Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Reinforcement Learning

AY: 2022 - 23

Experiment 4

Aim: To solve the Gambler's Problem using the Value Iteration technique of Dynamic Programming.

Theory:

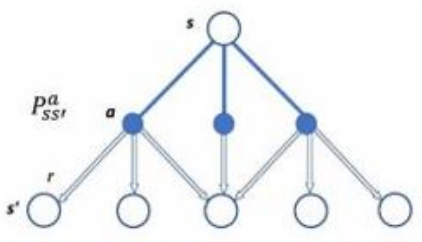
Within Reinforcement Learning, dynamic programming (DP) can be described as a collection of algorithms that can be used to compute optimal policies iteratively, given a perfect model of the environment as a Markov Decision Process (MDP).
In Markov Decision Process (MDP), we assume that the state, action, and reward sets, S, A(s), and R, for s ∈ S, are finite, and that its dynamics are given by a set of probabilities p(s,r|s,a), for all s ∈ S, a ∈ A(s), r ∈ R, and s∈ S+ (S+ is S plus a terminal state if the problem is episodic).

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.
DP algorithms are obtained by turning Bellman equations into assignments, that is, into update rules for improving approximations of the desired value functions.

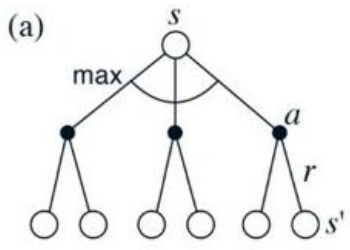
Value Iteration:
In value iteration, we compute the optimal state value function by iteratively updating the estimate. We start with a random value function. At each step, we update it. Hence, we look-ahead one step and go over all possible actions at each iteration to find the maximum.

Formula used in Value Iteration:

$$V(s) = \max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$$



Policy Iteration



Value Iteration

Gambler's Problem:

A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of $100, or loses by running out of money.

On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP.

The state is the gambler's capital, s ∈ {1, 2, . . . , 99}. The actions are stakes, a ∈ {0, 1, . . . , min(s, 100 − s)}. The reward is zero on all transitions except those on which the gambler reaches his goal, when it is +1.

The state-value function then gives the probability of winning from each state. A policy is a mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal. Let p_h denote the probability of the coin coming up heads. If p_h is known, then the entire problem is known and it can be solved, for instance, by value iteration.

Algorithm:

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Department of Computer Science and Engineering (Data Science)

Lab Assignment to do:

1. Implement Value Iteration for the Gambler's Problem
2. Solve and generate graphs for value estimate and final policy vs capital when p_h = 0.4
3. Solve the problem and show the results graphically for p_h = 0.25 and p_h = 0.55

Hint:

Introduce two dummy states corresponding to termination with capital of 0 and 100, giving them values of 0 and 1 respectively.

RL LAB-4

Sarvagya Singh

60009200030 – K1

RL LAB4

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
def value_iteration_for_gamblers(p_h, theta=0.0001, discount_factor=1.0):
    rewards = np.zeros(101)
    rewards[100] = 1
    V = np.zeros(101)

    def one_step_lookahead(s, V, rewards):
        A = np.zeros(101)
        stakes = range(1, min(s, 100-s)+1)
        for a in stakes:
            A[a] = p_h * (rewards[s+a] + V[s+a]*discount_factor) + (1-p_h) * (rewards[s-a] + V[s-a]*discount_factor)
        return A

    while True:
        delta = 0
        for s in range(1, 100):
            A = one_step_lookahead(s, V, rewards)
            best_action_value = np.max(A)
            delta = max(delta, np.abs(best_action_value - V[s]))
            V[s] = best_action_value
        if delta < theta:
            break

    policy = np.zeros(100)
    for s in range(1, 100):
        A = one_step_lookahead(s, V, rewards)
        best_action = np.argmax(A)
        policy[s] = best_action
    return policy, V
```

p_h=0.4

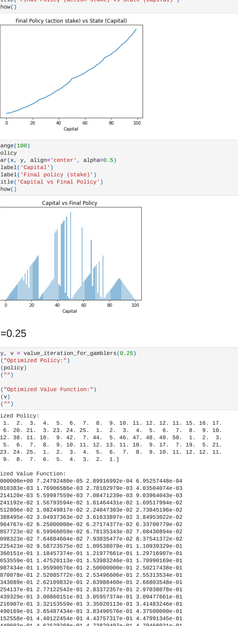
In [ ]:
```
policy, v = value_iteration_for_gamblers(0.4)
print("Optimized Policy:")
print(policy)
print("")

print("Optimized Value Function:")
print(v)
print("")
```

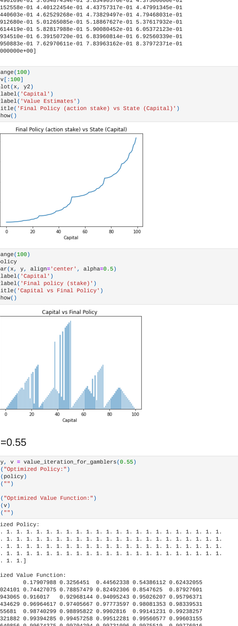
```
Optimized Policy:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 11. 15.  9. 17.
  7. 19. 20. 21. 22. 23. 24. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.
 11. 12. 38. 39. 40.  9.  8. 43. 44. 45.  4. 47.  2.  1. 50.  1.  2.  3.
  4.  5.  6.  7.  8. 41. 10. 11. 12. 13. 14. 15. 34.  8. 10. 19. 20.  4.
 22.  2.  1. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 11.
 10.  9.  8.  7.  6.  5.  4.  3.  2.  1.]

Optimized Value Function:
[0.         0.00203152 0.00515507 0.00925512 0.01290418 0.01738208
 0.02306279 0.02781403 0.03227457 0.03767825 0.04346082 0.05035153
 0.05707757 0.06521897 0.06952507 0.07442925 0.08068842 0.08660695
 0.09421092 0.10313138 0.10865755 0.11596417 0.12507803 0.1335785
 0.1441471  0.16         0.16309394 0.16774251 0.17383767 0.17936474
 0.18607649 0.19459454 0.20172104 0.20841305 0.21652655 0.22519453
 0.2355273  0.24648826 0.25785582 0.2643826  0.27164589 0.28183263
 0.28991593 0.30131638 0.31471349 0.32298754 0.33394956 0.3488281
 0.36038974 0.37622184 0.4        0.40309304 0.40774251 0.41383767
 0.41936474 0.42607649 0.43459454 0.44172104 0.44841305 0.45652655
 0.46519453 0.4755273  0.48648826 0.49785582 0.5043826  0.51164589
 0.52103263 0.52991593 0.54131638 0.55471349 0.56298754 0.57394956
 0.5880281  0.60036974 0.61622184 0.64         0.6446455  0.65161885
 0.66075673 0.66904783 0.67911672 0.69189296 0.70258156 0.71261958
 0.72478983 0.73779252 0.75325684 0.7697331  0.78078873  0.79645494
 0.80747003 0.82154894 0.8348739  0.85197811 0.87207238 0.88448202
 0.90092434 0.92324343 0.9405546  0.96433276 0.         ]
```

In [ ]:
```
x = range(100)
y1 = v[:100]
plt.plot(x, y1)
plt.xlabel('Capital')
plt.ylabel('Value Estimates')
plt.title('Final Policy (action stake) vs State (Capital)')
plt.show()
```



In [ ]:
```
x = range(100)
y = policy
plt.bar(x, y, align='center', alpha=0.5)
plt.xlabel('Capital')
plt.ylabel('Final policy (stake)')
plt.title('Capital vs Final Policy')
plt.show()
```



p_h=0.25

In [ ]:
```
policy, v = value_iteration_for_gamblers(0.25)
print("Optimized Policy:")
print(policy)
print("")

print("Optimized Value Function:")
print(v)
print("")
```

```
Optimized Policy:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 12. 11. 15. 16. 17.
 18.  6. 20. 21.  3. 23. 24. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.
 11. 12. 38. 11. 10.  9. 42.  7. 44.  5. 46. 47. 48. 49. 50.  1.  2.  3.
  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 11. 10.  9. 17.  7. 19.  5. 21.
 22. 23. 24. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 12. 11.
 10.  9.  8.  7.  6.  5.  4.  3.  2.  1.]

Optimized Value Function:
[0.00000000e+00 7.24792480e-05 2.89916992e-04 6.95257448e-04
 1.16016383e-03 1.76906586e-03 2.78192979e-03 4.03504074e-03
 4.66214120e-03 5.59997559e-03 7.08471239e-03 9.03964843e-03
 1.11241192e-02 1.56793594e-02 1.61464431e-02 1.69517994e-02
 1.86512086e-02 2.19524981e-02 2.24047303e-02 2.73845196e-02
 2.83388495e-02 3.04937363e-02 3.61633897e-02 3.84953022e-02
 4.44964767e-02 6.25000000e-02 6.27174377e-02 6.33700779e-02
 6.45857723e-02 6.59966659e-02 6.78135343e-02 7.08430894e-02
 7.46098323e-02 7.64846464e-02 7.93035477e-02 8.37541372e-02
 8.96225423e-02 9.58723575e-02 1.00538078e-01 1.10939329e-01
 1.13308151e-01 1.18457374e-01 1.21977651e-01 1.29716987e-01
 1.44653559e-01 1.47520113e-01 1.53983246e-01 1.70990169e-01
 1.77887434e-01 1.95999576e-01 2.50000000e-01 2.50217438e-01
 2.50870078e-01 2.52085772e-01 2.53496606e-01 2.55313534e-01
 2.58343009e-01 2.62109832e-01 2.63988406e-01 2.66803548e-01
 2.71254137e-01 2.77122542e-01 2.83372357e-01 2.97038078e-01
 2.98439323e-01 3.00800151e-01 3.05957374e-01 3.09477861e-01
 3.17216907e-01 3.32135559e-01 3.35020113e-01 3.41483246e-01
 3.58090169e-01 3.65487434e-01 3.83499576e-01 4.37500000e-01
 4.38152558e-01 4.40122454e-01 4.43757317e-01 4.47991345e-01
 4.53448693e-01 4.62529268e-01 4.73829497e-01 4.79468831e-01
 4.87912680e-01 5.01265885e-01 5.18867627e-01 5.37617932e-01
 5.78614419e-01 5.82837988e-01 5.90088452e-01 6.05372123e-01
 6.15934510e-01 6.39156720e-01 6.83960814e-01 6.92560339e-01
 7.11950883e-01 7.62970611e-01 7.83963162e-01 8.37923731e-01
 0.00000000e+00]
```

In [ ]:
```
x = range(100)
y2 = v[:100]
plt.plot(x, y2)
plt.xlabel('Capital')
plt.ylabel('Value Estimates')
plt.title('Final Policy (action stake) vs State (Capital)')
plt.show()
```

In [ ]:
```
x = range(100)
y = policy
plt.bar(x, y, align='center', alpha=0.5)
plt.xlabel('Capital')
plt.ylabel('Final policy (stake)')
plt.title('Capital vs Final Policy')
plt.show()
```

p_h=0.55

In [ ]:
```
policy, v = value_iteration_for_gamblers(0.55)
print("Optimized Policy:")
print(policy)
print("")

print("Optimized Value Function:")
print(v)
print("")
```

```
Optimized Policy:
[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1.]

Optimized Value Function:
[0.         0.17907988 0.3256451  0.44562338 0.54386112 0.62432055
 0.69024101 0.74427875 0.78857479 0.82402366 0.8547625  0.87927681
 0.89943065 0.916017   0.92968144 0.94095243 0.95026207 0.95790371
 0.96434629 0.96964417 0.97405667 0.97773597 0.98081353 0.98339531
 0.9856581  0.98740299 0.98895822 0.9902816  0.99141233 0.99238257
 0.99321982 0.99394285 0.99457258 0.99513261 0.99560577 0.99603155
 0.99640856 0.99674375 0.99704294 0.99731096 0.9975519  0.99776916
 0.99795564 0.99814377 0.99830564 0.99845362 0.99858745 0.99871025
 0.99882255 0.99892537 0.99901957 0.99910594 0.99918515 0.99925782
 0.99932449 0.99938566 0.99944178 0.99949324 0.99954041 0.99958363
 0.9996232  0.99965942 0.99969253 0.99972278 0.99975041 0.9997759
 0.99979853 0.99981941 0.99983838 0.99985561 0.99987123 0.99988537
 0.99989815 0.9999097  0.99992011 0.99992947 0.99993789 0.99994544
 0.9999522  0.99995825 0.99996364 0.99996844 0.99997271 0.99997649
 0.99997983 0.99998278 0.99998538 0.99998766 0.99998965 0.99999139
 0.9999929  0.99999421 0.99999534 0.99999631 0.99999714 0.99999785
 0.99999845 0.99999895 0.99999937 0.99999972 0.         ]
```

In [ ]:
```
x = range(100)
y3 = v[:100]
plt.plot(x, y3)
plt.xlabel('Capital')
plt.ylabel('Value Estimates')
plt.title('Final Policy (action stake) vs State (Capital)')
plt.show()
```


In [ ]:
```
x = range(100)
y = policy
plt.bar(x, y, align='center', alpha=0.5)
plt.xlabel('Capital')
plt.ylabel('Final policy (stake)')
plt.title('Capital vs Final Policy')
plt.show()
```


ALL

In [ ]:
```
plt.plot(y1)
plt.plot(y2)
plt.plot(y3)
plt.legend(["0.4","0.25","0.55"])
plt.xlabel('Capital')
plt.ylabel('Value Estimation')
plt.show()
```


In [ ]: