Assignment 2

Introduzione al problema

- Gamler’s problem:
  - Un giocatore ha la possibilità di scommettere sui risultati di una sequenza di lanci di monete
  - Se esce testa, vince tanti dollari quanti ne ha puntati in quel lancio. Se esce croce, perde la sua puntata.
  - Il gioco finisce quando arriva a 100$ o perde tutti i soldi
  - Reward +1 se vince e -1 se perde

Iterative policy evaluation

- Predizione: data una policy stimarne il valore
- Bellman come regola d'aggiornamento
- Aggiornamento in-place

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}^+, r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v_k(s'\right.$$

```
#policy evaluation
while True:
    while True:
        delta = 0
        for s in range(1,goal):
            v = V[s]
            V[s]= V[s] = p * (R[min(goal, s + pi[s])] + gamma * V[min(goal, s + pi[s])]) \
                        + (1 - p) * (R[max(0, s - pi[s])] + gamma * V[max(0, s - pi[s])])
            delta=max(delta,abs(v-V[s]))
        if delta<theta:
            break
```

Policy improvement

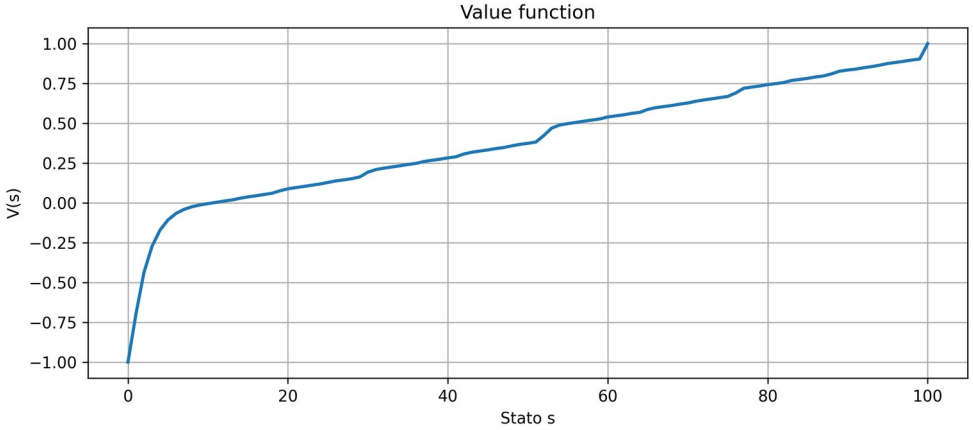
- Policy improvement theorem
- Miglioramento greedy della policy

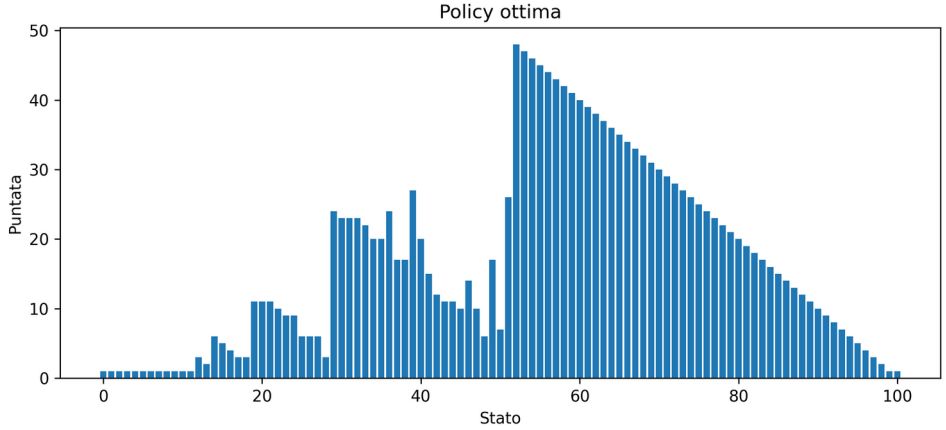
$$\pi'(s) = \arg\max_a q_\pi(s,a)$$
$$= \arg\max_a \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right)$$

```
#policy improvement
stable=True
for s in range(1, goal):
    x = -np.inf
    old_a=pi[s]
    max_a = 0
    for a in range(1, min(s, goal - s) + 1):
        y = p * (R[s + a] + gamma * V[s + a]) + \
            (1-p) * (R[max(0, s - a)] + gamma * V[max(0, s - a)])
        if y > x:
            x = y
            max_a = a
    pi[s] = max_a
    if old_a!=pi[s]:
        stable=False
if stable:
    break
```

Policy iteration

- Alternando evaluation e improvement otteniamo una sequenza di funzioni valore e policy in continuo miglioramento





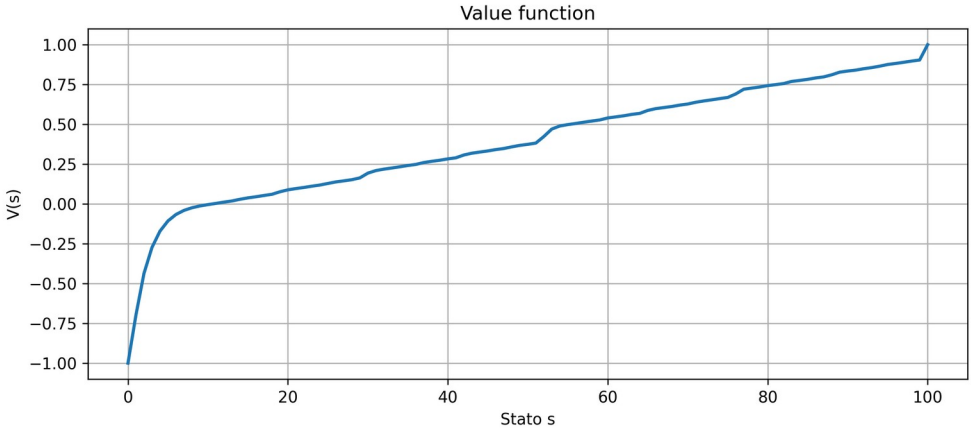
Value iteration

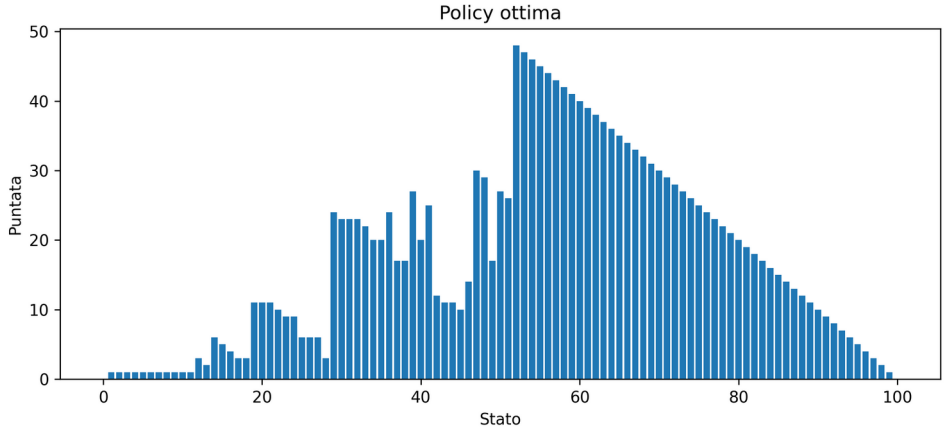
- Ci si ferma dopo un singolo sweep tra gli stati per policy evaluation
- Principio di ottimalità di Bellman
- L'aggiornamento combina evaluation § e improvement

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_k(s')\right)$$

```
#value iteration algorithm
while True:

    delta = 0
    for s in range(1,goal):
        v = V[s]
        A = np.zeros(min(s,goal-s))#ha senso puntare solo fino all'obiettivo
        for a in range(1,min(s,goal-s)+1):
            A[a-1]=(p*(R[s+a]+gamma*V[s+a])+
                    (1-p)*(R[max(0,s-a)]+gamma*V[max(0,s-a)])) #vincita + perdita
        V[s]=max(A)
        delta=max(delta,abs(v-V[s]))
    if delta<theta:
        break

pi = np.zeros(goal + 1, dtype=int)

for s in range(1, goal):
    x = -np.inf
    max_a = 0
    for a in range(1, min(s, goal - s) + 1):
        y = 0.5 * (R[s + a] + gamma * V[s + a]) + \
            0.5 * (R[max(0, s - a)] + gamma * V[max(0, s - a)])
        if y > x:
            x = y
            max_a = a
    pi[s] = max_a
```

Value iteration





FINE