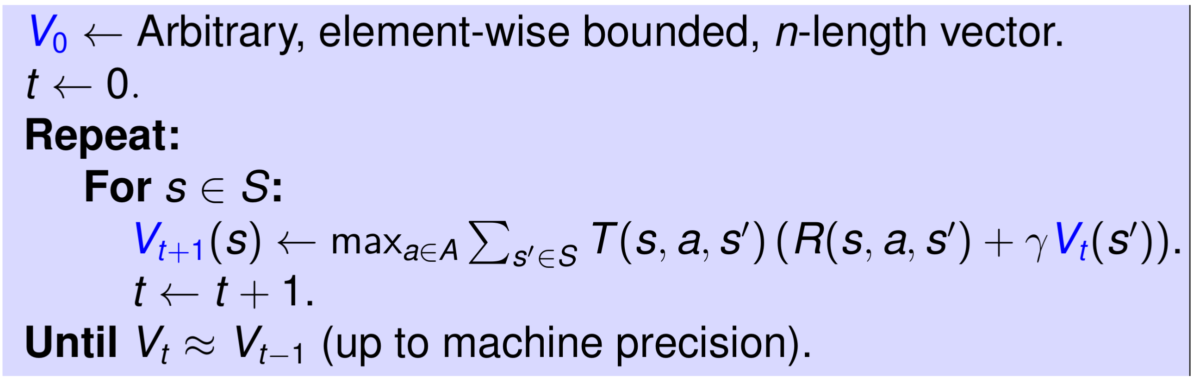
CS747 - Assignment 2

Name - Utkarsh Ranjan
Roll No. - 200050147

Task 1

(1) Value Iteration

- The algorithm used was the value iteration explained in the lecture slides to iteratively compute V^*
- $V_0 \overset{B^*}{\rightarrow} V_1 \overset{B^*}{\rightarrow} V_2 \overset{B^*}{\rightarrow} \dots$



```
$V_0 \leftarrow$ Arbitrary, element-wise bounded, $n$-length vector.
$t \leftarrow 0$.
Repeat:
    For $s \in S$:
        $V_{t+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V_t(s')\right)$.
        $t \leftarrow t + 1$.
Until $V_t \approx V_{t-1}$ (up to machine precision).
```

(2) Policy Evaluation

- The objective of the algorithm was to figure out the value function corresponding to a given policy and the MDP.
- To solve this problem we first formed, n-equations, with n-unknowns, using the bellman equation.
- For γ < 1, the equations were guaranteed to have a unique solutions.

$$V^\pi(s) = \sum_{s \in S} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^\pi(s')\}$$

- These linear equations were converted into matrix form as these:

$$\begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix} = \begin{bmatrix} R_1^\pi \\ \vdots \\ R_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} P_{11}^\pi & \dots & P_{1n}^\pi \\ \vdots & \ddots & \vdots \\ P_{n1}^\pi & \dots & P_{nn}^\pi \end{bmatrix} \begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix}$$

- Here R vector is element-wise product of T and R (in Bellman Equation) summed across the third dimension s'.
- P matrix is same as T matrix in our Bellman Equation
- Thus this problem can be converted into the form Ax = B and can be solve using np.linalg

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v},$$

(3) Howard's Policy Iteration

- The following algorithm (as per the slides) was implemented as a part of this algorithm

```
choose an arbitrary policy  $\pi'$
loop
          $\pi := \pi'$
          compute the value function of policy  $\pi$ :
                    solve the linear equations
                              $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$
          improve the policy at each state:
                    $\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s')\right)$
until  $\pi = \pi'$
```

- The solution of value function for a given policy was done by invoking the policy evaluation function.
- After we have the value function, policy improvement was performed for every state (By choosing the most optimal policy)
- Then It was checked if further improvement is possible or not, If further improvement won't be possible policy in the next loop would remain the same and we would break out of the loop.

(4) Linear Programming

- The linear programming problem was solved using the PuLP library.
- The linear programming formulation for the continuing task was done as per the following constraints mentioned in the slides :-

$$\text{Maximize}\left(-\sum_{s \in S} V(s)\right)$$
$$\text{subject to}$$
$$V(s) \geq \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V(s')\}, \forall s \in S, a \in A$$

- For Episodic task an additional constraint was added :-

$$V(s_{terminal}) = 0$$

Task 2

- If the file cricket states file had **n** states, I created **2n+2** states in my MDP
- These were each of the n states twice, once with Player A on strike and another with Player B, on strike.
- Two states - won and lost states were also made additionally.
- The transition probabilities for each of the states with player A, were decided using the parameters file, while for the states with player B, it was decided using the argument q.

- Reward for any transition which goes to the won state was made 1, for every other transition it was 0.

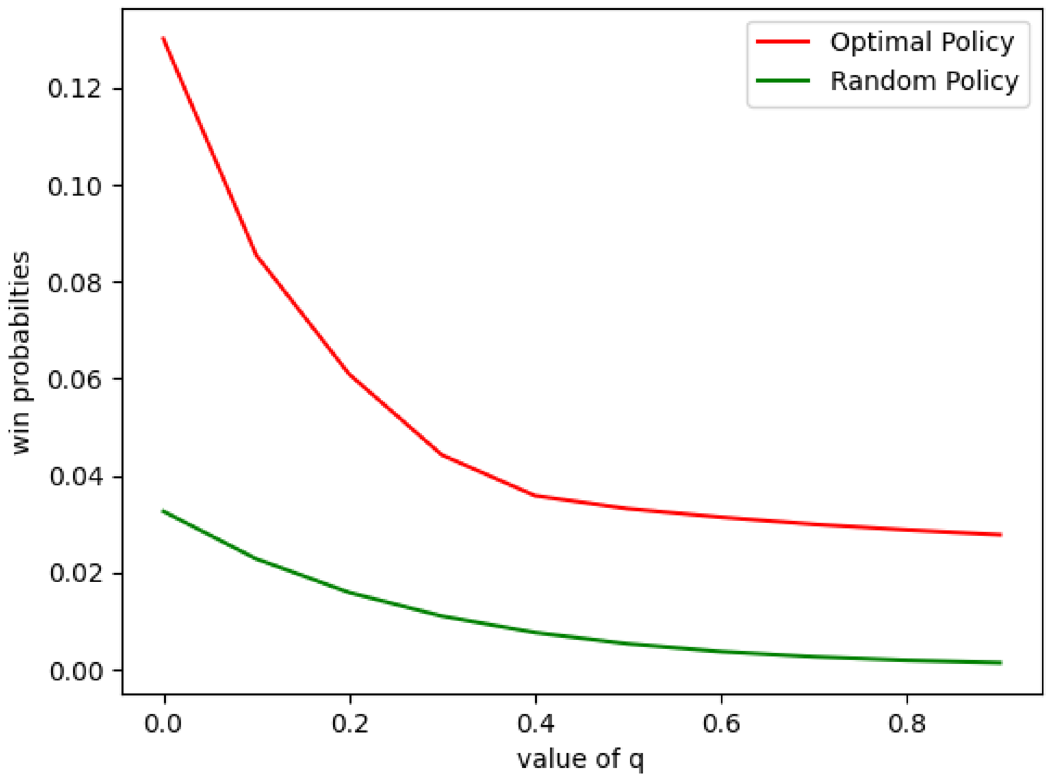
Logic

```
if nextRun == -1 or (balls == 1 and runs - nextRun > 0):
    T[stateNo, action, lost_state] += prob
elif runs - nextRun <= 0:
    T[stateNo, action, win_state] += prob
    R[stateNo, action, win_state] = 1
else:
    strikeChange = False
    if(nextRun % 2 == 1 or (balls % 6 == 1)):
       strikeChange = True

    if(nextRun % 2 == 1 and (balls % 6 == 1)):
       strikeChange = False

    if strikeChange:
       T[stateNo, action, states[(balls - 1, runs - nextRun, 1- player)]] += prob
    else:
       T[stateNo,action,states[(balls - 1, runs - nextRun, player)]] += prob
```

Graphs

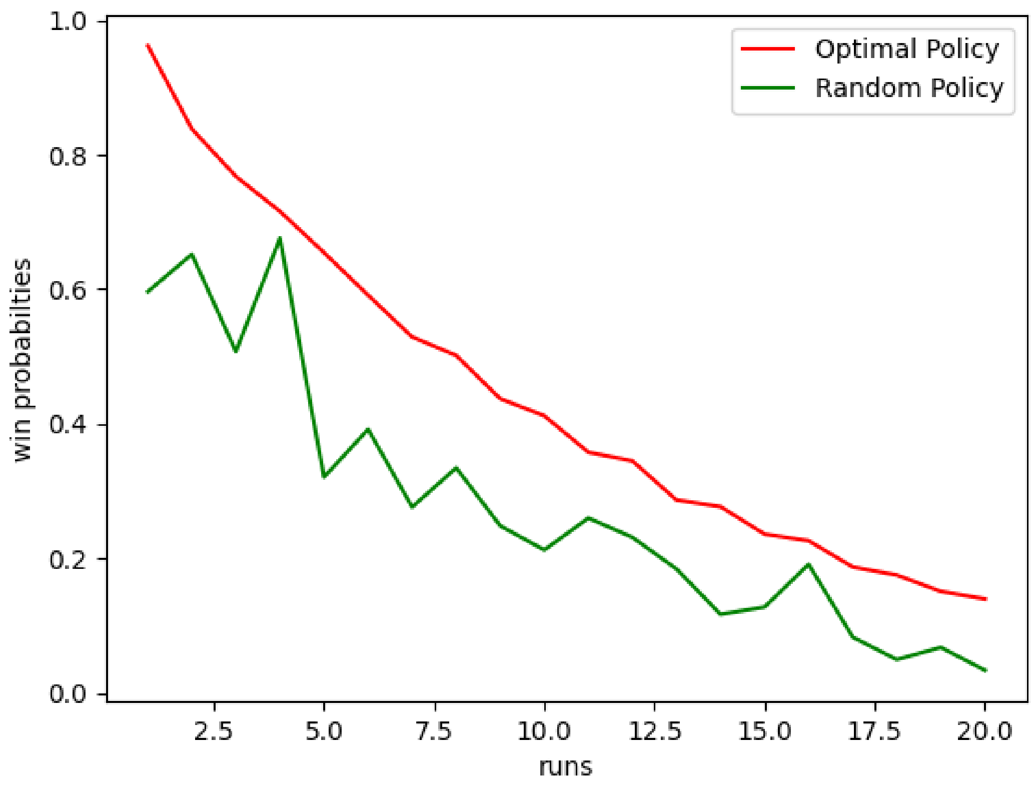
Yes, trends match with the intuition.
Optimal Policy is able to produce better results than the baseline they have provided.

Plot 1



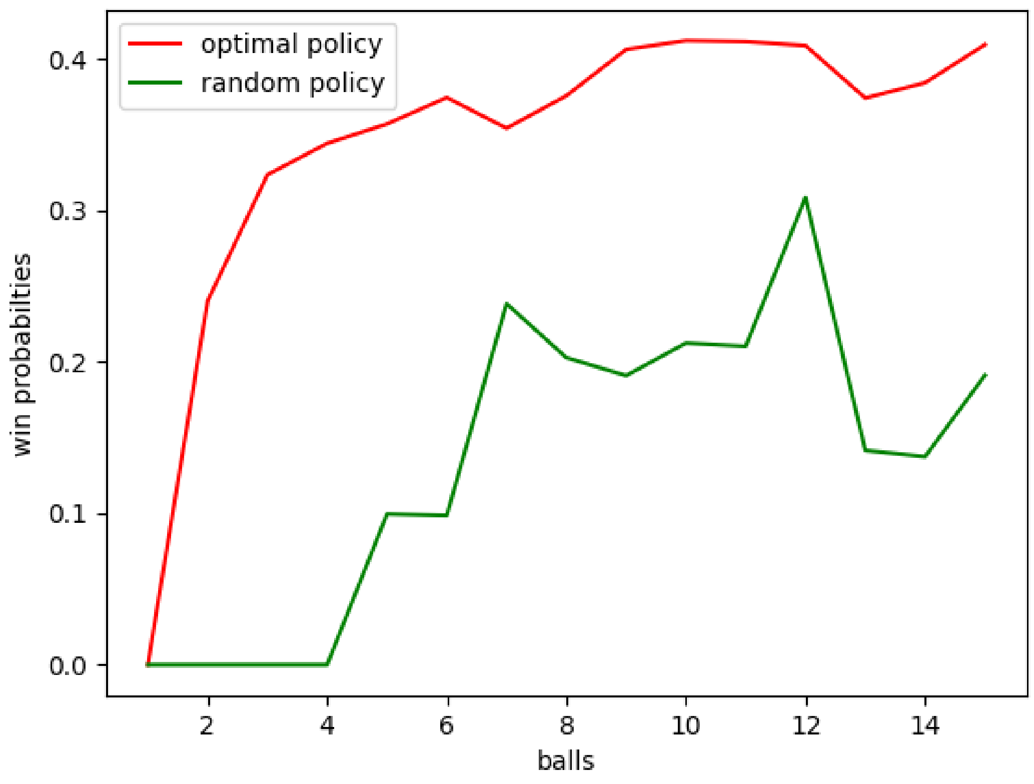
As the value of q increases, the chances of player B, getting out increases which reduces our probability to win the match. Also compared to a random policy. optimal policy always have higher probability to win the match.

Plot 2



It can be clearly seen that the optimal policy performs better than the random policy, for each runs required to win, for a fixed number of balls. This difference in probability reduces as the number of runs increases because, when there is a need to make more runs, policy has less role to play, because the chances of winning is already low and the optimal policy doesn't have much to improve upon the random policy.

Plot 3



Here also, for a given number of runs required to win, as the number of balls increase the probability to win increase, but its always higher for the optimal policy compared to the random policy. When we had to make runs in less number of balls, random policy was performing very poor, while optimal policy still has some probability to win the game.