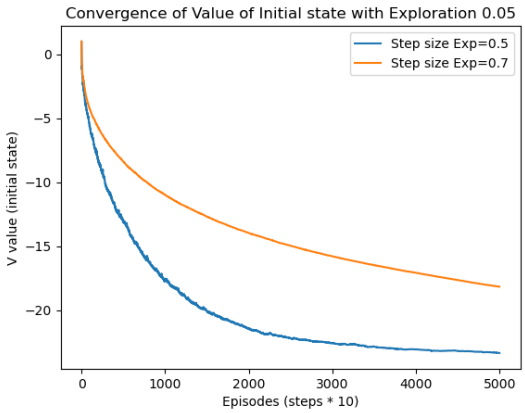

Figure 5: QLearning Convergence under variable Step Size Exponent. Episodes times 10.

1.7 SARSA

The differences evolving from the fact that the observation includes the next action a' leading to (s, a, r, s', a'):

(i) Sample the next action a' according to the ϵ-greedy policy for the next state s'

(ii) The target for the TD error is defined using the next action a' leading to $Q(s', a')$ (instead of the best action in the next state $\max_b(Q(s', b))$).

(iii) Transfer the action a' to the next iteration at the end $a \leftarrow a'$

In our case SARSA worked not as good as Q-learning. As you can see in both of the following images. For 6 we increased the episodes to 100'000 instead of 50'000 to increase the chance and time of finding the goal and propagating the value through the states. This variant did not find the goal path and according to the graph also has not converged yet (although we gave it double the amout of time). Only the SARSA variant with a slowly decaying exploration depicted in 7 with $\delta = 0.5$ worked reliably. While the other setups could sometimes find the goal path, it was not guaranteed.

So we could definitely identify an improvement in convergence for the case of Step size parameter α and Exploration decay parameter δ: $\alpha = 2/3 > \delta = 0.5$. Experiments with an inverse relationship resulted in the agent not finding the goal path. This is obvious from the fact that when the step size is decaying faster than the exploration rate, then the agent can no longer properly utilize and learn from the exploratory behavior. If the step size aka. learning rate vanishes we stop learning, disregarding the fact that we still explore.

(i) α = high $\rightarrow$ fast decaying / smaller step sizes

(ii) α = low $\rightarrow$ slow decaying / greater step sizes

(iii) δ = high $\rightarrow$ fast decaying / less exploration

(iv) δ = low $\rightarrow$ slow decaying / more exploration

Again, I think that the convergence speed would be affected by a good initialization of the Q values with the same reasoning as before, but due to the formulation of the TD error sampling the next action according to the ϵ-greedy policy it might be to a smaller extend as we do not choose the best action and therefore influence directly the learned step.

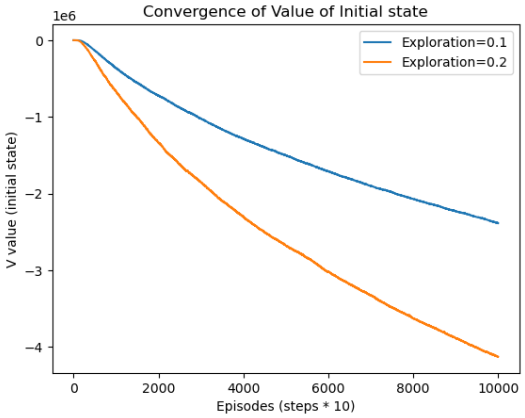

Figure 6: SARSA Convergence under variable Exploration. Episodes times 10. Note increased episodes.

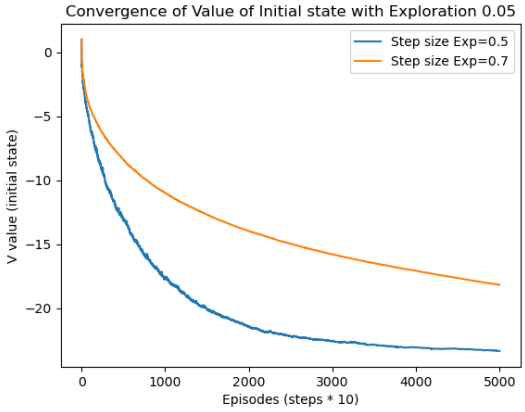

Figure 7: SARSA Convergence under decaying Exploration. Episodes times 10.

1.8 Probabilities

The escape probability of Q-learning with $\alpha = 0.5$ and Exploration probability $= 0.05$ is $91,5\%$. SARSA with $\alpha = 2/3$ and Exploration decay $\delta = 0.4$ is $61,3\%$.

The following Q-values for the initial state are quite evenly spread across all actions. This does also not change if we let the algorithm run 10 times more episodes. Therefore the probabilities we identified are not close to the V or Q-values. Our take is that since Q-learning is model-free, we do not learn any probabilities but rather try to identify actions in each state that let us achieve the goal. Since the action in the initial state of our MDP with geometrical distribution of mean 50 may not be very relevant the Q-values stayed rather equal. While the reward function is one of the most important tools to setup the problem definition in order to solve it, in our case we could not

find one where the Q-values relate to the survival probability of the agent.

Q-values of initial state:
SARSA: $[0.200, 0.200, 0.200, 0.200, 0.200]$
Q-learning: $[0.208, 0.199, 0.197, 0.199, 0.197]$

2 RL with linear function approximators

2.1 Training process

The following parameters were used in the training process:

- Training episodes = 200
- Discount factor, $\gamma = 1$
- Eligibility trace, $\lambda = 0.7$
- Learning rate, $\alpha_i = \frac{0.01}{||\eta_i||_2}$
- $\epsilon = 0.3$ as initial value, and is reduced by up to 90 percent during the training process. This is done over nine steps, where each step reduces the value of ϵ by 10%.
- Momentum parameter, $m = 0.1$
- Initial weights, $w \sim \mathcal{N}(0,\ 0.01)$
- Fourier basis, $\eta = [(0,1),(1,0),(1,1),(1,2),(2,0),(2,2)]$

During the training process the Sarsa(λ) algorithm is trying to approximate the Q-function of the MountainCar environment without knowing the system. The Fourier basis of order 2 is used to parametrize the Q-function, defined as the following

$$Q_w(s,a) = w_a^T \Phi(s)$$

$$\Phi_i(s) = cos(\pi \eta_i^T s), i \in \{1, ..., m\}$$

The Sarsa(λ) updates the eligibility trace for each action as well as the weights by using the observations $a_t, r_t, s_{t+1}, a_{t+1}$. The advantage of using eligibility traces is that the actions contributing the most to the total reward get updated correctly. Since the environment of our problem is sparse, it is important to use eligibility traces.
To balance between exploration and exploitation, the $\epsilon-$greedy method is used when selecting the

actions during the training process. Furthermore, the SGD with Nesterov Acceleration is used when calculating the update of the weights in order to reduce the oscillations.
For $\eta = [k_1, 0]$ or $[0, k_2]$ where k_1, k_2 are 1 or 2 the basis function only depend on one of the two states; position (s_1) or velocity (s_2), respectively. The value of the non-negative parameter determines the frequency of basis function along that dimension. We get a coupling between the two state variables when we have $\eta = [k_1, k_2]$. The direction of interaction is given by the ratio between k_1 and k_2.

2.2 Analysis of the policy and training process

$\eta = [0, 0]$ means we get $cos(0) = 1$ i.e. we get a constant function, and the Q-function becomes $Q_w = w_a^T$. The average total reward decreases when $\eta = [0, 0]$ is added because then the Q-function does not depend on the state anymore.

Since the reward in our problem is sparse, we benefit from choosing random initial Q-values. By doing so the agent is able to explore more in order to reach the goal.

We see in Figure 8 that both the episodic and average total reward increases over the episodes. We get an average total reward larger than -135 for the last 50 episodes.
From the plot in Figure 9 with large learning rate, we see that the average reward does not increase much over the episodes compared to the plot in Figure 8. Also, the algorithm converges to undesired value. One reason could be the larger update steps when calculating the weight. For small learning rate however, it takes longer time for the algorithm to learn how to reach the top of the hill, as seen in Figure 10. When the reward start increasing it keeps doing so and reaches the desired values unlike the reward with a high learning rate.

Figure 8: Episodic and average total reward across 200 episodes with $\alpha = 0.01$.

Figure 9: Episodic and average total reward across 200 episodes with $\alpha = 0.1$.

Figure 10: Episodic and average total reward across 200 episodes with $\alpha = 0.001$.

Low value for the eligibility trace result in the algorithm converging to a high reward (see Figure 11) over many episodes. While a high λ results in a less value for the total reward (see Figure 12), so we can conclude that a low value is better.

Figure 11: Episodic and average total reward across 200 episodes with $\lambda = 0.1$.

Figure 12: Episodic and average total reward across 200 episodes with $\lambda = 0.9$.

In Figure 13 we can see the optimal policy, and in Figure 14 we see the optimal value function. Both are plotted over the state space domain. It is clear from Figure 13 that for high velocities the agent takes the action *push right* and for low velocities the action *push left* is taken. Velocities around the middle results in the action *no push*, i.e. the position we are at is important. Similar behaviour can be concluded from Figure 14 where high velocities result in a high value for the Q-function, while low velocities leads to lower Q-values.

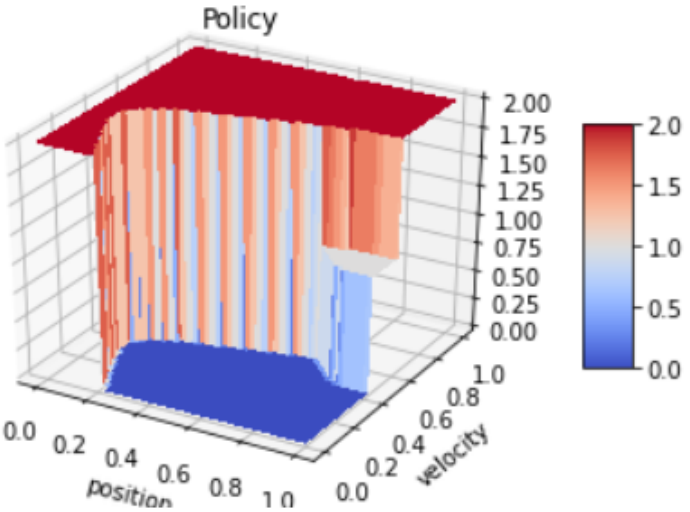

Figure 13: Optimal policy over the state space domain.

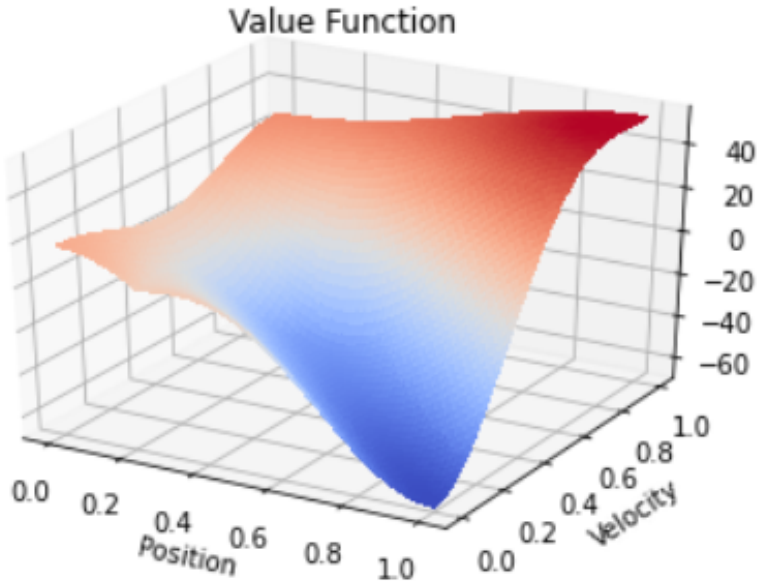

Figure 14: Value function of the optimal policy over the state space domain.