DYNAMIC PROGRAMMING

ASSIGNMENT 2

LUCA SUGAMOSTO , MATRICOLA 0324613

MATTIA QUADRINI , MATRICOLA 0334381

PROF. CORRADO POSSIERI

OBIETTIVO DEL PROGETTO

Dato il modello di gioco $problema\ dello\ scommettitore$, un giocatore d'azzardo ha la probabilità di scommettere sui risultati di una sequenza di lanci di una moneta:

- Se esce TESTA, il giocatore vince tanti dollari quanti ne ha scommessi in quel lancio;
- Se esce CROCE il giocatore perde la somma di denaro puntata.

Il gioco termina quando il giocatore d'azzardo vince (raggiungendo l'obiettivo di $100\$$) o perde (rimanendo senza soldi). Il reward è così assegnato:

- $+1$ in caso di vittoria;
- -1 in caso di sconfitta;
- 0 in caso si avesse una somma di denaro nel mezzo.

Successivamente analizzare i risultati ottenuti e comparare il comportamento e le differenze dei due algoritmi usati

CONSIDERAZIONI PRELIMINARI

- È utilizzato un MDP (**Markov Decision Process**); questa condizione permette di assumere che l'ambiente è totalmente osservabile e cioè sono note le probabilità di transizione di stato $p(s'|s,a)$.
  Per l'ambiente vale la **proprietà di Markov**, cioè lo stato successivo S_{t+1} dipende soltanto dallo stato corrente S_t in quanto questo tiene traccia di tutta storia passata : $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, \dots, S_t]$.

- Si considerano dei **tasks episodici** poiché sono presenti degli stati terminali dove convergere. Il tempo di terminazione T è una variabile che cambia da episodio ad episodio. Ogni episodio terminato in uno stato terminale è seguito da un reset.

- La **funzione valore** di uno stato s sotto una policy π è il ritorno atteso quando si parte da s e seguendo π successivamente : $v_\pi(s) = E_\pi[G_t|S_t = s]$.

- La **funzione qualità** di uno stato s e di un'azione a sotto una policy π è il ritorno atteso quando si parte da s, si prende l'azione a e seguendo π successivamente : $q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$.

MATRICE DELLE PROBABILITÀ DI TRANSIZIONE P

- La massima quantità di denaro che si può vincere è $100\$$, quindi il **numero totale di stati** è 101 (si considera anche lo stato con $0\$$) mentre il **numero totale di azioni** è 99 (non si considera l'azione di giocare $100\$$).

- La moneta lanciata per determinare la vittoria o la sconfitta **non è truccata**, quindi si ha la stessa probabilità (50%) che esca testa o croce.

- La **matrice di probabilità di transizione P** è costruita in modo che se si considera lo stato s e l'azione a, con $a > s$, allora il giocatore punta tutto il denaro posseduto nello stato s.

- Per esempio se si considera un deposito di massimo $10\ \$$, quindi 11 stati e 9 azioni e si sceglie l'azione 'punta $3\ \$$', allora la matrice P è definita come segue:

$stati$

$stati$

```
val(:,:,3) =

    1.0000         0         0         0         0         0         0         0         0         0         0
    0.5000         0    0.5000         0         0         0         0         0         0         0         0
    0.5000         0         0         0    0.5000         0         0         0         0         0         0
    0.5000         0         0         0         0         0    0.5000         0         0         0         0
         0    0.5000         0         0         0         0         0    0.5000         0         0         0
         0         0    0.5000         0         0         0         0         0    0.5000         0         0
         0         0         0    0.5000         0         0         0         0         0    0.5000         0
         0         0         0         0    0.5000         0         0         0         0         0    0.5000
         0         0         0         0         0    0.5000         0         0         0         0    0.5000
         0         0         0         0         0         0    0.5000         0         0         0    0.5000
         0         0         0         0         0         0         0         0         0         0    1.0000
```

```
%Inizializzazione della matrice delle probabilità di transizione P
%-----------------------------------------------------------------
P = zeros(S, S, A);

for s = 1:S                 %per ogni stato appartenente ad S
    [numRow,numCol] = ind2sub([S S], s);          %numRow indica lo stato s-esimo
    numRow = numRow - 1;                          %denaro effettivo in deposito

    for a = 1:A             %per ogni azione appartenente ad A
        %La giocata effettuata sarà l'azione 'a' se questa è minore o
        %uguale al denaro posseduto, mentre sarà il massimo denaro nello
        %stato 's' se si considera un'azione maggiore del denaro posseduto.

        if (numRow == 0 || numRow == maxWin)
            %Caso in cui mi trovo in uno stato terminale.
            %Indipendentemente dall'azione scelta torno sempre in esso
            newNumRow = numRow + 1;
            next_s = sub2ind([S S], newNumRow, numCol);    %calcolo dello stato successivo

            %Essendo l'unica transazione possibile quella di tornare nello
            %stesso stato, questa ha probabilità 1 di verificarsi
            P(s, next_s, a) = 1;                          %lo stato attuale si trova sull'indice di riga mentre lo stato successivo sull'indice di colonna
        else
            %Caso in cui mi trovo in uno stato non terminale e quindi con
            %probabilità 50% vado in uno stato, mentre con il 50% vado in
            %un altro

            %la giocata effettiva è il valore minimo tra l'azione scelta
            %'a' ed il denaro effettivamente in deposito
            bet = min(a, numRow);

            newNumRow1 = min((numRow + 1) + bet, S);       %calcolo del nuovo stato in caso di vittoria
            newNumRow2 = max((numRow + 1) - bet, 1);       %calcolo del nuovo stato in caso di sconfitta

            next_s1 = sub2ind([S S], newNumRow1, numCol);  %nuova coordinata dello stato in caso di vittoria
            next_s2 = sub2ind([S S], newNumRow2, numCol);  %nuova coordinata dello stato in caso di sconfitta

            P(s, next_s1, a) = probability;               %lo stato attuale si trova sull'indice di riga mentre lo stato successivo sull'indice di colonna
            P(s, next_s2, a) = probability;               %lo stato attuale si trova sull'indice di riga mentre lo stato successivo sull'indice di colonna
        end
    end
end
```

MATRICE DEI REWARDS R

- Ogni elemento del vettore **earning** corrisponde ad un determinato stato s e ad esso viene assegnato il valore del reward istantaneo:
  1. $+1$ se $s = S$ (100 \$);
  2. -1 se $s = 1$ (0 \$);
  3. 0 *altrimenti*.

  Successivamente, si definisce la **matrice dei rewards R** tramite un prodotto matriciale tra la matrice delle probabilità di transizione ed il vettore dei rewards istantanei.

- Considerando anche qui l'esempio descritto nella diapositiva precedente, si ottiene la seguente matrice R :

azioni

stati

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
3	0	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
4	0	0	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
5	0	0	0	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0.5000	0.5000	0	0	0	0
8	0	0	0.5000	0.5000	0.5000	0.5000	0	0	0
9	0	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0	0
10	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0
11	0	0	0	0	0	0	0	0	0

```
earning = zeros(S,1);       %vettore dei guadagni istantanei che non dipendono dall'azione

for s = 1:S                 %per ogni stato appartenente ad S
    if (s == 1)             %stato che corrisponde ad avere 0$ nel deposito
        earning(s, 1) = -1;
    elseif (s == S)         %stato che corrisponde ad avere 100$ nel deposito
        earning(s, 1) = 1;
    %per tutti gli altri stati intermedi il guadagno istantaneo è pari a 0
    end
end

R = zeros(S, A);
for a = 1:A                 %per ogni azione appartenente ad A
    R(:, a) = P(:, :, a) * earning;
end
%siccome lo stato 1 ed S sono terminali allora il reward assegnato ad essi
%quando viene presa una qualsiasi azione è pari a 0
R(1, :) = 0;
R(S, :) = 0;

save gamblerProblem_data.mat P R        %salvataggio delle matrici che determinano il modello
```

ALGORITMI POSSIBILI PER IL GIOCATORE

- Policy Iteration Algorithm
- Value Iteration Algorithm

Policy Iteration Algorithm

ITERATIVE POLICY EVALUATION

La funzione di $(Fig.1)$ è utilizzata per risolvere il problema di predizione, cioè data una policy π calcolare la funzione valore v_π migliore.

La legge di aggiornamento che permette di risolvere tale problema è :
$v_{\pi,k+1} = R + (\gamma * P * v_{\pi,k}) \quad \forall s \in \mathcal{S}.$
Questa viene applicata ciclicamente finché non è rispettata una condizione di uscita, determinata da un parametro θ detto $valore\ di\ soglia.$

L'aggiornamento di v_π è $in-place$, quindi si utilizza una singola variabile v che viene sovrascritta ripetutamente (ciò garantisce una convergenza più veloce).

L'uscita della funzione (v_π) viene passata in ingresso ad un'altra, che ha l'obiettivo di calcolare una nuova policy migliore π', se esiste.

```
function obj = iterativePolicyEvaluation(obj, matrixP, matrixR, Case)
    %funzione per il calcolo della stima della funzione valore Vpi
    %dati in ingresso la policy "pi" e il valore di soglia "theta"
    Ppi = zeros(obj.S, obj.S);          %matrice P associata alla policy in ingresso
    Rpi = zeros(obj.S, 1);              %vettore R associatao alla policy in ingresso

    %inserimento di nuovi valori all'interno sia di Ppi sia di Rpi
    for s = 1:obj.S                 %per ogni stato dell'insieme S
        a = obj.pi(s);              %azione dettata dalla policy "pi" se ci si trova nello stato "s"

        %"squeeze()" seleziona il vettore riga di P associato alla
        %riga s-esima e all'azione a-esima
        Ppi(s, :) = squeeze(matrixP(s, :, a));
        Rpi(s) = matrixR(s, a);
    end

    %calcolo della stima della funzione valore Vpi associata a "pi"
    if (Case == 0)
        %caso in cui si utilizza il "iterativePolicyEvaluation"
        %prima del loop e quindi considero la funzione iniziale V0
        %come la stima della funzione valore da usare e migliorare
        value = obj.V0;
    elseif (Case == 1)
        %caso in cui si utilizza il "iterativePolicyEvaluatio"
        %all'interno del loop e quindi considero la funzione Vpi
        %calcolata precedentemente come la stima della funzione
        %valore da usare e migliorare
        value = obj.Vpi;
    end

    while true
        nextValue = Rpi + ((obj.gamma .* Ppi) * value);
        if (norm((nextValue - value), "inf") < obj.thresholdValue)
            %condizione di uscita dal loop poichè la funzione
            %valore non varia rispetto a quella calcolata prima
            obj.Vpi = nextValue;
            break                   %uscita dal loop
        else
            %si rimane nel loop
            value = nextValue;      %aggiornamento "in-place" poichè si memorizza una sola variabile
        end
    end
    %salvataggio della policy usata per il calcolo della stima
    %della funzione valore Vpi per confrontarla in seguito con
    %la futura nuova policy calcolata con policyImprovement
    obj.prev_pi = obj.pi;
end
```

$(Fig.1)$ Iterative Policy Evaluation

POLICY EVALUATION

Anche questa funzione risolve un problema di predizione, ma a differenza della precedente, restituisce la soluzione in tempi più brevi per il seguente motivo :

- Un singolo passo di policy evaluation, cioè si calcola la nuova stima della funzione valore $v_{\pi,k+1}$ e si esce immediatamente (senza controllare che la funzione valore ottenuta sia migliorata rispetto alle precedenti).

L'uscita della funzione (v_π) viene passata in ingresso ad un'altra, che ha l'obiettivo di calcolare una nuova policy migliore π', se esiste.

```
function obj = policyEvaluation(obj, matrixP, matrixR)
    %funzione che calcola la stima della funzione valore "Vpi" dato
    %in ingresso la policy "pi"
    Ppi = zeros(obj.S, obj.S);          %matrice P associata alla policy in ingresso
    Rpi = zeros(obj.S, 1);              %vettore R associatao alla policy in ingresso

    for s = 1:obj.S
        a = obj.pi(s);                  %azione dettata dalla policy "pi" se ci si trova nello stato "s"

        %"squeeze()" seleziona il vettore riga di P associato alla
        %riga s-esima e all'azione a-esima
        Ppi(s, :) = squeeze(matrixP(s, :, a));
        Rpi(s) = matrixR(s, a);
    end
    I = eye(obj.S);                     %matrice identità quadrata di dimensionr SxS
    obj.Vpi = (I - (obj.gamma .* Ppi)) \ Rpi;

    %salvataggio della policy usata per il calcolo della stima
    %della funzione valore Vpi per confrontarla successivamente con
    %la futura nuova policy calcolata con policyImprovement
    obj.prev_pi = obj.pi;
end
```

($Fig. 2$) Policy Evaluation

POLICY IMPROVEMENT

La funzione in $(Fig.3)$ riceve in ingresso la v_π calcolata in precedenza e la utilizza per risolvere il problema del controllo, cioè cercare una nuova policy migliore π'.

Per fare ciò si determina una stima della funzione qualità $Q_\pi(s,a)$ per ogni coppia stato - azione.
Successivamente, si calcola la nuova policy (per ogni stato del MDP) tramite la seguente legge di aggiornamento :
$\pi'(s) = \arg max_a Q_\pi(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

Questo tipo di aggiornamento della policy garantisce un miglioramento continuo di π, infatti varrà la seguente condizione : $\pi_k \leq \pi_{k+1} \quad \forall k$

La nuova policy π', restituita in uscita alla funzione, è utilizzata per risolvere un nuovo problema di predizione all'istante successivo, se non è essa la policy ottima.

```
function obj = policyImprovement(obj, matrixP, matrixR)
    %funzione per il calcolo della policy "pi*" migliore rispetto a
    %quella precedente, valutando la stima della funzione valore
    %associata alla policy "pi" passata in ingresso
    Q = zeros(obj.S, obj.A);            %inizializzazione della stima della funzione qualità associata ad ogni coppia (stato, azione)
    obj.next_pi = zeros(obj.S, 1);      %inizializzazione della nuova policy "pi*"

    for s = 1:obj.S                 %per ogni stato appartenente ad S
        for a = 1:obj.A             %per ogni azione appartenente ad A
            Q(s, a) = matrixR(s, a) + ((obj.gamma .* matrixP(s, :, a)) * obj.Vpi);
        end
        %calcolo della nuova azione da prendere se ci si trova
        %nello stato s e inserimento di questa nel vettore della
        %policy pi
        if (s == 1 || s == obj.S)
            obj.next_pi(s) = 1;
        else
            obj.next_pi(s) = find(Q(s, :) == max(Q(s, :)), 1, "first");
        end
    end
    obj.pi = obj.next_pi;               %salvataggio della nuova policy calcolata
end
```

$(Fig.3)$ Policy Improvement

POLICY ITERATION *(Algoritmo finale contenente tutte le funzioni precedenti)*

Alternando *policy evaluation* e *policy improvement* si ottiene una sequenza di funzioni valore v_π e policy π che migliorano gradualmente nel tempo. L'algoritmo termina quando confrontando la nuova policy π' con la precedente π, si ha che esse sono uguali.

Non c'è il rischio di avere un ciclo, cioè una policy non può ripresentarsi se si scelgono due azioni diverse, questo perché vale la condizione : $v_{\pi_k} \leq v_{\pi_{k+1}}, \forall k$

NOTA:

Nella funzione descritta, in particolare all'interno del ciclo, si esegue una funzione di policy evaluation tra le due proposte, in base al valore del parametro γ (Fattore di sconto).

Tale parametro indica quanta importanza dare alle funzioni valore calcolate precedentemente, rispetto ai rewards istantanei.

```
function obj = policyIteration(obj, matrixP, matrixR)
    %funzione per il calcolo della policy ottima eseguendo in modo
    %alternato le funzioni policy evaluation, policy improvement
    obj = iterativePolicyEvaluation(obj, matrixP, matrixR, 0);      %primo passo di policy evaluation
    obj = policyImprovement(obj, matrixP, matrixR);                 %primo passo di policy improvement

    counter = 0;
    while true
        counter = counter + 1;
        fprintf("PI - iterazione n°: ");
        disp(counter)

        %passo di POLICY EVALUATION
        if (obj.gamma < 1)
            obj = policyEvaluation(obj, matrixP, matrixR);
        else
            %poichè per gamma = 1 si hanno problemi con la
            %divisione seguente: Ppi \ Rpi
            obj = iterativePolicyEvaluation(obj, matrixP, matrixR, 1);
        end
        %passo di POLICY IMPROVEMENT
        obj = policyImprovement(obj, matrixP, matrixR);
        if (norm((obj.pi - obj.prev_pi), 2) == 0)
            %caso in cui la policy trovata non è cambiata rispetto
            %alla precedente e quindi si è trovata una policy
            %stabile
            break                           %uscita dal loop
        end
    end
end
```

(*Fig.* 4) Policy Iteration

Value Iteration Algorithm

VALUE ITERATION STEP

Lo svantaggio dell'algoritmo di figura $(Fig.4)$ è dato dalla funzione $iterative\ policy\ evaluation$, poiché richiede più scansioni (algoritmo iterativo) per calcolare la migliore stima della funzione valore, per ogni singolo stato s.

- L'idea è di troncare gli aggiornamenti immediatamente dopo aver calcolato $v_{\pi,k+1}$, determinare quindi una stima della funzione qualità usando $v_{\pi,k+1}$ ed infine assegnare a $v_{\pi,k+1}(s)\ \ \forall s \in \mathcal{S}$, il massimo valore assunto dalla variabile $Q_\pi(s)\ \ \forall a \in \mathcal{A}$.

- Quindi, invece di valutare la policy π, si massimizza direttamente la funzione valore v_π tramite la selezione delle azioni migliori.

```
function obj = valueIterationStep(obj, matrixP, matrixR)
    %funzione che calcola una stimma della funzione valore dopo
    %aver applicato un singolo passo di policy evaluation
    Ppi = zeros(obj.S, obj.S);          %matrice P associata alla policy in ingresso
    Rpi = zeros(obj.S, 1);              %vettore R associatao alla policy in ingresso

    for s = 1:obj.S
        a = obj.pi(s);                  %azione dettata dalla policy "pi" se ci si trova nello stato "s"

        %"squeeze()" seleziona il vettore riga di P associato alla
        %riga s-esima e all'azione a-esima
        Ppi(s, :) = squeeze(matrixP(s, :, a));
        Rpi(s) = matrixR(s, a);
    end

    %singolo passo di policy evaluation
    nextVpi = Rpi + ((obj.gamma .* Ppi) * obj.Vpi);
    %aggiornamento della stima della funzione valore  utilizzando
    %la stima della funzione qualità
    obj.next_Vpi = zeros(obj.S, 1);
    Q = zeros(obj.S, obj.A);            %stima della funzione qualità
    for s = 1:obj.S
        for a = 1:obj.A
            Q(s, a) = matrixR(s, a) + ((obj.gamma .* matrixP(s, :, a)) * nextVpi);
        end
        obj.next_Vpi(s) = max(Q(s, :));
    end
end
```

$(Fig.5)$ Value Iteration Step

VALUE ITERATION *(Algoritmo finale contenente la funzione precedente)*

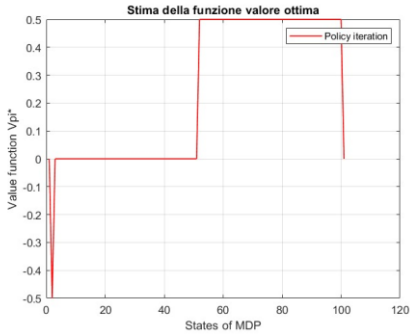
La funzione in figura $(Fig.6)$ contiene al suo interno l'algoritmo *value iteration step*, il quale viene eseguito iterativamente finché non è soddisfatta la condizione di uscita. Quest'ultima è necessaria per definire la convergenza poiché mette a confronto le funzioni valore $v_{\pi,k}, v_{\pi,k+1}$.

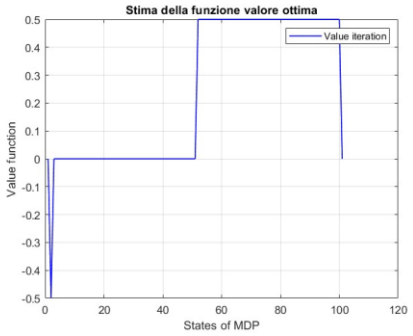
Infine, si calcola la policy ottima π^* considerando la funzione valore ottima v^* restituita dal ciclo per mezzo della funzione *policy improvement* di figura $(Fig.3)$.

```
function obj = valueIteration(obj, matrixP, matrixR)
    %funzione che calcola la policy ottima usando un singolo passo
    %di policy evaluation e il policy improvement
    obj.pi = obj.piForValueIteration;
    obj.Vpi = obj.V0;

    counter = 0;
    while true
        counter = counter + 1;
        fprintf("VI - iterazione n°: ");
        disp(counter)

        %esecuzione dell'algoritmo di value iteration step
        obj = valueIterationStep(obj, matrixP, matrixR);
        %confronto tra la nuova stima della funzione valore
        %calcolata tramite "valueIterationStep" e della vecchia
        %stima della funzione valore "Vpi"
        if (norm((obj.next_Vpi - obj.Vpi), "inf") < obj.thresholdValue)
            obj.Vpi = obj.next_Vpi;
            break
        else
            obj.Vpi = obj.next_Vpi;
        end
    end
    %calcolo della policy ottima per mezzo di policy improvement
    obj = policyImprovement(obj, matrixP, matrixR);
end
```

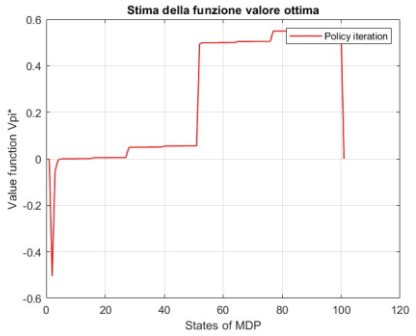
$(Fig.6)$ Value Iteration

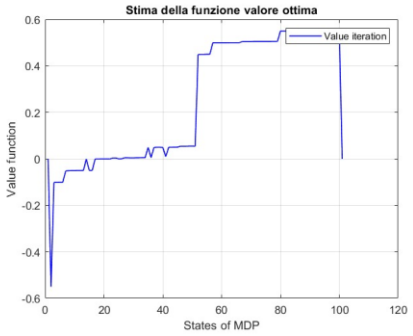
ANALISI DEI RISULTATI OTTENUTI



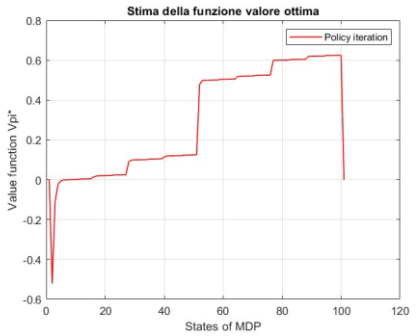


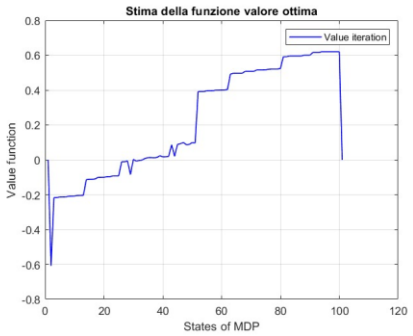
- $\gamma = 0$: In questo caso la stima della funzione valore V_π, dipende solo dalle componenti della matrice dei rewards R quindi è pari al valore atteso dei ritorni istantanei (dato lo stato di partenza).



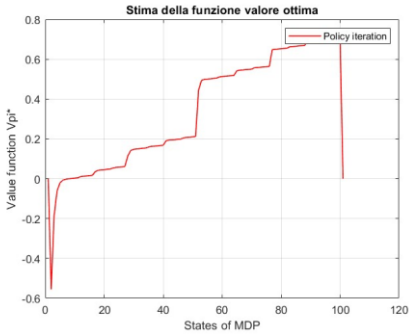


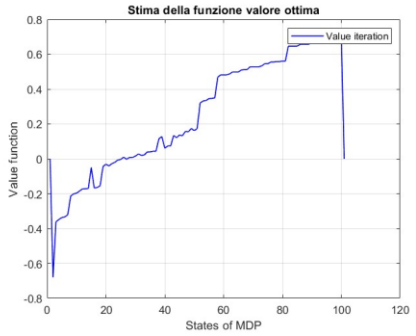
- $\gamma = 0.2$: In questo caso la stima della funzione valore V_π, dipende anche dalla stima della funzione valore all'istante precedente $V_{\pi,k-1}$, quindi vengono considerati anche i risultati inerenti gli altri stati del MDP. Essendo γ basso, l'importanza di questo temine aggiuntivo incide poco sul risultato finale.



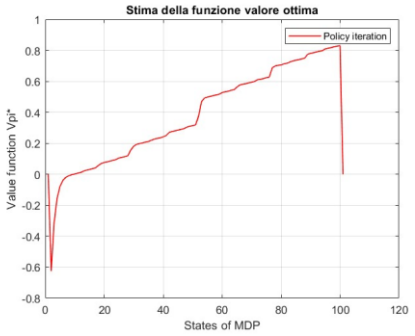


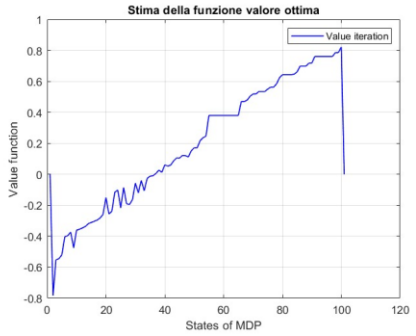
- $\gamma = 0.4$: Vale lo stesso ragionamento del caso precedente, con il parametro che inizia ad aumentare quindi il termine : $\gamma * P * V_{\pi,k-1}$, influisce maggiormente sul calcolo di $V_{\pi,k} = R + (\gamma * P * V_{\pi,k-1})$.



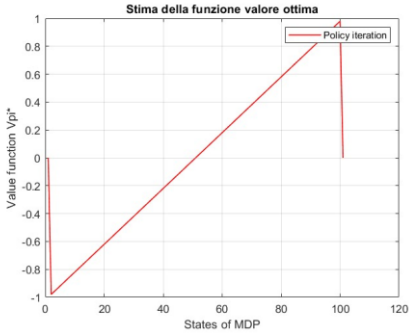


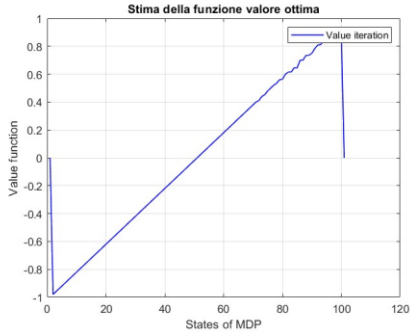
- $\gamma = 0.6$: Ragionamento identico al caso precedente, aumentando la lungimiranza.





- $\gamma = 0.8$: Ragionamento identico al caso precedente, aumentando la lungimiranza.

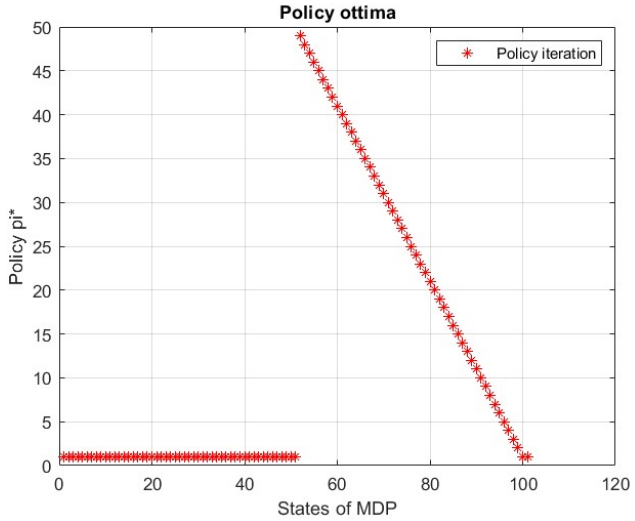


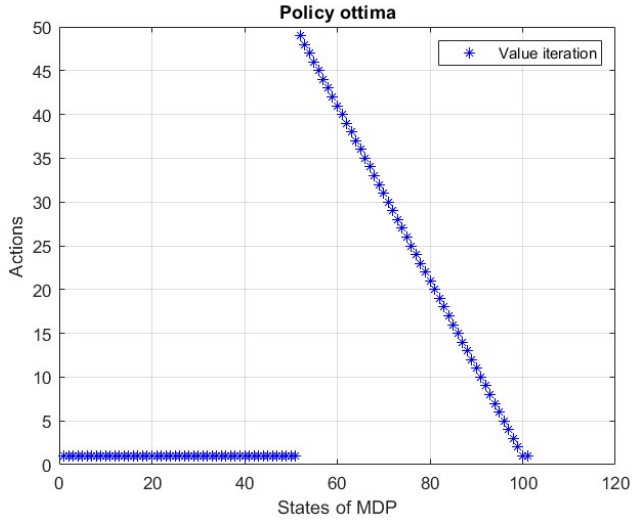


- $\gamma = 1$: In questo caso si ha massimo valore di lungimiranza quindi si dà stessa importanza sia alla matrice dei rewards R istantanei sia alla funzione valore calcolata all'istante precedente $V_{\pi,k-1}$.
  Il risultato, indipendentemente dall'algoritmo usato, è una variazione della funzione valore V_π molto più marcata al variare dello stato ed assume valori compresi nell'intervallo $\{-1, 1\}$.

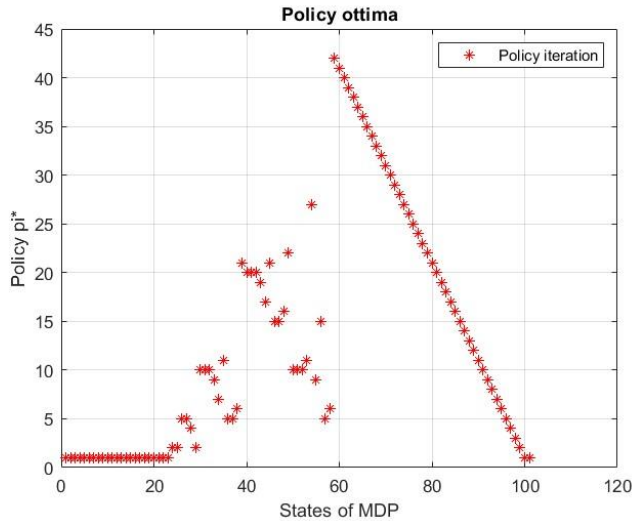
OSSERVAZIONE : Dato che l'algoritmo di $Value\ Iteration$ esegue un solo passo di $policy\ evaluation$, allora si notano maggiori oscillazioni della stima della funzione valore ottima V_π^* rispetto all'algoritmo di $Policy\ Iteration$.

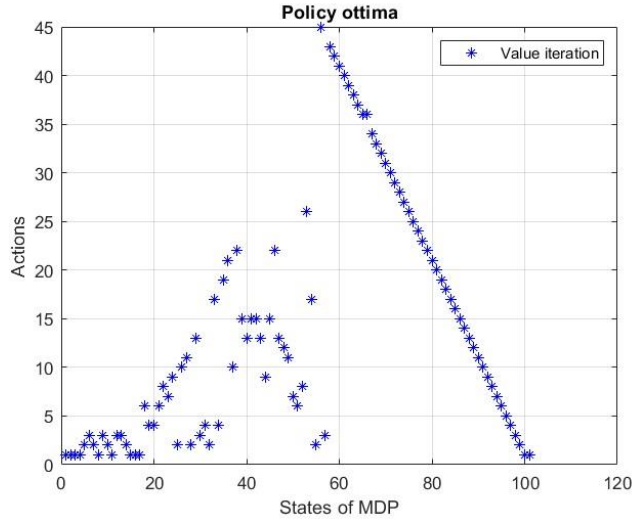
Caso $\gamma = 0$: la funzione valore v_{π^*} dipende soltanto dal reward istantaneo, si scommette 1$ fino ad arrivare allo stato intermedio





Caso $\gamma = 0{,}99$: la lungimiranza è massima, a partire dalla seconda metà si scommette la quantità di denaro mancante per raggiungere la somma desiderata





GRAZIE

LUCA SUGAMOSTO (0324613)

MATTIA QUADRINI (0334381)

luca.sugamosto@students.uniroma2.eu

mattia.quadrini.1509@students.uniroma2.eu