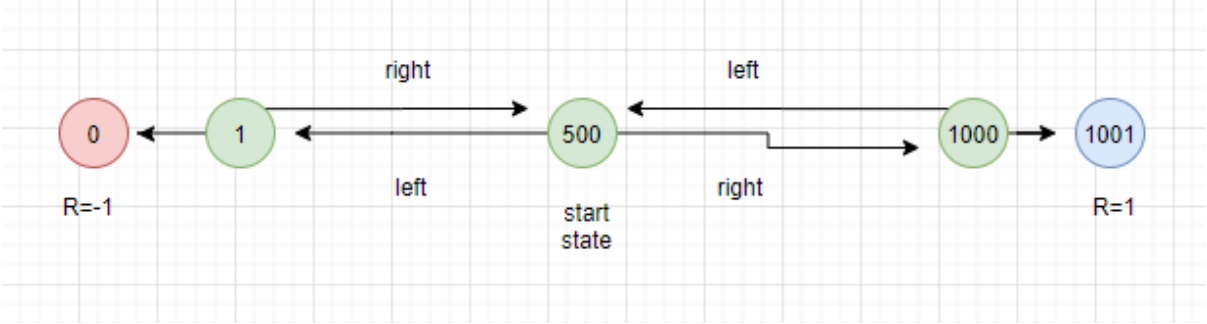
Homework 7

A. Environment Set-Up:

In this homework, we are to work on the 1000 state random walk task. Here our agent starts from the state '500' and can take either left or right action with a 50% chance of probability. Once the agent takes either action, the next state is chosen randomly between the previous or next '100' states. For example, the agent can move right, and he could end up in state 600 or 550, etc. Lastly, the end goal is to terminate towards the right-hand side, which would earn the agent a Reward of '+1', whereas ending on the left side would give him a reward of '-1'. All other states are given a Reward of '0'. Thisl can be visualized by the following figure below.



Furthermore, we implement three kinds of algorithms to generate different results :

B. Algorithm Explanation:

1. Function approximation by state aggregation using monte Carlo algorithm :

The basic idea of function approximation is to approximately calculate the state and its true values without using tabular methods. State aggregation groups 100 consecutive states and update their weight when any one of them is encountered. Monte Carlo algorithm is an algorithm wherein we update our policy or, in this case, the weight after generating a whole episode.

In the algorithm implementation, we first declare the values of alpha the step size, which is *2x105*. Next, we declare the current state and use a randomly generated probability; we move either left or right. Then we randomly generate the number of steps that our agent would take in that direction. The probability is generated using the inbuilt 'rand()' function, whereas the number of steps are generated using the inbuilt 'randi' function. Next, we update our next state and check if it has crossed/reached the terminal state '0' or '1001'. If it has reached the terminal state, we move to the next step (explained later), or we keep taking steps until we reach the terminal state. Once we have reached the terminal state, we assign the rewards. Once we have done this step, we traverse through all the states taken and calculate the *Gt*

using the Monte Carlo algorithm. Once that has been done, we find which group that the states belong to and update its weight accordingly using the following equation.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) \right] \nabla \hat{v}(S, \mathbf{w})$$

Where :

- W=weights
- Alpha =step size
- R=reward
- Gamma=discount factor
- v_hat= is the weights for a given state
- last term gradient remains 1 for state aggregation.

Once we have run the code for about 100000 iterations, we collect all the 'weights' store them and then plot these weights vs states.

2. Semi gradient TD(0) algorithm:

The Semi gradient TD algorithm is similar to the Monte Carlo method in many ways. The main difference is we update the group weights after taking one action. We observe the next state, its reward, its group weight, and update our weights for our current state using a different formula, which is given below.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) \right] \nabla \hat{v}(S, \mathbf{w})$$

Where the v for the next state is also found using 'if' statements and then added to the formula to update the weights, again, we plot the states vs. the weights obtained.

3. True value: using iterative policy evaluation

The true values are calculated for the next state by estimating the probability of reaching the next state from the current state. A probability matrix of dimension 1002x1002 is needed according to the formula. It takes into account the 1000 walkable states as well as the two end states. The indices 1 and 1002 represent the final states, whereas the indices between 2 and 1001 represent the walkable states. The true value function used in this code takes into account three possible scenarios. They are as follows:

• States between 2 and 101: For these states, the movement to the left is restrictive as there are not 100 states to the left. Thus, the probability is distributed between the number of states there are to the left such that the individual walkable states are assigned a probability of 0.01, and the terminal state is assigned by subtracting the sum of probabilities of walkable

states from 1. The 100 states to the right of a particular state are assigned a probability value of 0.01.

• States between 102 and 901: Every state in this range has 100 states to their left as well as right. Thus, each of these states to the left and right is assigned a probability value of 0.01.

• States between 902 and 1001: For these states, the movement to the right is restrictive as there are not 100 states to the right. Thus, the probability is distributed between the number of states there are to the right such that the individual walkable states are assigned a probability of 0.01, and the terminal state is assigned by subtracting the sum of probabilities of walkable states from 1. The 100 states to the left of a particular state are assigned a probability value of 0.01.

Once we have calculated the probability matrix, we introduce algorithm parameters known as theta and delta. Theta helps us determine the accuracy of the estimation, while delta helps in converging the code. Since we need to run the whole code until we get a value that is closer to the real value or the estimate. The value estimation can only take place over many iterations, which is why we run the code in an infinite loop until the delta value becomes less than the theta. Next, we create a loop that would go from all the current states and another loop over all possible next states. This looping is to mimic the movement of our agents across all possible states. We define a value register that stores values based the following formula

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$$

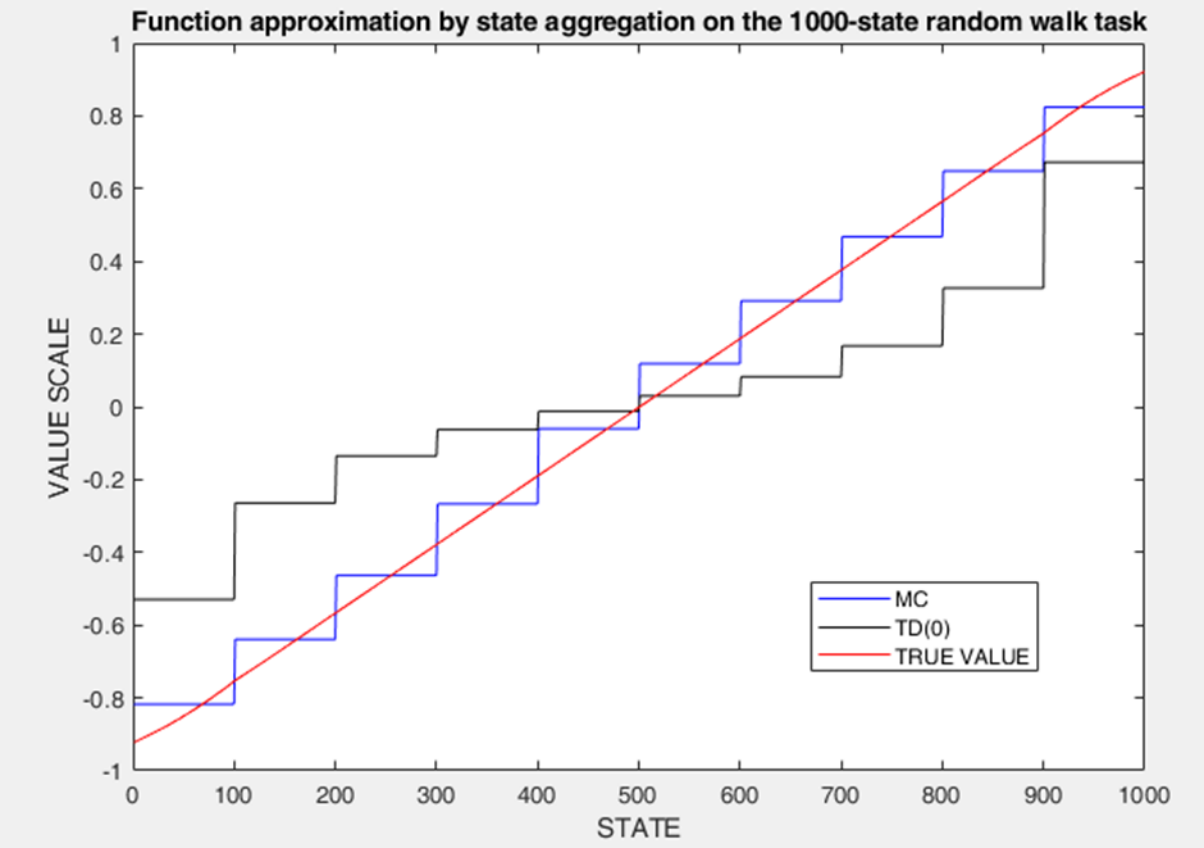
Wherein we provide our conditional probability of pi(a|s) by our policy, which 0.5 in this case because we can move either left or right with a probability of 50%if 50%. The other conditional probability term is our probability of going to a state to another one. We create three separate cases using a 'switch' function, and these cases pertain to the different rewards available for different cases. We use all this knowledge to update the above equation. We make use of an expected update operation in the Bellman equation. Each iterative policy evaluation updates the value of every state to produce the new approximate value function.

Upon updating the value each step using the equation, which helps us to find the value of delta, which in turn helps us find the terminating condition for the infinite loop, which is when 'delta' is less than 'theta.' In the end, we return the data to the value function. Next, we plot this vs. the number of states.

C. Time taken to run code :

33.067165 seconds.

The graph obtained from code :



Graph from the textbook: