# RL Lab 3

Harsh Raj (180010017)

Akhilesh Bharadwaj (180010009)

**Note**: Code for all questions (including simulations etc) is attached in the submitted zip. Artifacts including the images generated are also attached in the zip. Please make sure to install dependencies mentioned in the requirements.txt before running any submitted code.

```
# Install dependencies
pip3 install -r requirements.txt

# run codes corresponding to each question
python3 runner.py
```

All the charts can be accessed online on this wandb repo.

## Env Details:

For the current assignment, we consider the following Grid world environment as MDP:

```
 __________________
|0  |0  |0  |1    |
|0  |0  |0  |-100 |
|0  |XXX|0  |0    |
|0  |0  |0  |0    |
-------------------
```

The cells represent reward associated with the states, `xxx` represents the state that is blocked. The states with non zero reward are absorbing states. An agent reaching an absorbing state can not move out of the state and the episode ends once an agent reaches the absorbing state.

If an agent selects an action a, there is 80% probability of the agent moving in that direction, and 10% each probability of moving in the direction orthogonal to the selected action a.

Even though this is a finite horizon setting, We formulate this as a discounted reward MDP (with discount factor 0.9), encouraging the agent to choose the actions that make them reach the optimal state in the least possible number of steps.

## Model Settings:

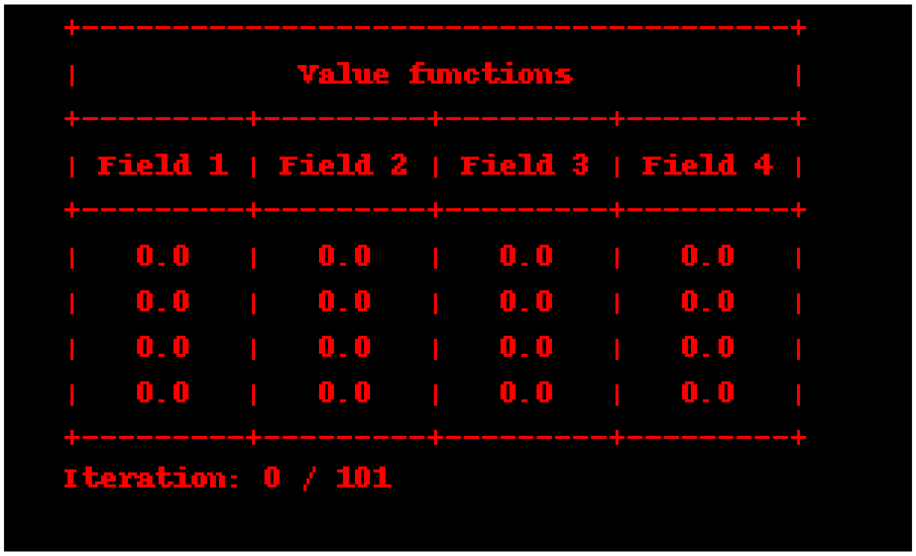
For policy and value iteration, we initialize the value functions with zeros. At each state, we determine the transition probabilities by taking samples of actions. More formally, at state S0, we take an action ‘a’, and observe the next state S1. We store this tuple (S0, a, S1) and set the environment state back to S0. We do this for num_steps (=100) to have a good estimate of transition probabilities corresponding to current state and action pairs.

## Observations and conclusion:

We observe that the policy iteration is faster to converge compared to the value iteration.

Since for an MDP, the optimal value function is unique, to confirm if the policy converged is an optimal policy, we can compare the corresponding value function with the optimal value function obtained from the value iteration.

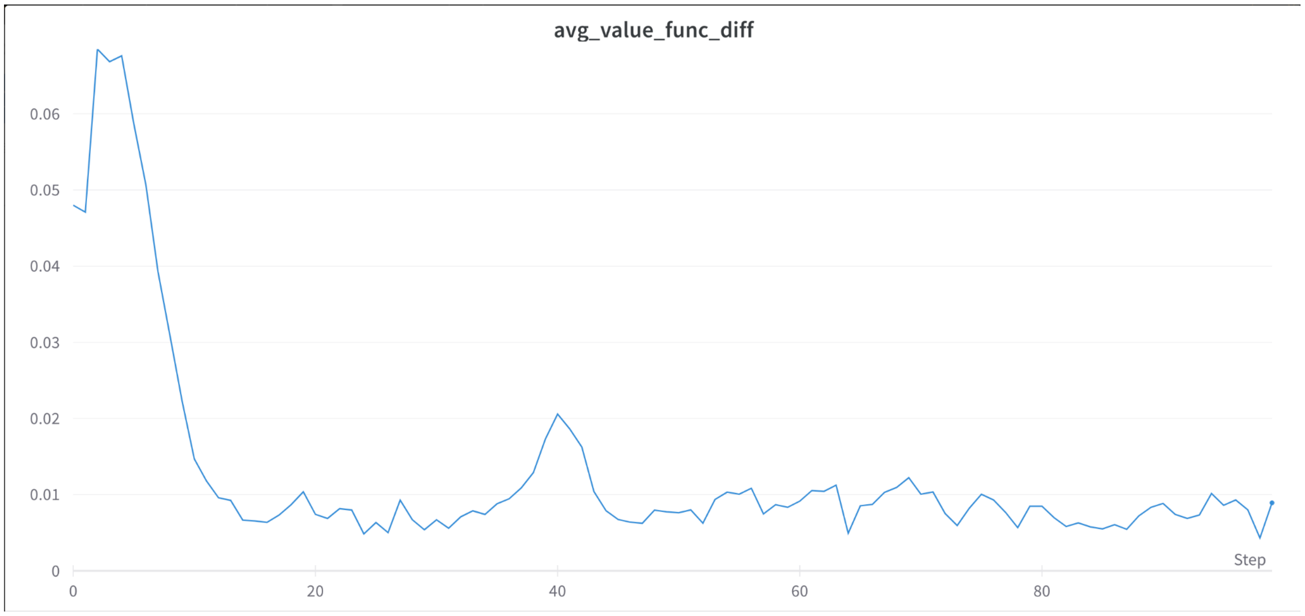
1. Value Iteration



```
+---------------------------------------+
|            Value functions            |
+---------+---------+---------+---------+
| Field 1 | Field 2 | Field 3 | Field 4 |
+---------+---------+---------+---------+
|   0.0   |   0.0   |   0.0   |   0.0   |
|   0.0   |   0.0   |   0.0   |   0.0   |
|   0.0   |   0.0   |   0.0   |   0.0   |
|   0.0   |   0.0   |   0.0   |   0.0   |
+---------+---------+---------+---------+
Iteration: 0 / 101
```

Value function corresponding to each state per iteration
(Converting to Pdf creates rendering issues. Refer to `artifacts/value_functions_diff.gif` or the online version of this doc)

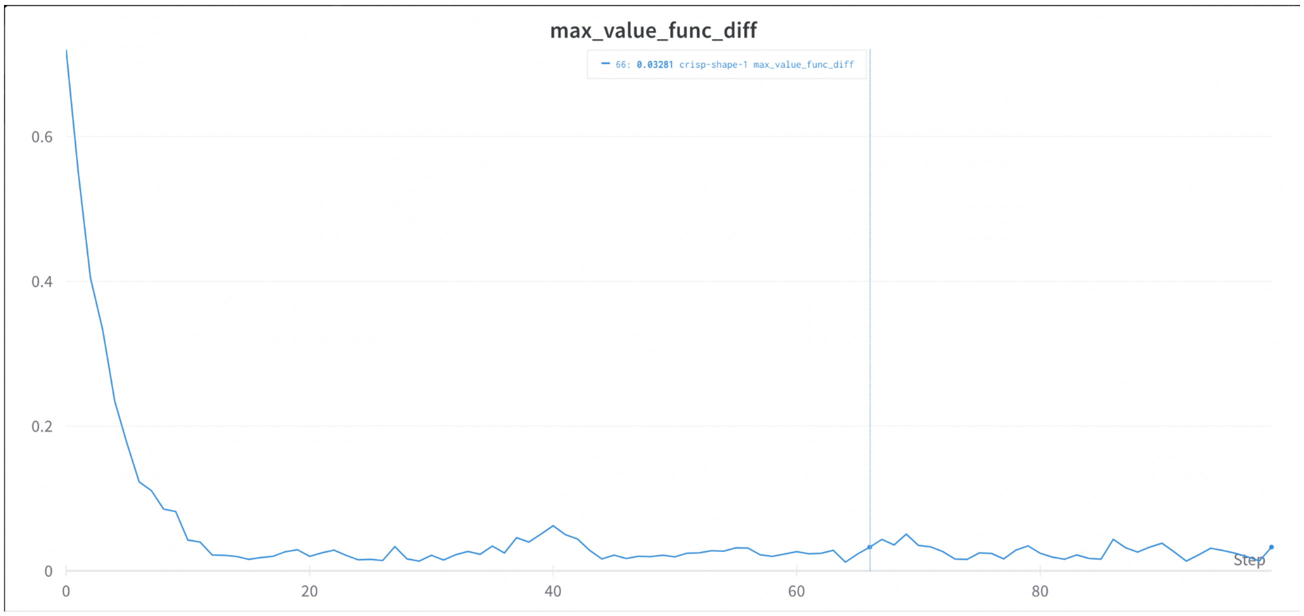
**Note**: For the given charts, x-axis represents the iterations.



Here we show the average value of difference in sum of value functions of states across each update. As can be observed from the given plot, the difference between value functions is large initially, and gradually it starts to decrease giving hints that the value function is approaching the optimal value function of the given task.

We use the following equation to obtain the average value function difference:

```
avg_diff = (old_val_func - new_val_func).abs().sum()
```

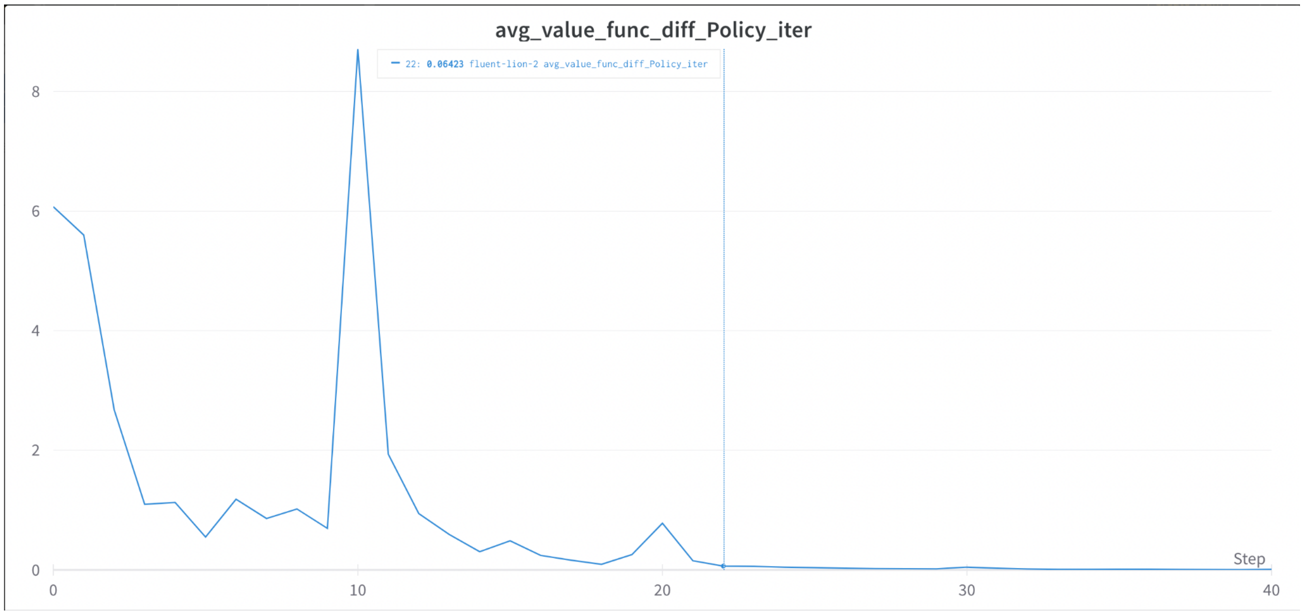


Here we show the maximum difference between the value of function of states across each update. As can be observed from the given plot, the largest difference between any state is decreasing across the iterations. This also gives strong hints that the value function has approached the optimal value function.

We use the following equation to obtain the max value function difference:

```
max_diff = (old_val_func - new_val_func).abs().max()
```

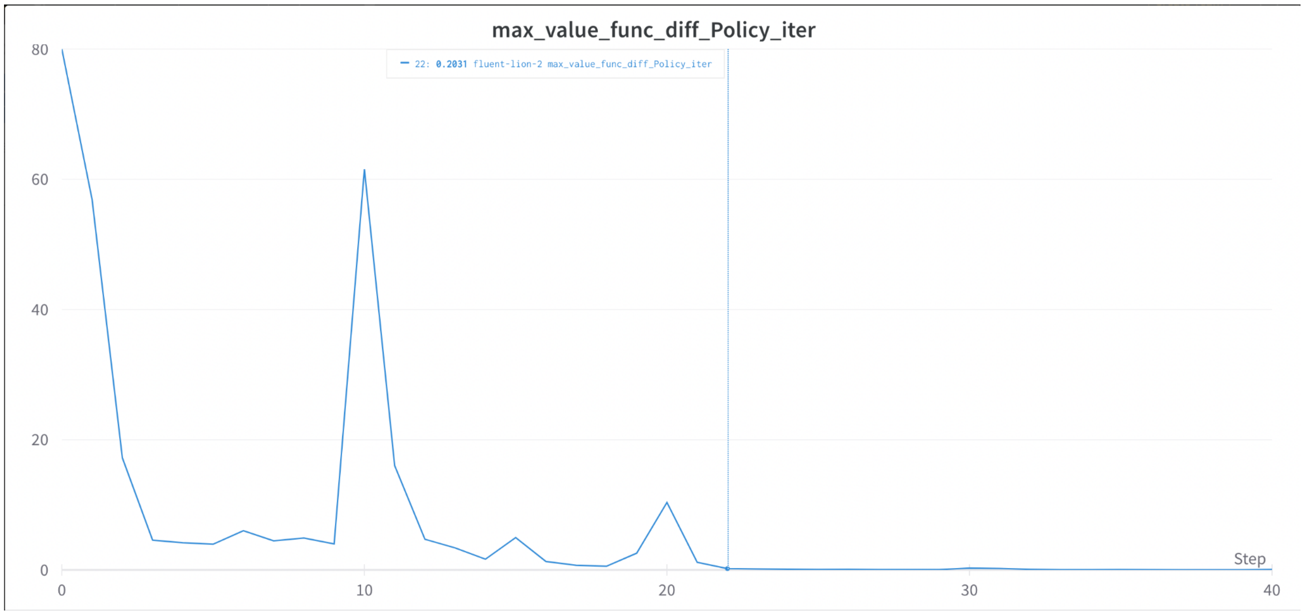
## 2. Policy Iteration:



Here we show the average value of difference in sum of value functions of states across each update. As can be observed from the given plot, the difference between value functions decreases and suddenly jumps at times. This could be understood by the fact that the policy evaluation process makes the value function converge to the optimal value function corresponding to the current policy. Then policy improvement comes, and the difference grows again as the optimal value function corresponding to the new policy is different from that of the old policy. Gradually the policy converges to optimal policy, and we do not see any more jumps in the difference of value function.

We use the following equation to obtain the average value function difference:

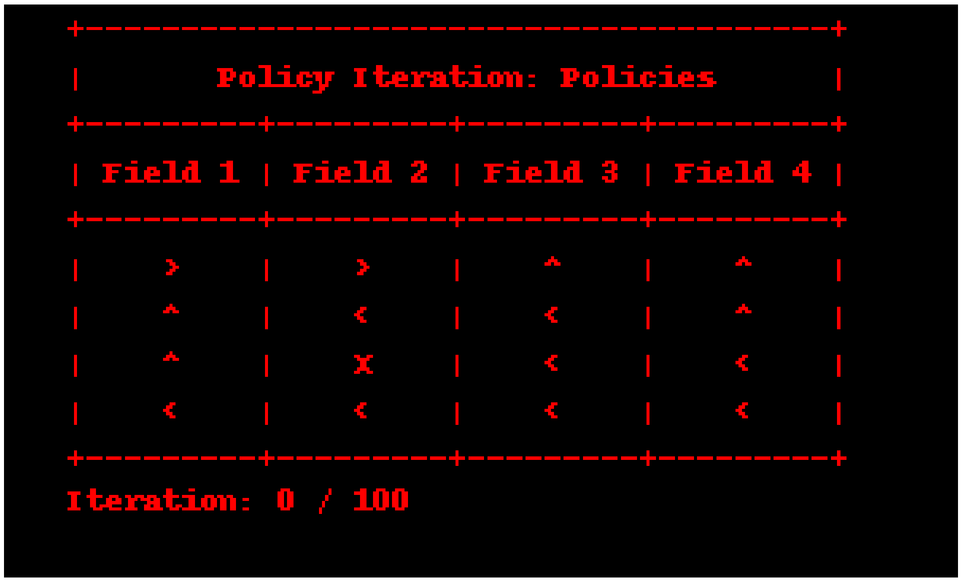
```
avg_diff = (old_val_func - new_val_func).abs().sum()
```



Here we show the maximum difference between the value of function of states across each update. A trend similar to what above is observed.

We use the following equation to obtain the max value function difference:

```
max_diff = (old_val_func - new_val_func).abs().max()
```



```
+----------------------------------------+
|        Policy Iteration: Policies      |
+---------+---------+---------+---------+
| Field 1 | Field 2 | Field 3 | Field 4 |
+---------+---------+---------+---------+
|    >    |    >    |    ^    |    ^    |
|    ^    |    <    |    <    |    ^    |
|    ^    |    X    |    <    |    <    |
|    <    |    <    |    <    |    <    |
+---------+---------+---------+---------+
Iteration: 0 / 100
```

Policies corresponding to each state per iteration
(Converting to Pdf creates rendering issues. Refer to `artifacts/policy_iteration_agent_gif.gif` or the [online version of this doc](online version of this doc))

**Note**: Some states have almost equal values after convergence. This justifies the fact that even after convergence, policy of some state is altering between two directions, as both the next states have almost the same values.