RL PA2

Ajay S Joshi (CS15B047)

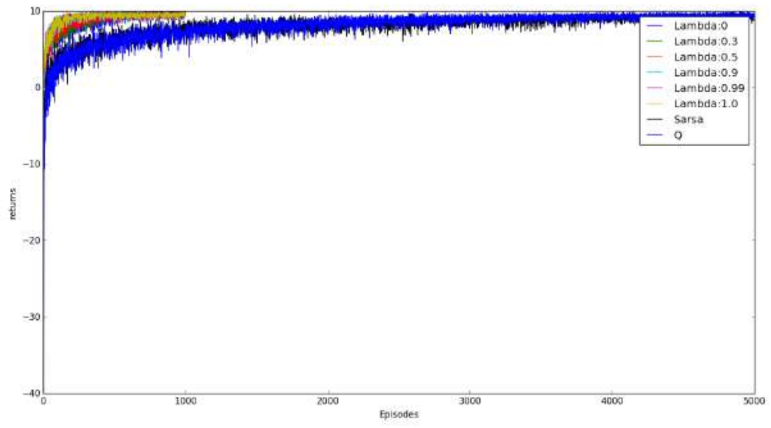
March 29, 2018

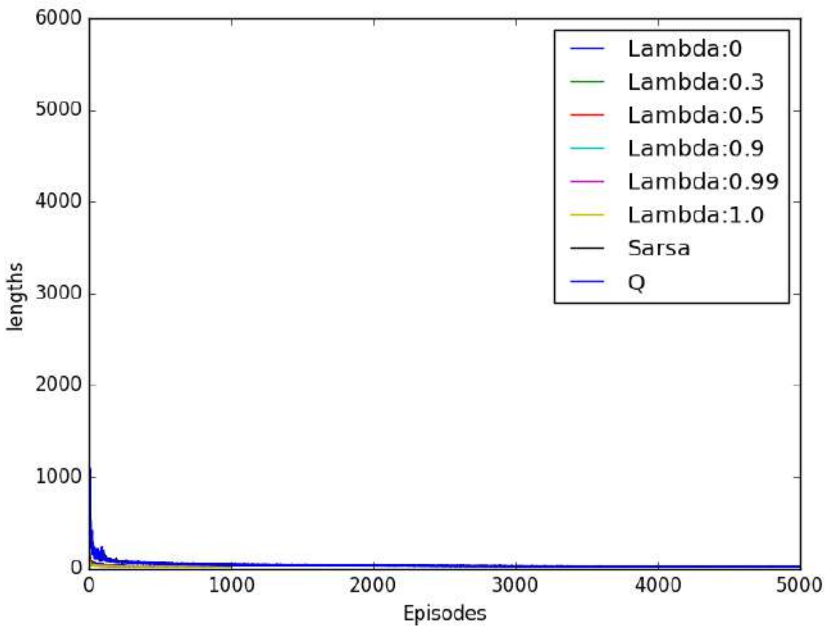
1 Q Learning and Sarsa

1.1 Comparison of learning algorithms

1. Initial number of steps per episode are high as agent policy is random and environment is also stochastic.
2. In case of C, steps are particularly high for Sarsa-λ, as goal is surrounded by negative rewards, which agent will try to avoid as effect of a reward is propagated through all states in trajectory, and so will not reach C easily.
3. In all 3 problems, Sarsa(λ) outperforms Q-Learning and Sarsa, in terms of average returns per episode, i.e. it gets fastest to highest returns.
4. In terms of number of average steps per episode also, Sarsa(λ) has the better performance out of the 3 for all values of λ, with the only exception of some spikes in initial part for some λ.

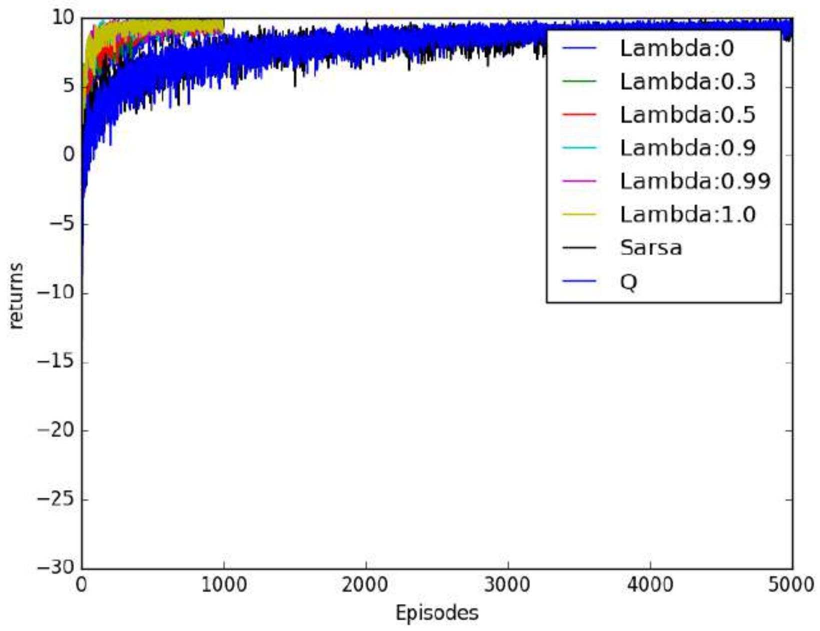
- Problem A :



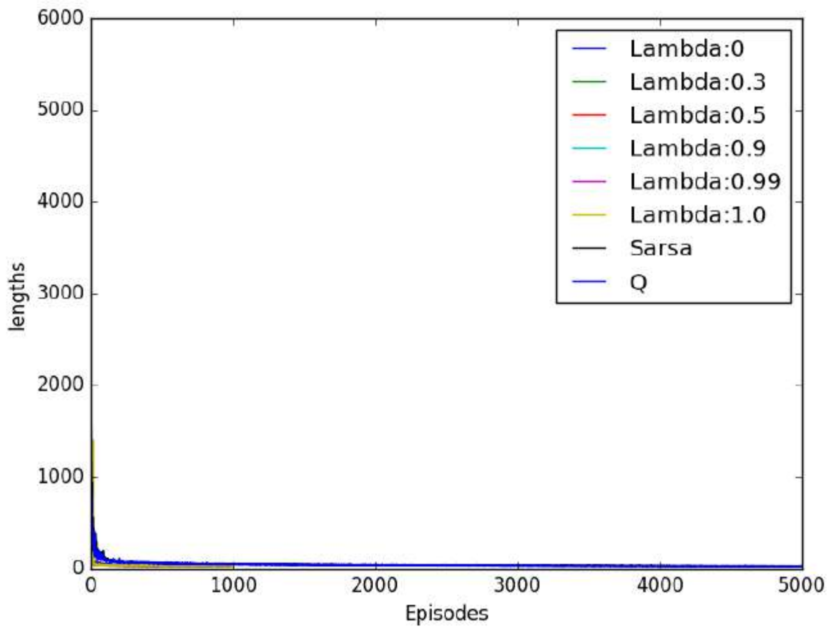


6000
5000
4000
3000
2000
1000
0
lengths
0
1000
2000
3000
4000
5000
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
Sarsa
Q

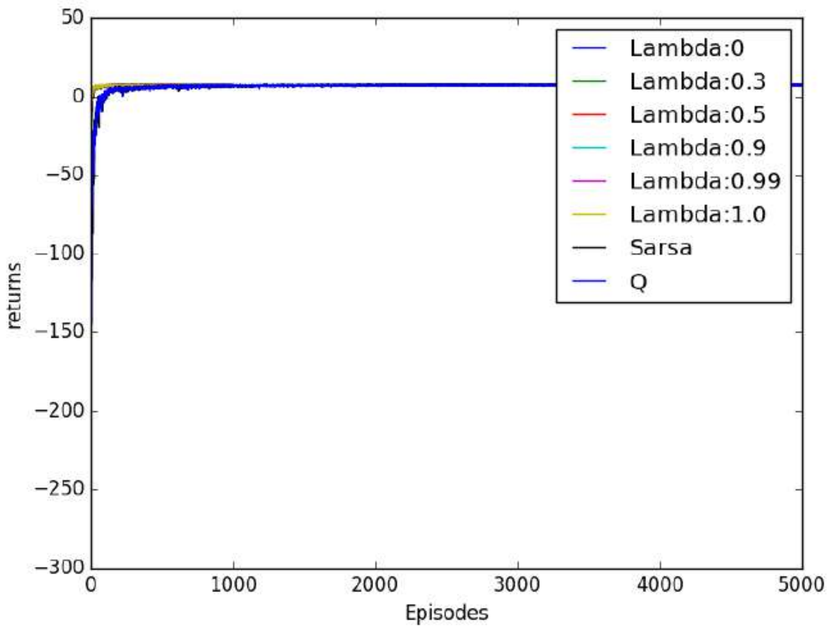
- Problem B :

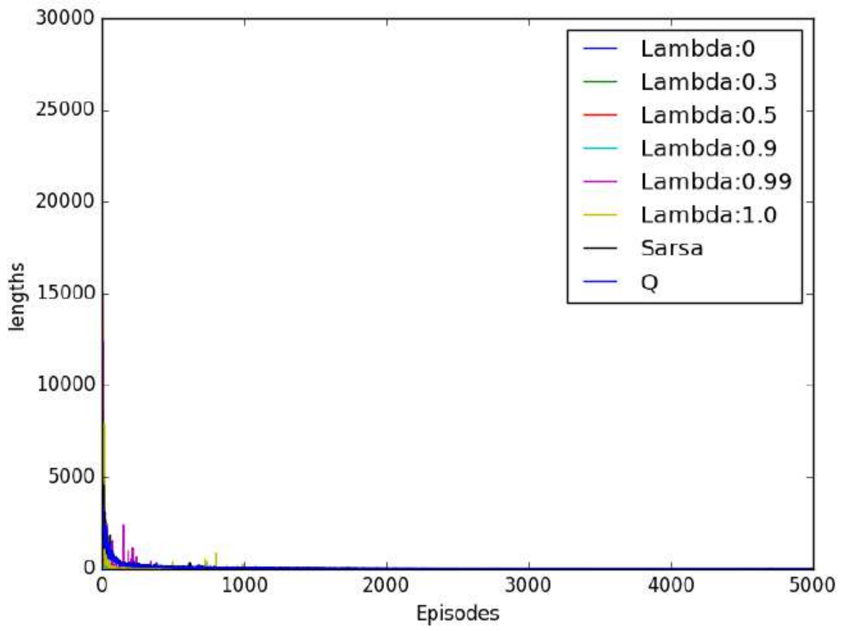


10
5
0
−5
−10
−15
−20
−25
−30
returns
0
1000
2000
3000
4000
5000
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
Sarsa
Q


lengths
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
Sarsa
Q

- Problem C :


returns
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
Sarsa
Q



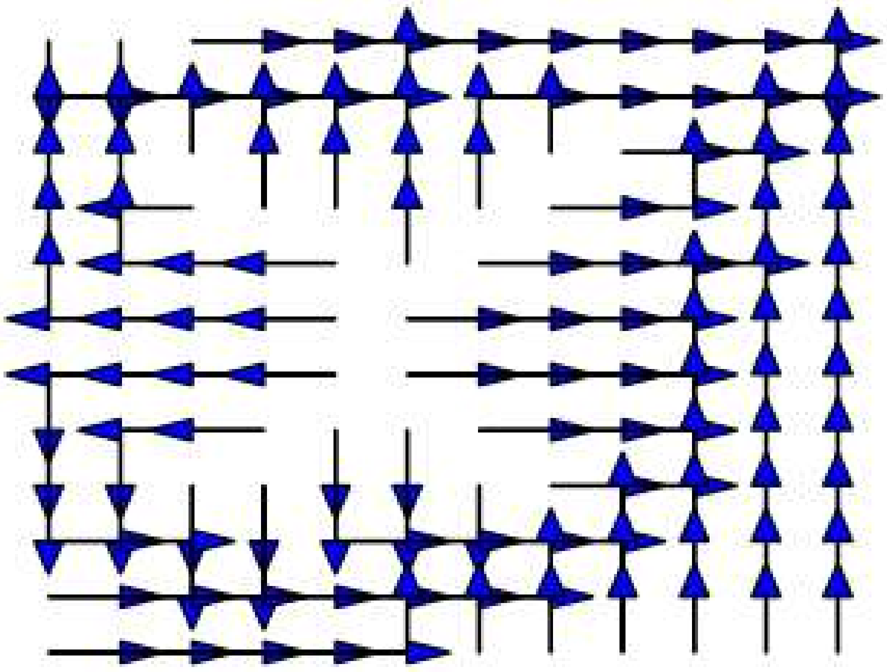
1.2 Learned Policies

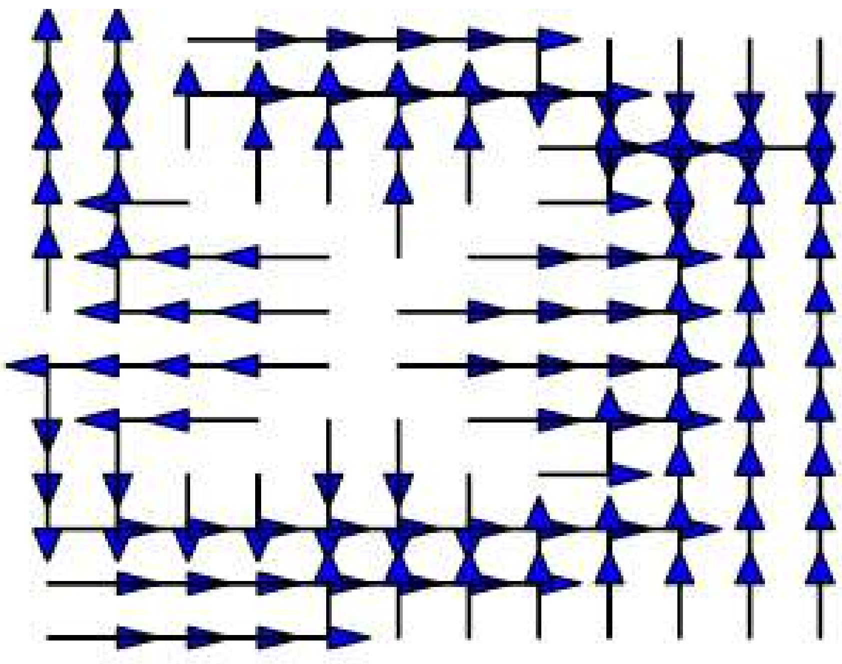
Learned policies by agent are given in order A,B,C.

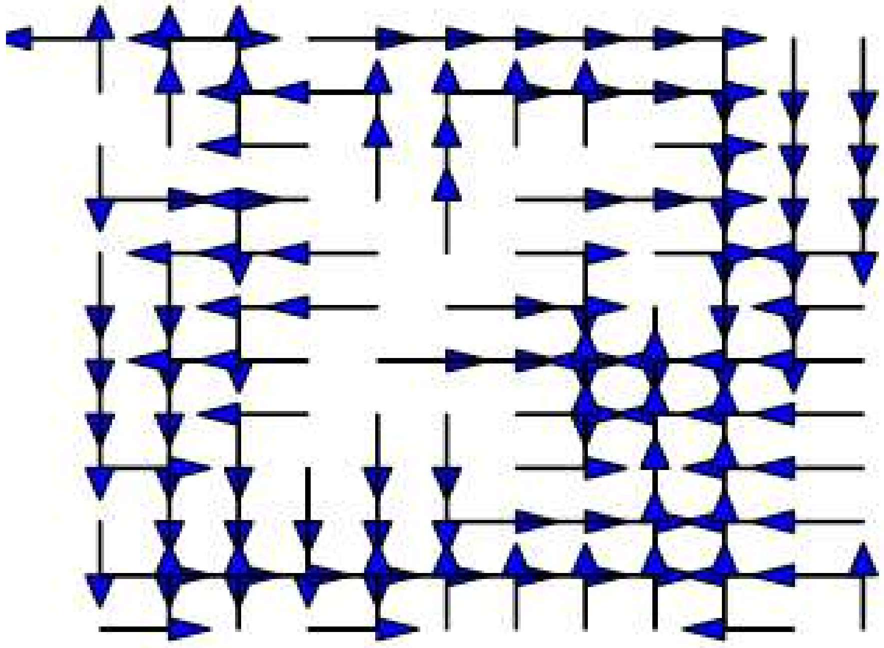
1. Below policies are good for the stochastic environment as they learn to get the goal quickly from almost every point in grid.

2. Policies can made optimal at every point by increasing number of episodes, which will eliminate suboptimal actions in some states.

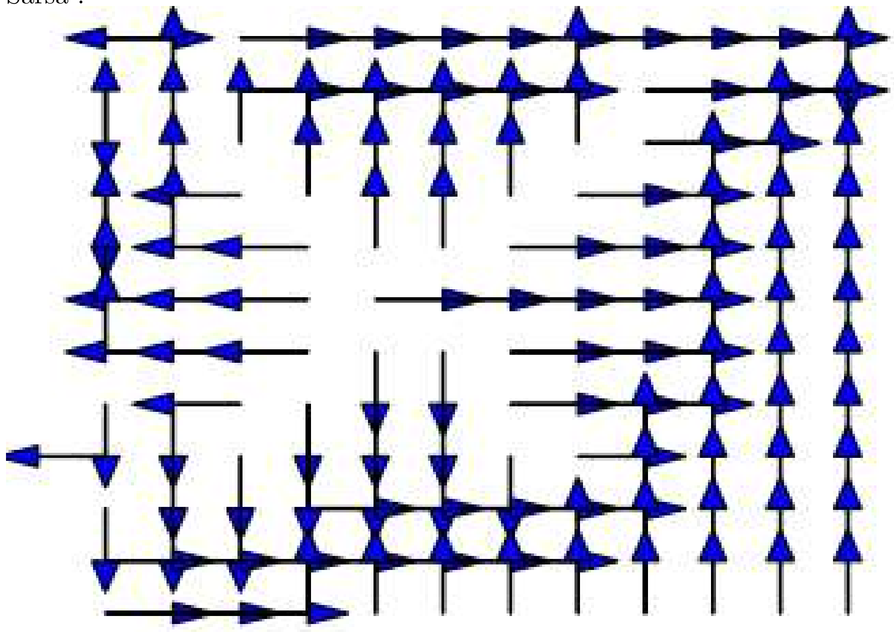
- Q-Learning:

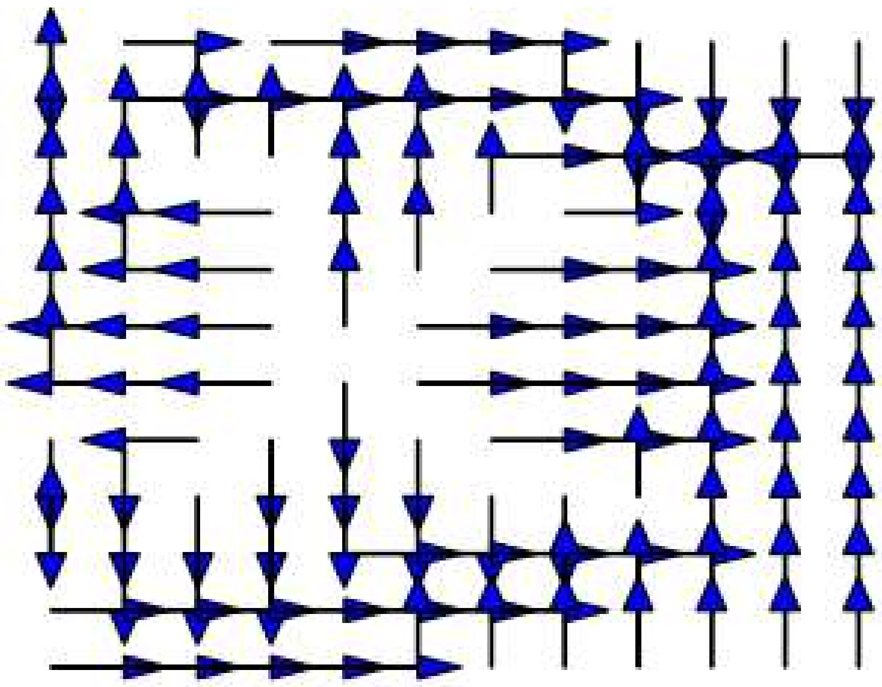


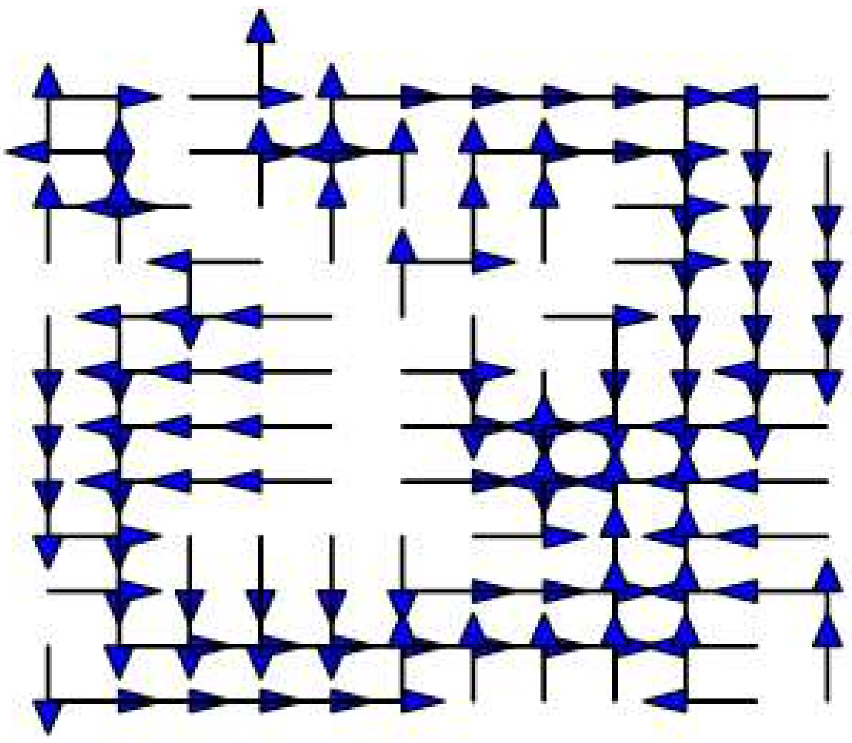




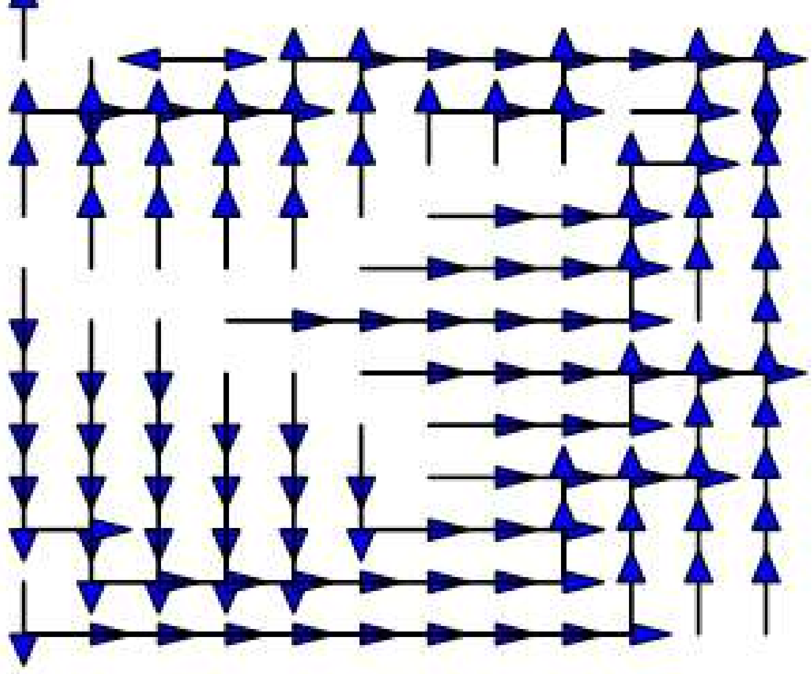
- Sarsa :

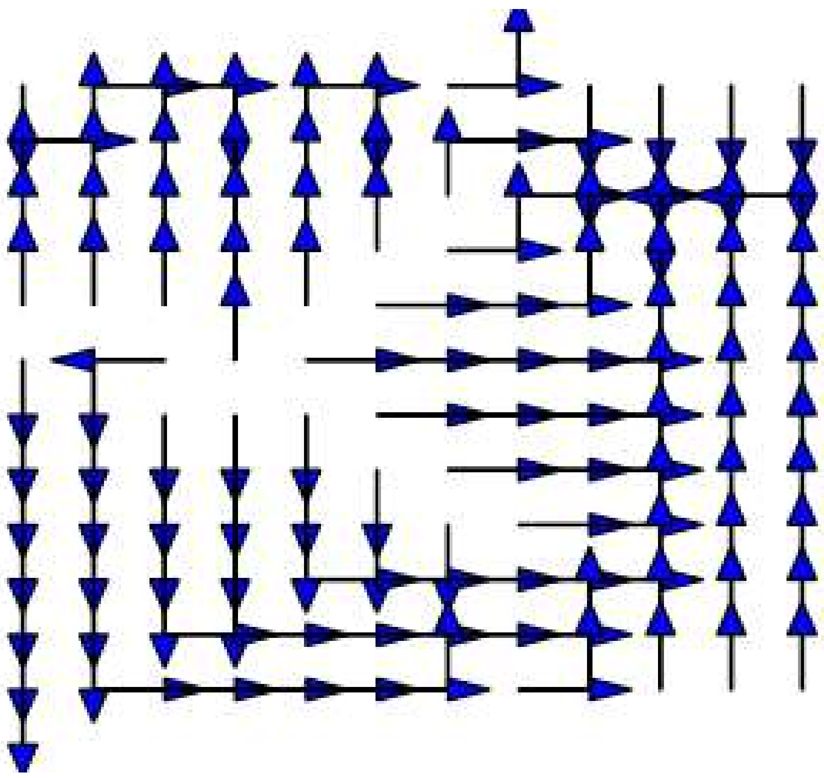


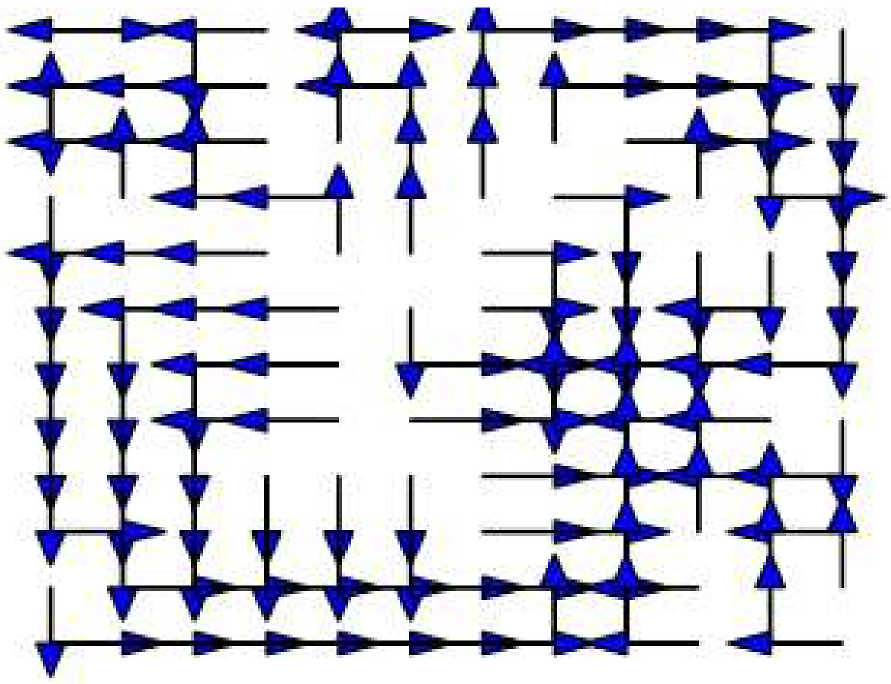




- Sarsa-λ : (λ=1)



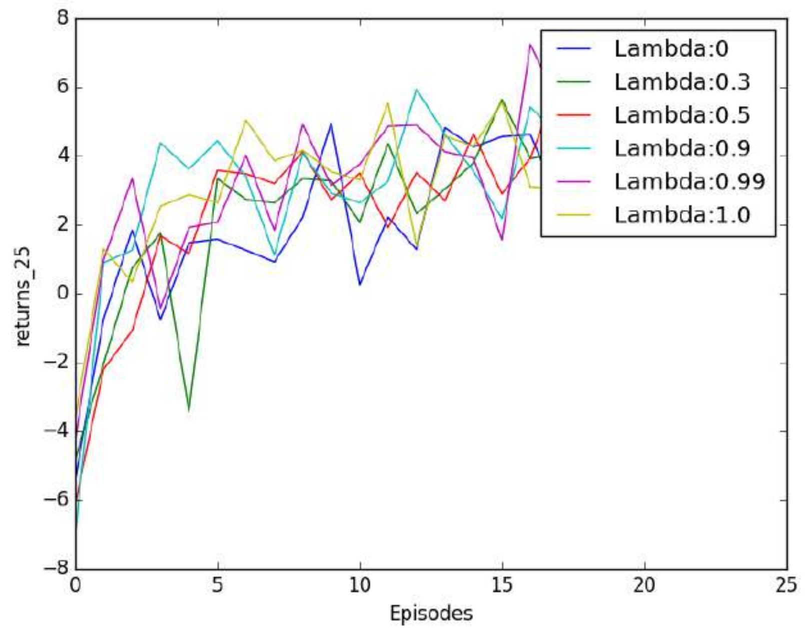


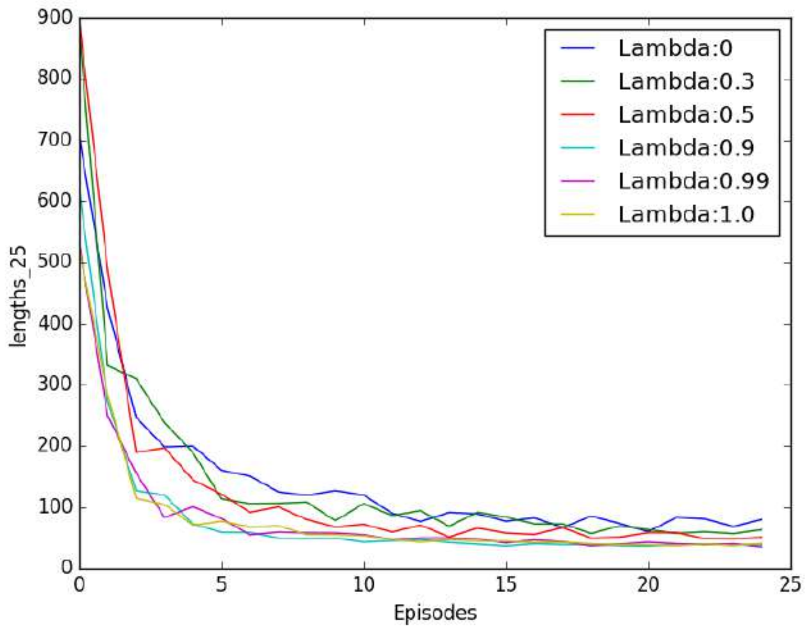


1.3 Sarsa- λ performance on 25 episodes

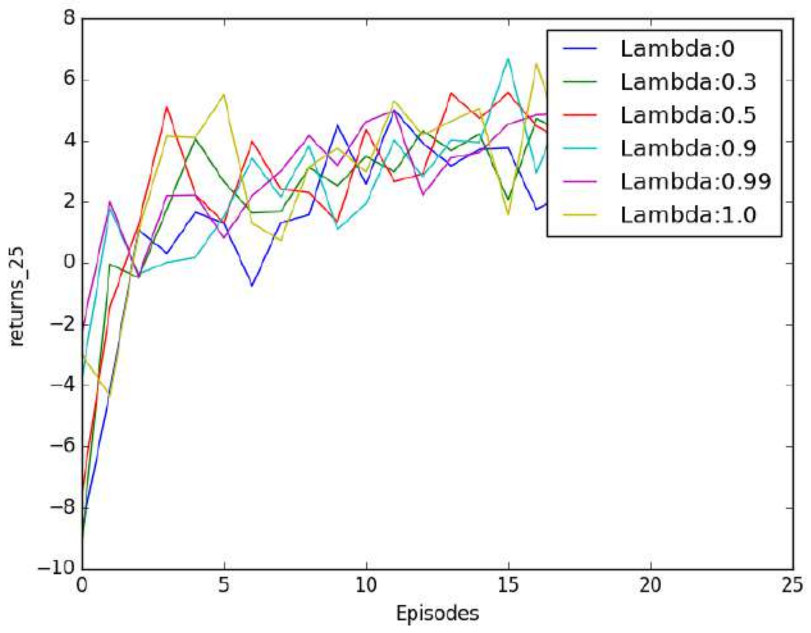
1. These graphs show learning of Sarsa-λ in the initial phase(25 episodes).
2. Sarsa-λ learns faster than both Q-Learning and Sarsa, as in it, due to use of eligibility traces, values of all states start getting updated right after they are visited for first time, resulting in faster learning.
3. Also shows initial spikes in episode length, showing that it can get stuck initially in the environment.

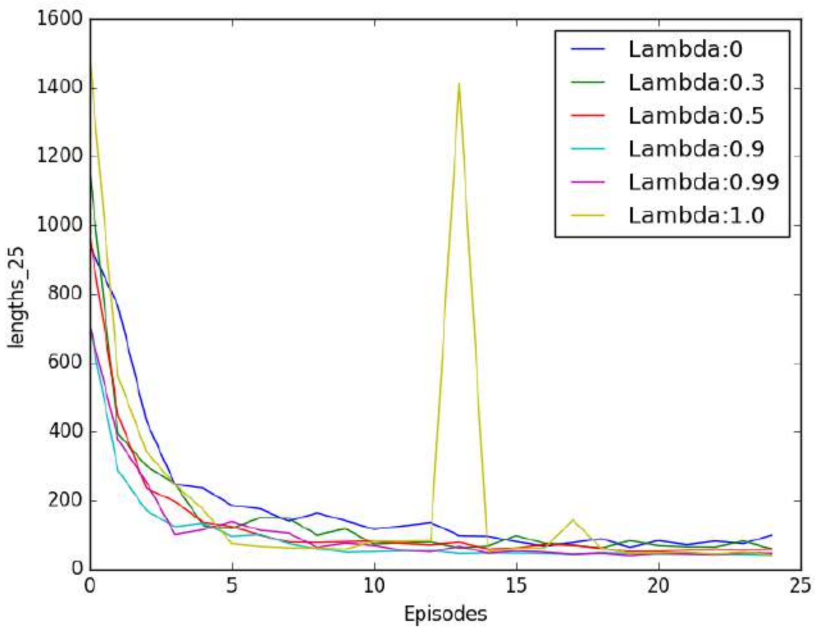
- Problem A :


Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
returns_25
Episodes

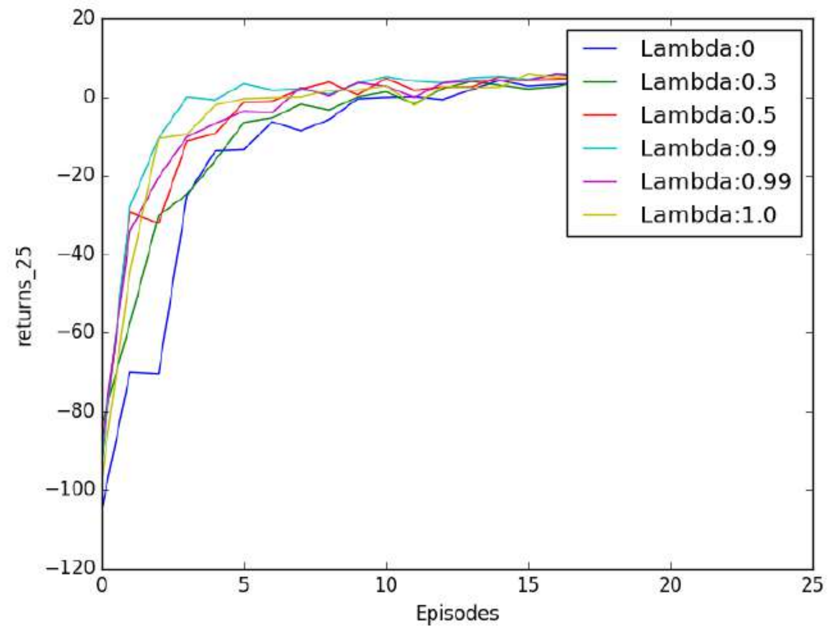

Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0
lengths_25
Episodes

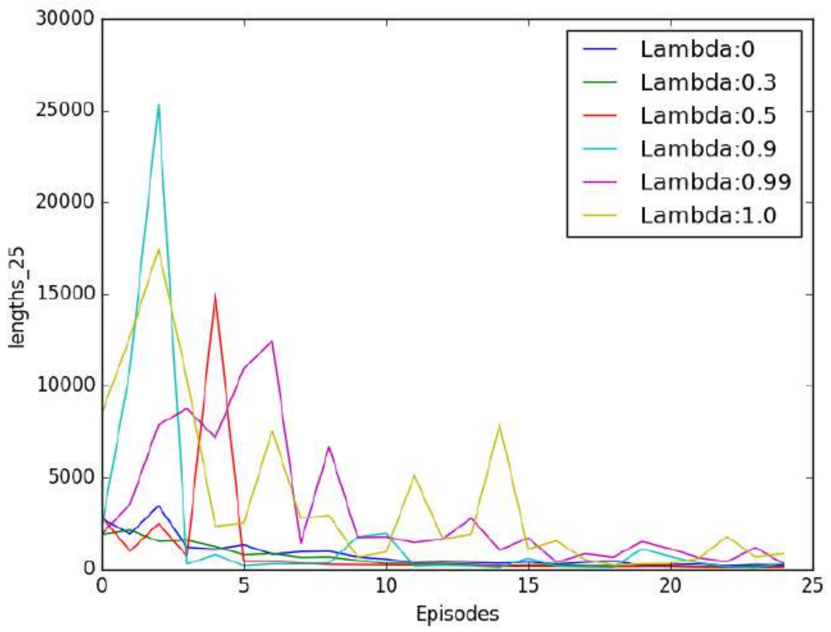
- Problem B :


8
6
4
2
0
−2
−4
−6
−8
−10
returns_25
0
5
10
15
20
25
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0


1600
1400
1200
1000
800
600
400
200
0
lengths_25
0
5
10
15
20
25
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0

- Problem C :


returns_25
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0

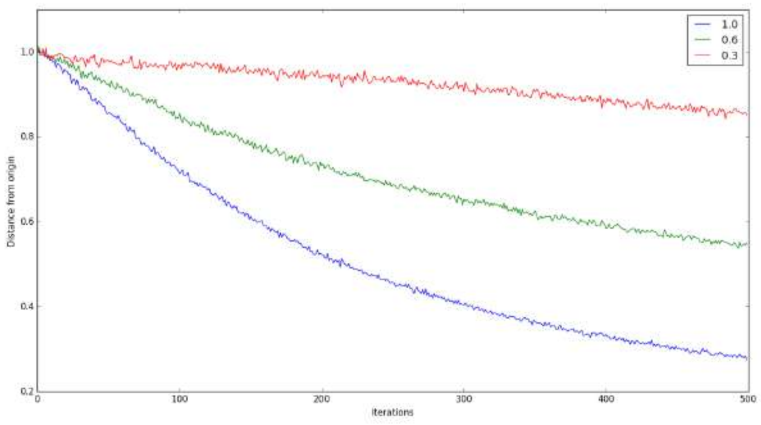

lengths_25
Episodes
Lambda:0
Lambda:0.3
Lambda:0.5
Lambda:0.9
Lambda:0.99
Lambda:1.0

2 Policy Gradient

2.1 Hyperparameter Tuning

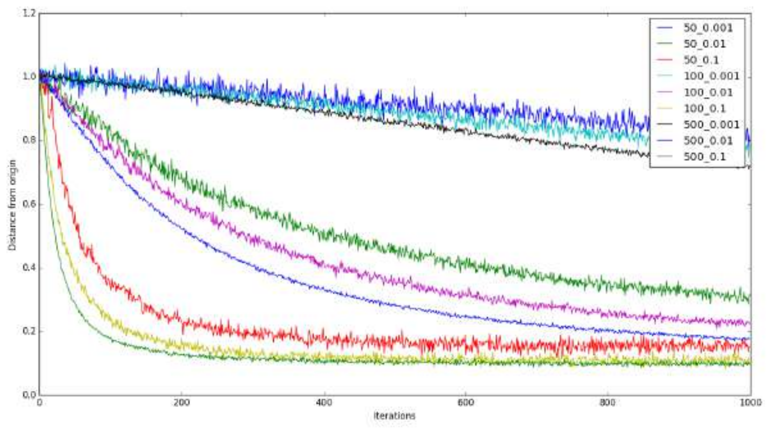
- Tune γ (discount factor):

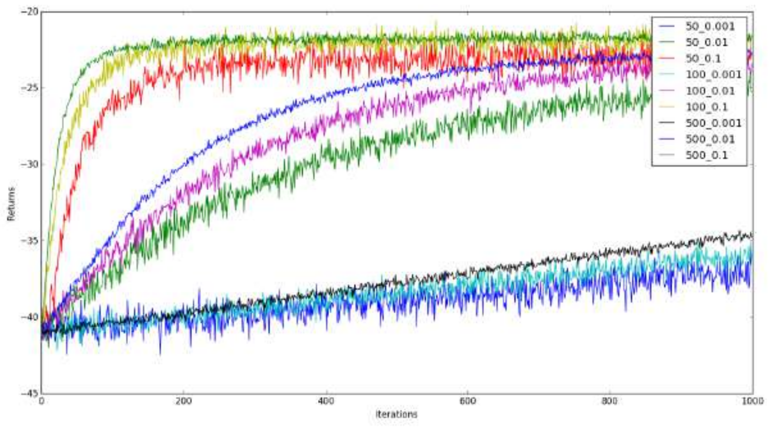
  1. Graph plotted for learning rate = 0.01 with batch size = 500 for all 3 γ values.
  2. As γ decreases, reward signal from later time steps fades away, thus causing agent to not learn from later time steps. So, $\gamma = 1.0$ gives better performance than all other lesser values of γ.
  3. Same logic applies to both circle and elliptical reward function cases.



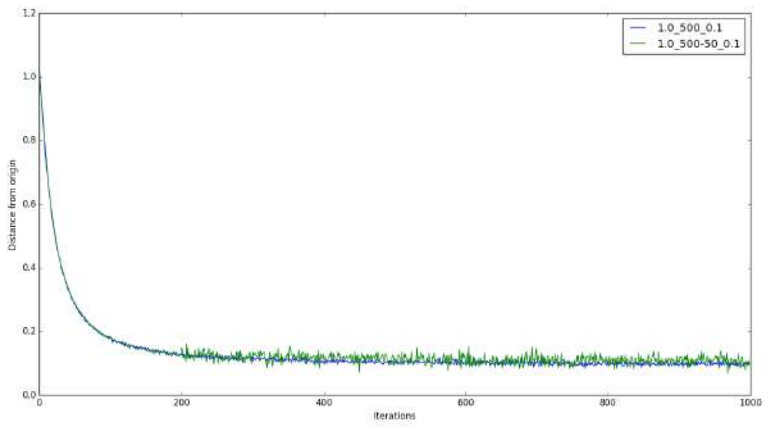
- Tune batch size, learning rate :

  1. All graphs are for $\gamma = 1.0$.
  2. We can see 3 distinct bands of lines together, corresponding to a specific learning rate.
  3. Higher learning rates also converge to the solution, so can be used to reach solution faster.
  4. Higher batch sizes for same learning rate give better performance, due to batch gradient pointing close to the right direction.



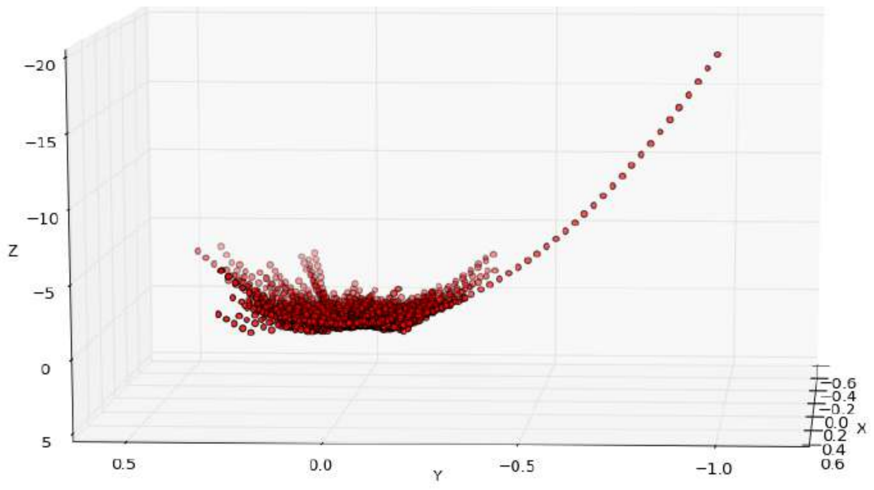


- Learning policy faster :
  Policy can be learnt faster by :
  1. Keeping learning rate high and batch size high at the start.
  2. Reduce batch size as learning progresses.
  3. Here, with learning rate = 0.1, batch size = 500 is used till 200 iterations, and reduced to 50 after that, doing almost 1/3 rd computation of batch size 500 case.
  4. Batch size can be reduced in many different ways other than shown method, to get speedup in learning.



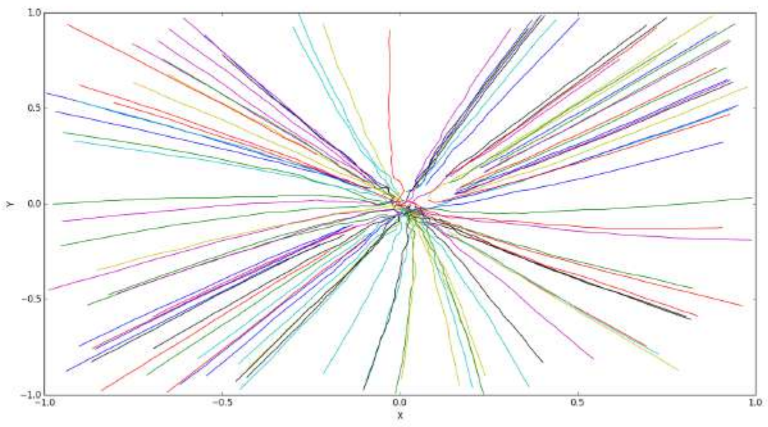
2.2 Value Function Visualization

- Value Function of a state is expected discounted return starting from that state.
- Thus, we can run trajectories in the state space using learned policy and collect discounted returns obtained starting from every point visited, giving a rough idea of value function, as every point can be visited practically only once at max, due to continuous state space.
- Thus, the value function is paraboloid in shape. (inverted)



2.3 Trajectories for learned policy

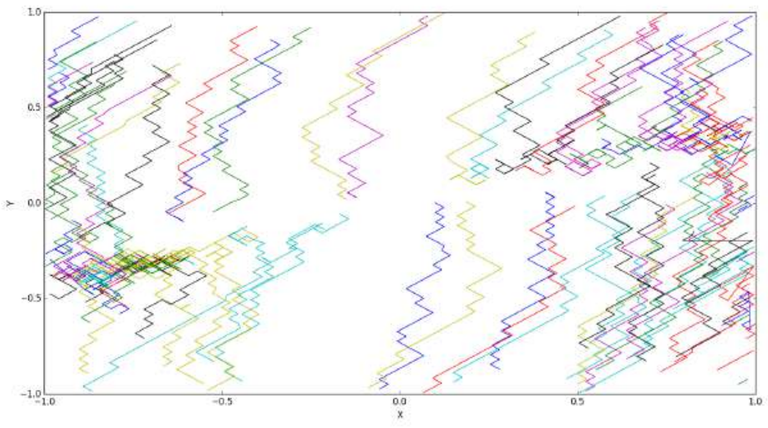
- Chakra :



In chakra case, agent moves towards the origin along a straight line, as it is the direction in which returns increase the most.

- VishamC :

1. In this case, agent moves towards X-axis.
2. The gradient component $\theta_{a_x,x}$, i.e. corresponding to action in X direction , corresponding to X-coordinate is overpowered by noise and the only significant component in gradient remains as $\theta_{a_y,y}$, which is negative.
3. Similar experiment on skewed, different environment with $\gamma = 2$ for calculation of reward, showed presence of weak continuous negative gradient for $\theta_{a_x,x}$, which was unable to make agent converge in X - direction.



1.0
0.5
Y 0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
X