Polytechnics School of ULB
First Master in Computer Engineering
Section: Computational Intelligence

Learning Dynamics

Assignment 2 : Dynamic Programming

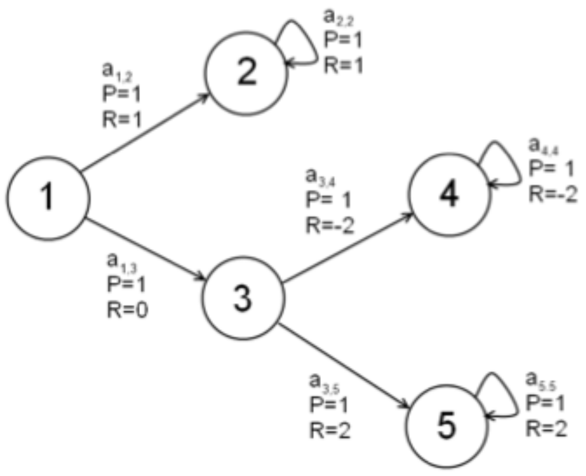
Alexandre Balon-Perin

Academic Year 2010-2011

Table of Contents

Introduction

We were asked in the course of Learning Dynamics to implements two algorithms using Dynamic Programming. Dynamic Programming is a method to solve complex problem by dividing them into smaller steps. The Policy Iteration and Value Iteration algorithm had to be applied on the following model.



There are 5 states with different actions to choose for each of them. Actually, only states 1 and 3 have a choice between two actions. The reward gain by taking an action and the probability to take it are written on the model.

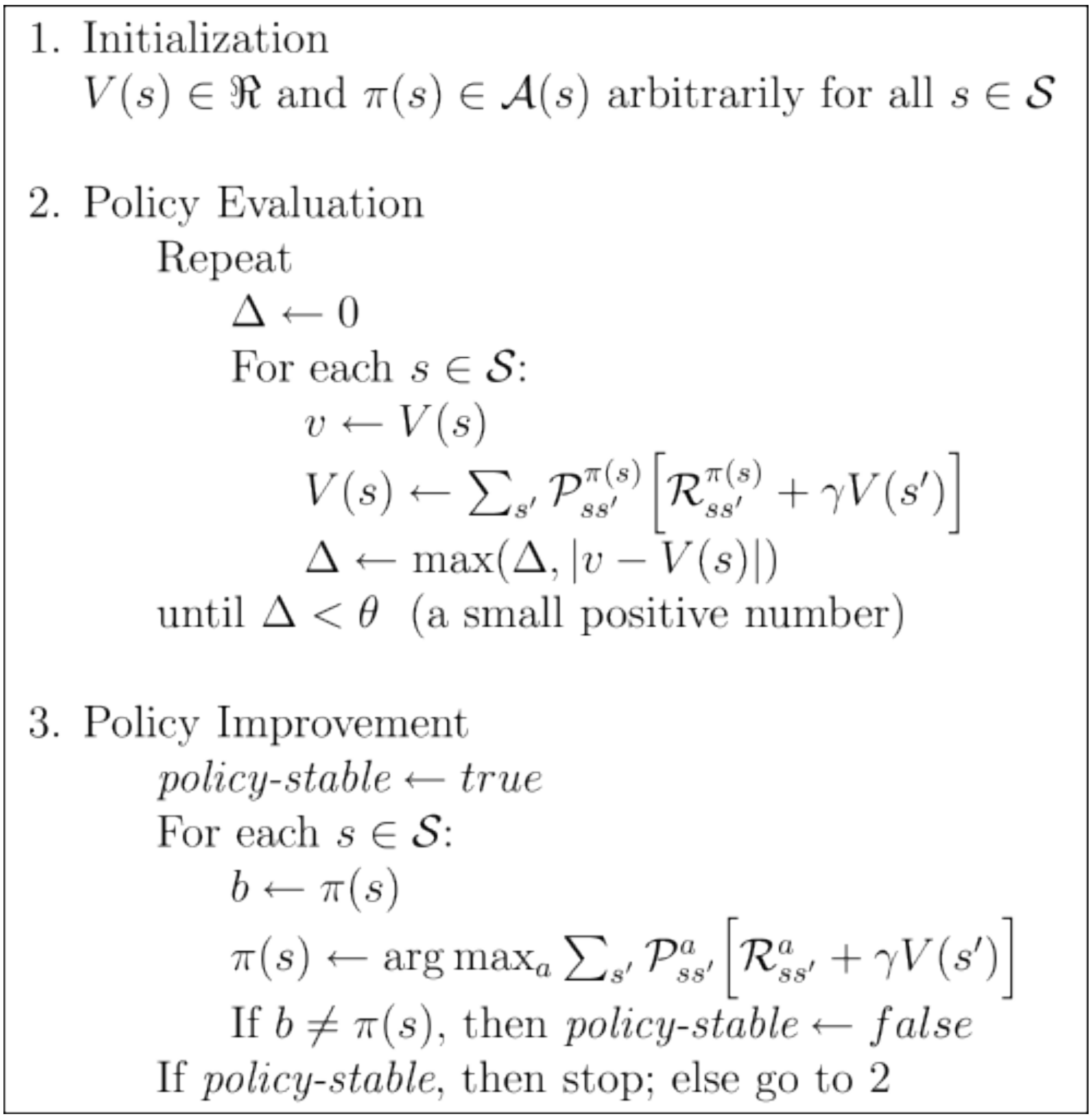
The specifications are:

- $\theta = 0.1$
- $\gamma = 0.9$
- $V_0(s)= 0$

For both algorithms, the implementation was made so that there is only one array for the V values. This way, it is faster than a two arrays solution however we must keep in mind that, normally, at each iteration, the new value of V is calculate directly with the last new values calculated. There is no such problem in this model because we don't use the value of V of a previous state to calculate the next one.

Policy Iteration

Algorithm



```
1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
       Repeat
           $\Delta \leftarrow 0$
           For each $s \in \mathcal{S}$:
               $v \leftarrow V(s)$
               $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[ \mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
               $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
       until $\Delta < \theta$  (a small positive number)

3. Policy Improvement
       $policy\text{-}stable \leftarrow true$
       For each $s \in \mathcal{S}$:
           $b \leftarrow \pi(s)$
           $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[ \mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
           If $b \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
       If $policy\text{-}stable$, then stop; else go to 2
```

Graphs

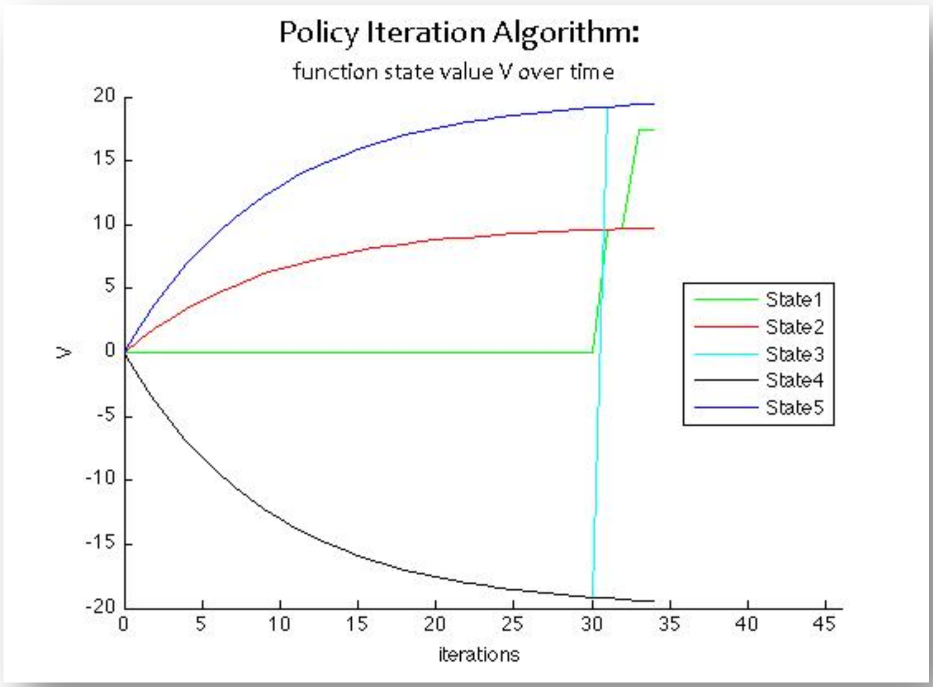


Figure 2: State Value V over time

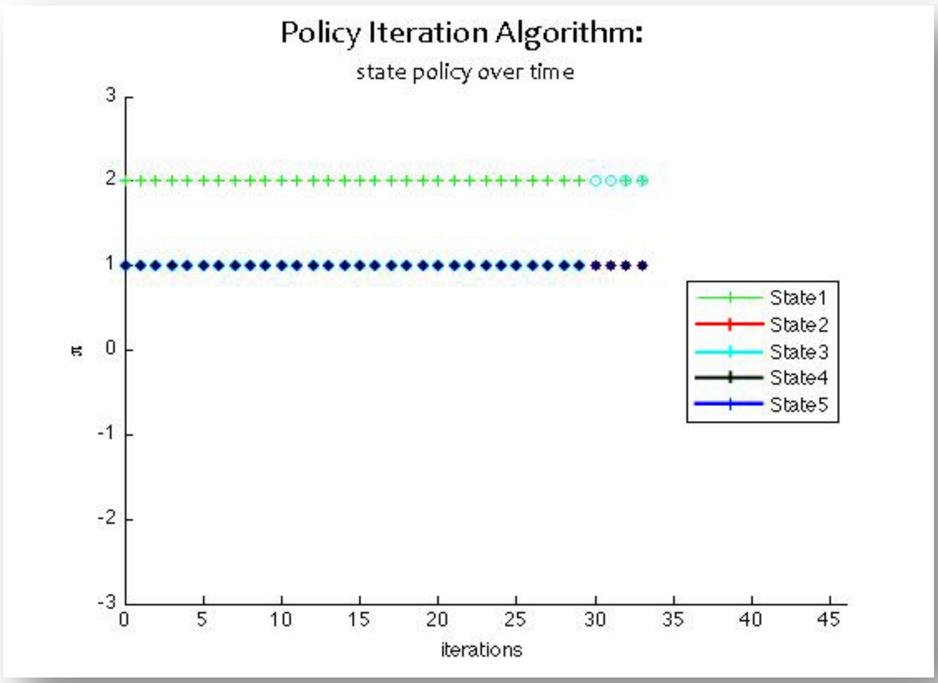
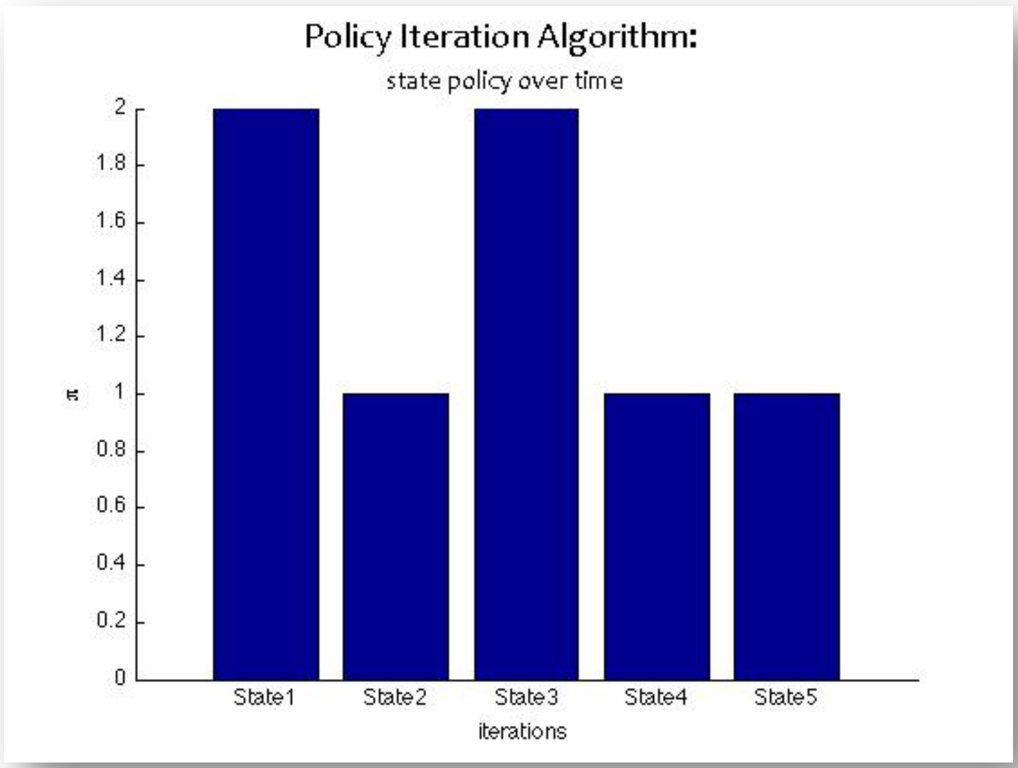


Figure 1: Pi over time

Figure 3: Last value of Pi



Analysis

As we can see on the first graph, the value of the function V for each state converges towards a fixed value.
That is when the value of the difference:

$$|V_i - V_{i+1}| < \theta$$

We observe that close to the 30th iteration the policy improve so the value of V changes radically. Indeed, this is the result of the equation:

$$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_\pi(s')]$$

The value of the policy π changes so only one part of the sum is not null. When the policy changes the other parts is taken into account and the previous one is now null (because there is only two possible actions for state 1 and 3).

Eventually, the we see on the second graph that the policy is stable for state 2,4 and 5 which have only one option and changes for the two other states. At the end the best policy is found that is: π = (2 1 2 1 1). This indicates that, from state 1 we must go to state 3 then to state 5 to gain the highest cumulated reward.

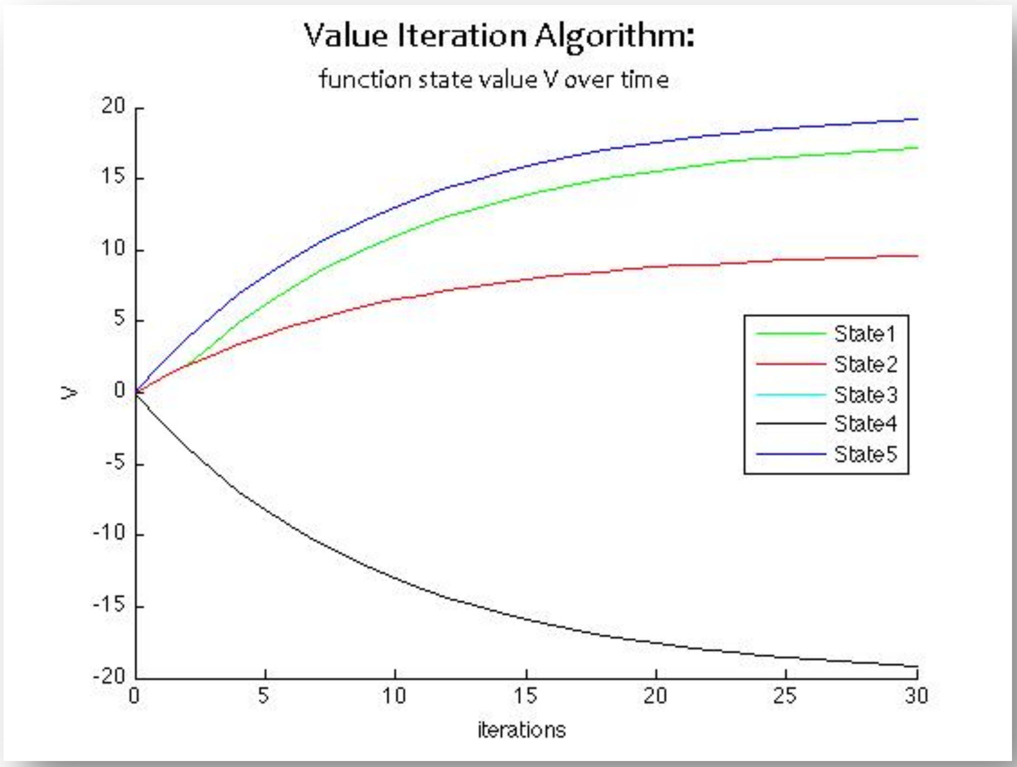
The last graph only shows the last value for the policy.

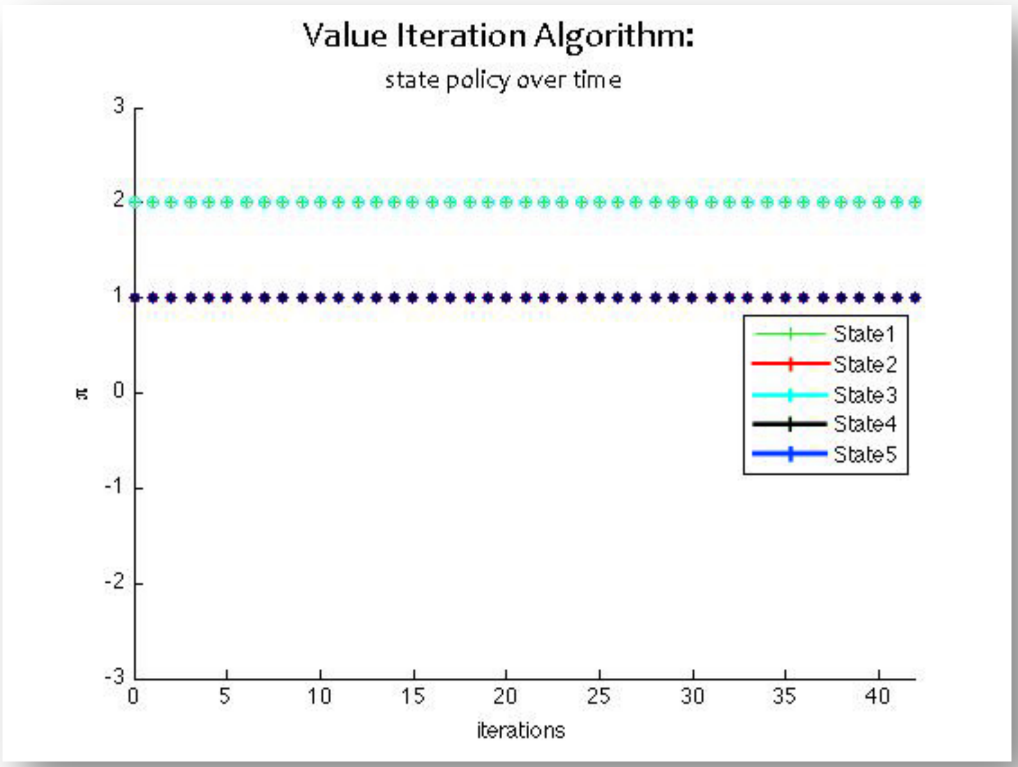
Value Iteration

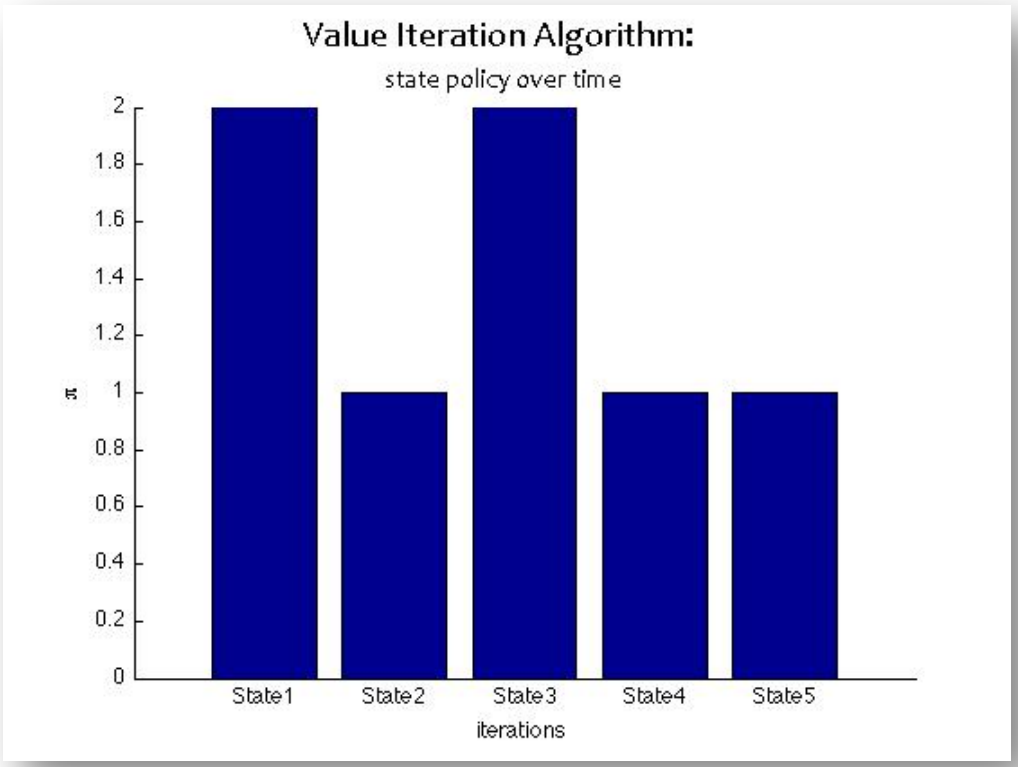
Algorithm

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

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
```

Graphs


Value Iteration Algorithm:
function state value V over time
V
iterations
State1
State2
State3
State4
State5
20
15
10
5
0
-5
-10
-15
-20
0
5
10
15
20
25
30

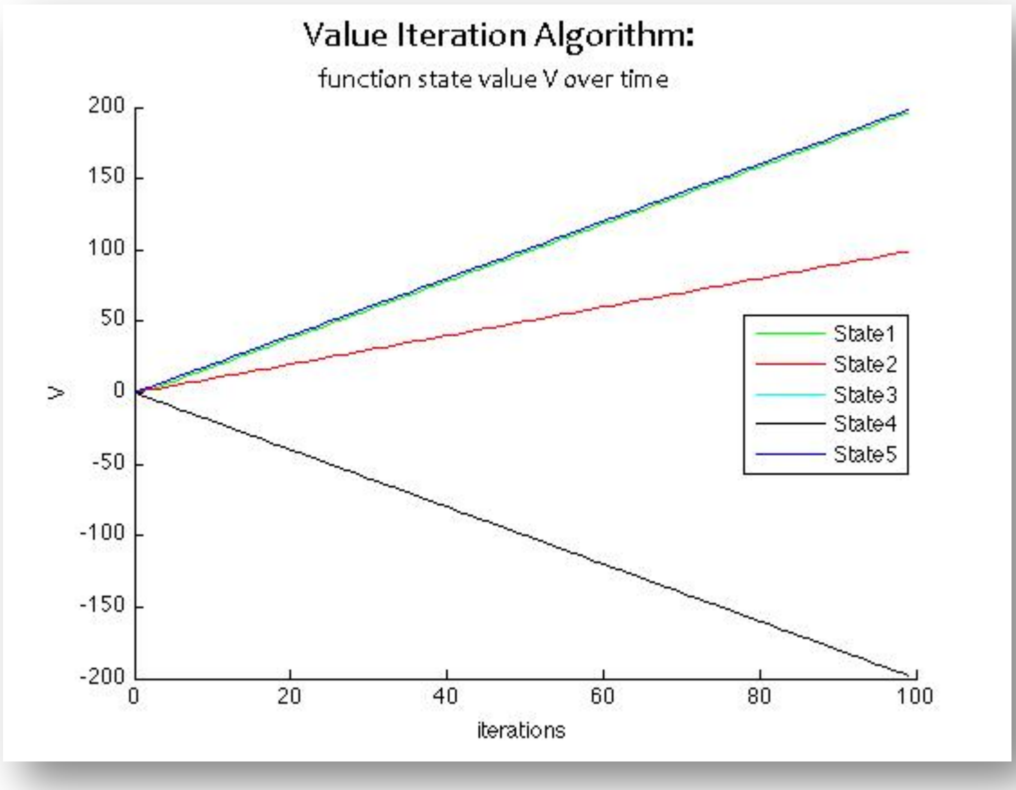

Value Iteration Algorithm:
state policy over time
π
iterations
State1
State2
State3
State4
State5
3
2
1
0
-1
-2
-3
0
5
10
15
20
25
30
35
40


Value Iteration Algorithm:
state policy over time
π
iterations
State1
State2
State3
State4
State5
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0

Analysis

The Value Iteration algorithm is very close to the other algorithm but it does only the Evaluation part of it. We can see on the first graph that it converges as well and even faster than the Policy Iteration algorithm. This is because we directly take a maximum between the values of V for the next possible states in the Evaluation part. We observe that the optimal policy is found directly so it does not need to be improved.

γ = 1



On this last figure, we show what happen when the value of γ is set to 1. The algorithm doesn't converge so when enter in an infinite loop. The difference that we talked about earlier is never lower than θ. If we remove the while loop we can observe what is on the last diagram. Straight lines that goes towards infinity. In fact, at each iteration of the loop we add to V a new value that is higher than 1 because the discounted factor doesn't play its role anymore.