RL-2

Abhilash Jain

October 13, 2019

1 Sailor Grid World

1.1 What is the agent and the environment in this setup?

The Agent is the sailor who wants to reach the harbour.

The environment is the sea, the rocks, the harbour and the narrow passage between the rocks. Also the calmness of the sea and the heavy windy conditions are part of the environment

2 Value iteration

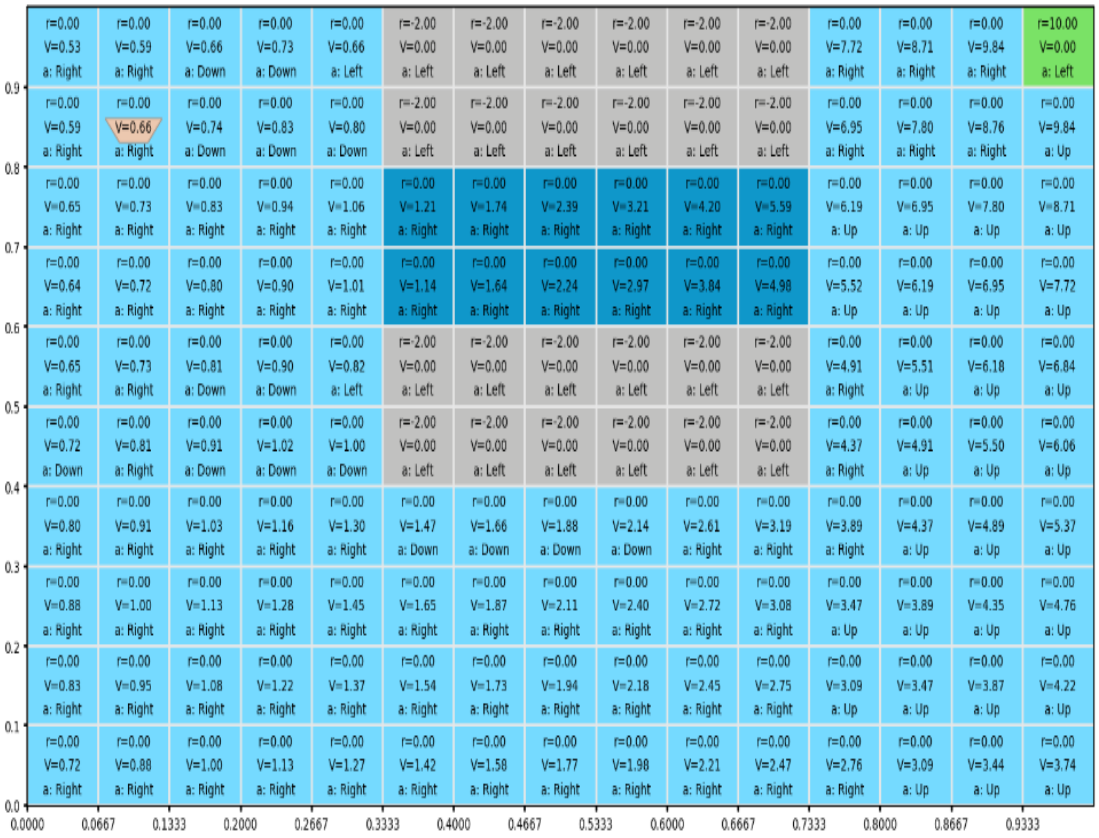
2.1 Task 1

```
def action_v(state, V):
    A = []
    for transition in env.transitions[state[0], state[1]]:
        action_value = 0
        for next_state, reward, done, prob in transition:
            action_value += prob * (reward + (0.9 *
                                    V[next_state] if not done else
    0))
        A.append(action_value)
    return A
# TODO: Compute the value function and policy (Tasks 1, 2 and 3)
    value_est, policy = np.zeros((env.w, env.h)), np.zeros((env.w,
    env.h))
    theta = value_est.copy()
    eps = 0.0001
    width,height=env.w,env.h
    for itera in range(100):
        env.clear_text()
        delta=0
        for i in range(width):
            for j in range(height):
                A = action_v((i, j), value_est)
                value_est[i,j] = np.max(A)
                policy[i,j] = np.argmax(A)
        env.draw_values(value_est)
        env.draw_actions(policy)
```

```
25          env.render()
```

Listing 1: Task 1

State Values:



2.2 Question 2

What is the state value of the harbour and rock states? Why?

The State Value of the harbour and states as seen is zero, as they are the termination conditions or terminal states,The value of a state is the expected sum of all future rewards when starting in that state and following a specific policy. For the terminal state, this is zero - there are no more rewards to be had.

2.3 Task 2

```
        done=False
        while not done:
            # Select a random action
            # TODO: Use the policy to take the optimal action (Task 2)
            #action = int(np.random.random()*4)
            action=policy[state]
            # Step the environment
            state, reward, done, _ = env.step(action)
            # Render and sleep
            env.render()
            sleep(0.1)
```

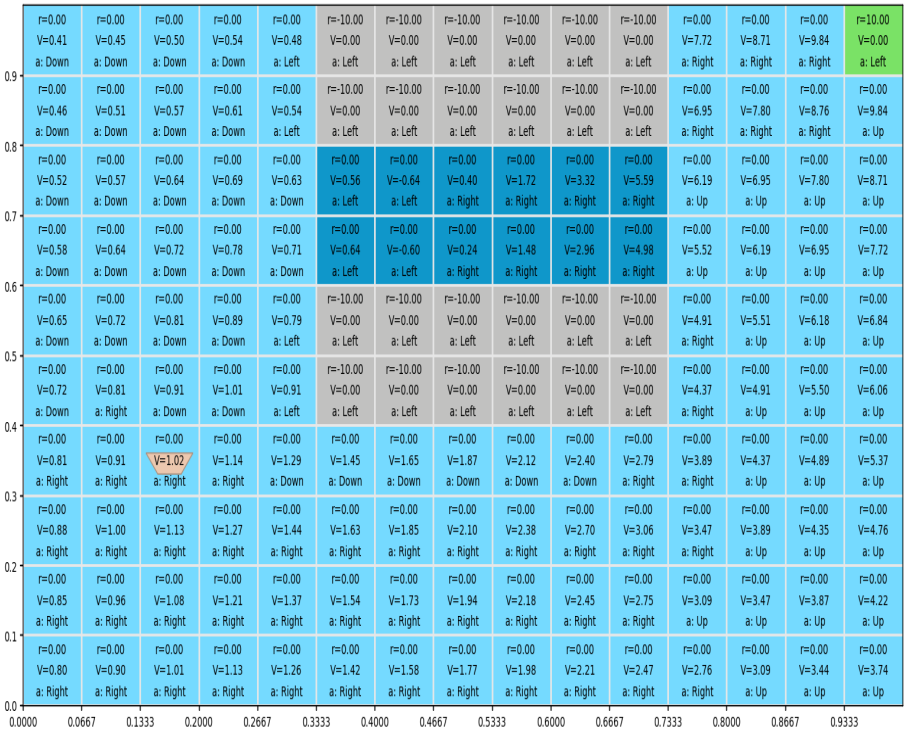
Listing 2: Task 2

The sailor is able to reach the harbour most of the times using the optimal path, whenever it failed, it was crashing into the rocks

2.4 Question 3

Which path did the sailor choose? If you change the reward for hitting the rocks to -10 (that is, make the sailor value his life more), does he still choose the same path?

The sailor takes the longer path instead of the narrower path as it is a more safer option.As you can see the Value function is very low for the states in the beginning of the narrow package and the other path has higher values.



2.5 Question 4

What happens if you run the algorithm for a smaller amount of iterations? Do the value function and policy still converge? Which of them - the policy or value function - needs less iterations to converge, if any? Justify your answer.

The algorithm at-least takes 30 iterations to converge, anything below that it doesn't converge for example it didn't converge at 15 iterations. Both the policy and the value function take the same time to converge, at-least that is what i observed. It is due to the fact that, as the Values keep changing, so does the policy, and once the Values have converged so has the policies.

This was checked by doing a simple comparison(numpy.array_equal), and both the policies converged at the 30th epoch and the value function converged as well at the 30th epoch (given the small epsilon)

But something also observed was, the optimal policy(shortest path from the narrow passage) was learnt faster than the value function i.e the values in the optimal path were still changing, but the policies werent.

2.6 Task 3

Set the reward for crashing into the rocks back to -2. Change the termination condition of your algorithm to make it run until convergence. You can assume the values to have converged if the maximum change in value is lower than a certain threshold :

```
# TODO: Compute the value function and policy (Tasks 1, 2 and 3)
    value_est, policy = np.zeros((env.w, env.h)), np.zeros((env.w,
    env.h))
    theta = value_est.copy()
    eps = 0.0001
    width,height=env.w,env.h
    for itera in range(100):
        env.clear_text()
        delta=0
        for i in range(width):
            for j in range(height):
                A = action_v((i, j), value_est)
                value_est[i,j] = np.max(A)
                policy[i,j] = np.argmax(A)
                #From here epsilon code impleted
                delta=max(delta,np.abs(value_est[i,j]-theta[i,j]))
        if (delta < eps):
            print('Stopped',itera)
            break
        else :
            theta = np.copy(value_est)
        #To check values every iteration
        #env.draw_values(value_est)
        #env.draw_actions(policy)
        #env.render()

    # Show the values and the policy
    env.draw_values(value_est)
    env.draw_actions(policy)
    env.render()
    sleep(1)
```

Listing 3: Task 3

2.7 Task 4

Modify the program to compute the discounted return of the initial state. Create a loop to run the program for N 1000 episodes and see what values you get. Report the average and standard deviation.

```
# TODO: Run multiple episodes and compute the discounted returns (
    Task 4)
    done = False
    discount_r=[]
    for i in range(1000):
         state = env.reset()
         temp_discount_r=0
         done=False
         j=0
```

```
        while not done:
            # Select a random action
            # TODO: Use the policy to take the optimal action (Task 2)
            action = int(np.random.random()*4)
            action=policy[state]
            # Step the environment
            state, reward, done, _ = env.step(action)
            temp_discount_r+=(0.9**j) * reward
            #Render and sleep
            #env.render()
            #sleep(0.1)\
            j+=1
        discount_r.append(temp_discount_r)
    print("Mean",mean(discount_r))
    print("std deviation",np.std(np.array(discount_r)))
```

Listing 4: Task 3

Mean 0.6578227696413183
std deviation 1.3761068012248063
Which is very close to the Value function of the state the sailor started in

2.8 Question 5

What is the relationship between the discounted return and the value function? Explain briefly.

As we can clearly observe that the mean of discounted rewards over 1000 episodes is very close to the value function of the state the sailor started in , we can say that the value function is the expectation of the discounted reward. As we do more and more iterations both the values will eventually converge

Cross checking this with setting the state of the sailor to start in 13,9 the value function is 9.44 and the average obtained is 9.4335466 which is again very close

2.9 Question 6

Imagine a reinforcement learning problem involving a robot exploring an unknown environment. Could the value iteration approach used here be applied directly to that problem? Why/why not? Which of the assumptions are unrealistic, if any?

No the value iteration cannot be applied directly in the robot exploring an unknown environment because

- Value iteration assumes that we know all the states, including the end states, and is a finite state space but in an unknown environment that is not possible
- In an unknown environment we also don't know the end state from the start, it would not be possible to iterate over a space for value iteration

when we don't know the "end state", as Value iteration starts from the end state, and works it way backwards

- The transition probabilities between states are not available in an unknown environment, making it not possible not 'iterate' over and we wont be able to define a policy