REINFORCEMENT LEARNING

Seminar @ UPC TelecomBCN Barcelona (2nd edition). Spring 2020.

Instructors

Josep Vidal

Margarita Cabrera

Xavier Giró-i-Nieto

Organizers

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Supporters

Google Cloud

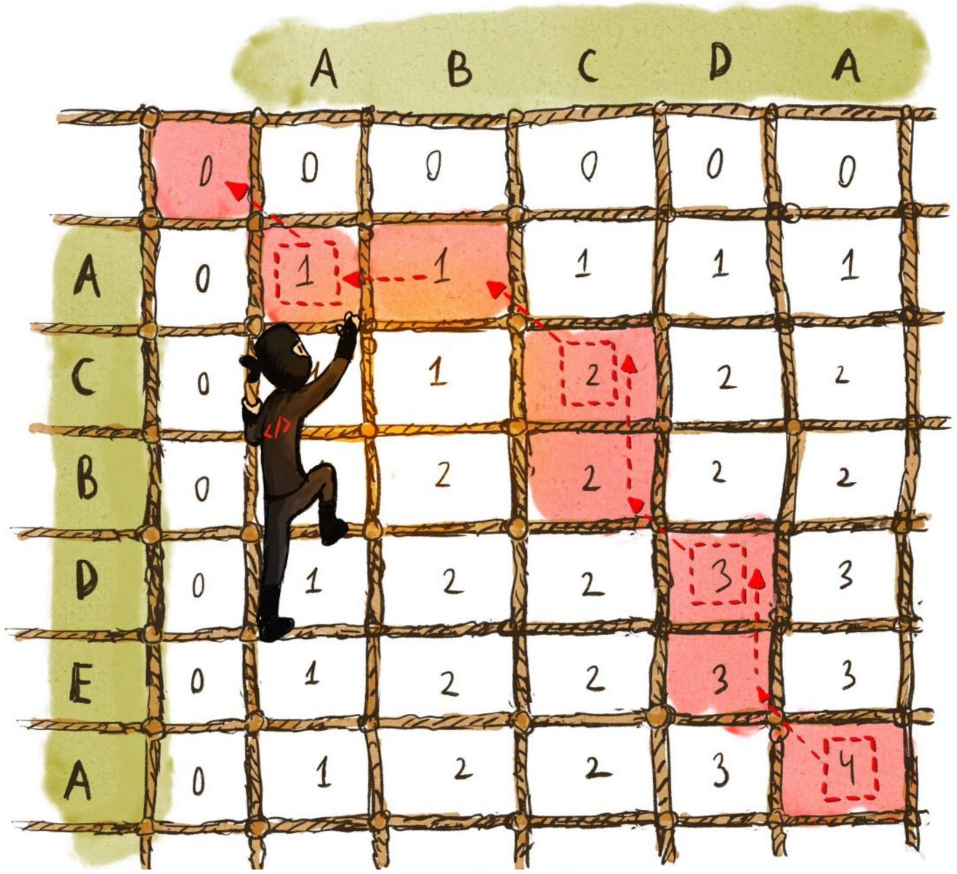
GitHub Education

+ info: http://bit.ly/upcrl-2020

telecos BCN

UPC

3. Dynamic programming



		A	B	C	D	A
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	[illegible]	1	2	2	2
B	0	[illegible]	2	2	2	2
D	0	1	2	2	3	3
E	0	1	2	2	3	3
A	0	1	2	2	3	4

Introduction

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).

- Classical DP algorithms are of limited utility in reinforcement learning both because they

  - assume a perfect model
  - require great computational expense

  but still they are still important theoretically.

- DP provides an essential foundation for the understanding of the methods presented in the rest of the seminar.

Introduction

Dynamic programming assumes full knowledge of the MDP. It is used for planning in an MDP:

- For **prediction**:

  input: an MDP and a policy, or an MRP

  output: the value function

- For **control**:

  input: an MDP

  output: the optimal value function and optimal policy

Introduction

Dynamic programming approaches for solving MDP imply the decomposition into two subproblems:

1. Policy evaluation: estimate a value function for a given policy
2. Policy improvement: get the best policy from the estimated value

They interact to build two different iterative algorithms: policy iteration and value iteration.

Policy evaluation

Apply iteratively the Bellman expectation equation:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right)$$

as

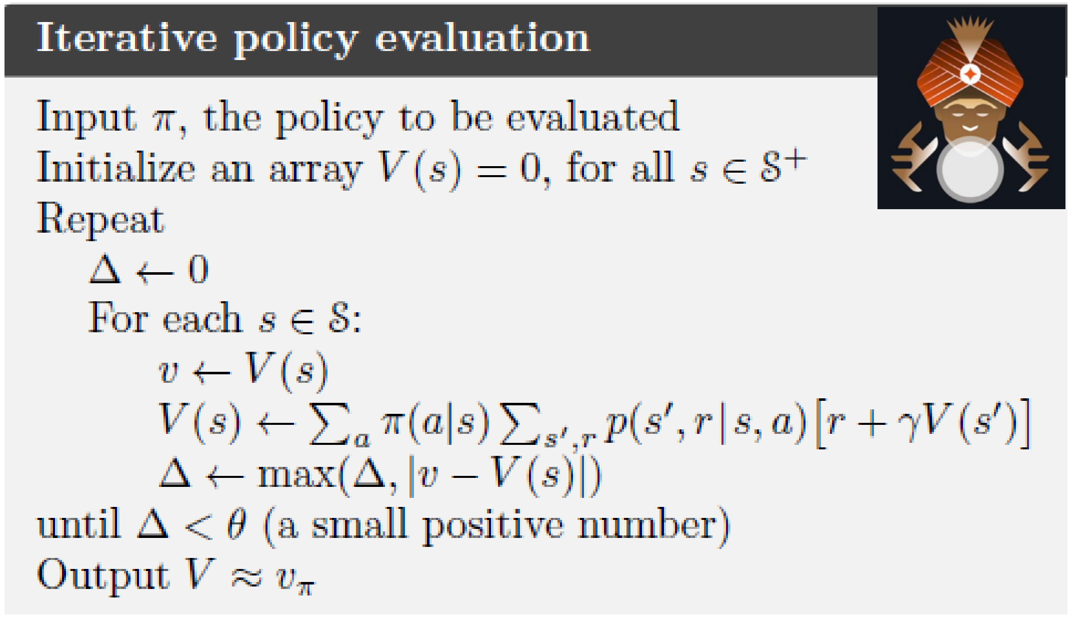
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_k(s')\right)$$

to get a matrix in-place solution: $\mathbf{v}_{k+1} = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}_k$ rather than inverting the matrix equation.

It can be shown that this iteration converges to $v_\pi(s)$ as $k \rightarrow \infty$.

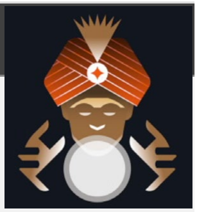
Policy evaluation

Iterative policy evaluation for a given policy using an in-place algorithm



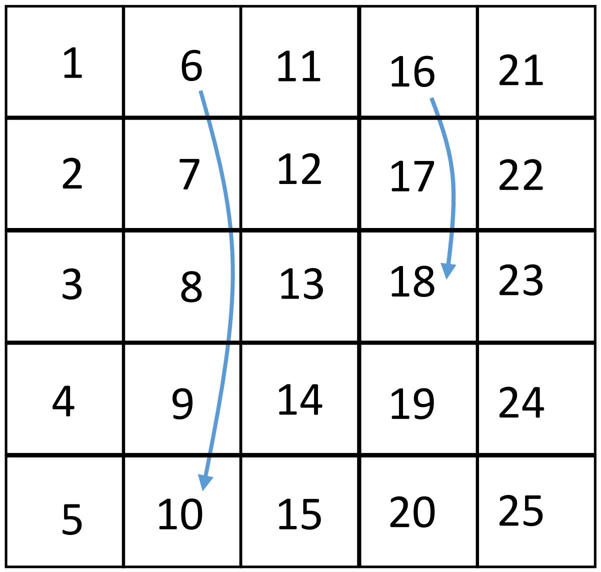
Iterative policy evaluation

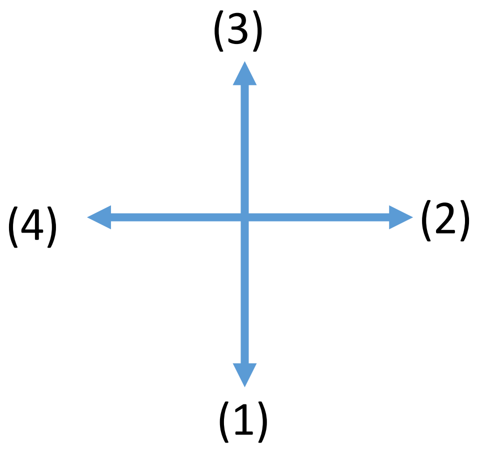
```
Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```



Example 2.2. 5×5 Gridworld example (IV)

d) Program the **Iterative Policy Evaluation** procedure to compute $v_\pi(s)$, for $s = 1,\dots,25$. Compare the result with the one obtained in question c), chapter 2, check they are the same.





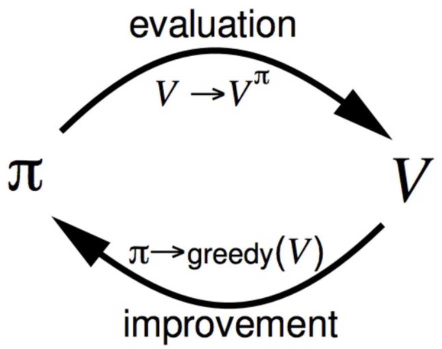
From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Policy improvement

Let us improve the policy by doing a two steps iteration:

- Given a policy π, evaluate it
- Improve the policy by acting greedily with respect to v_π

$$\pi' = greedy(\mathbf{v}_\pi) \geq \pi$$



$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Policy improvement

Let us act greedily on $q(s,a)$:

$$\pi' = \arg\max_{a \in \mathcal{A}} q_\pi\left(s,a\right)$$

This improves the value from any state s over one step:

$$q_\pi\left(s,\pi'(s)\right) = \max_{a \in \mathcal{A}} q_\pi\left(s,a\right) \geq q_\pi\left(s,\pi(s)\right) = v_\pi(s)$$

and therefore the value function is improved: $v_{\pi'}(s) \geq v_\pi(s)$

Two approaches for DP…

Policy iteration:

- Evaluate policy until convergence and then improve policy
- Operations per cycle $O(K\,|\mathcal{S}|^2 + |\mathcal{A}|\,|\mathcal{S}|^2)$, where K is the number of iterations in the inner policy evaluation loop
- Requires few cycles

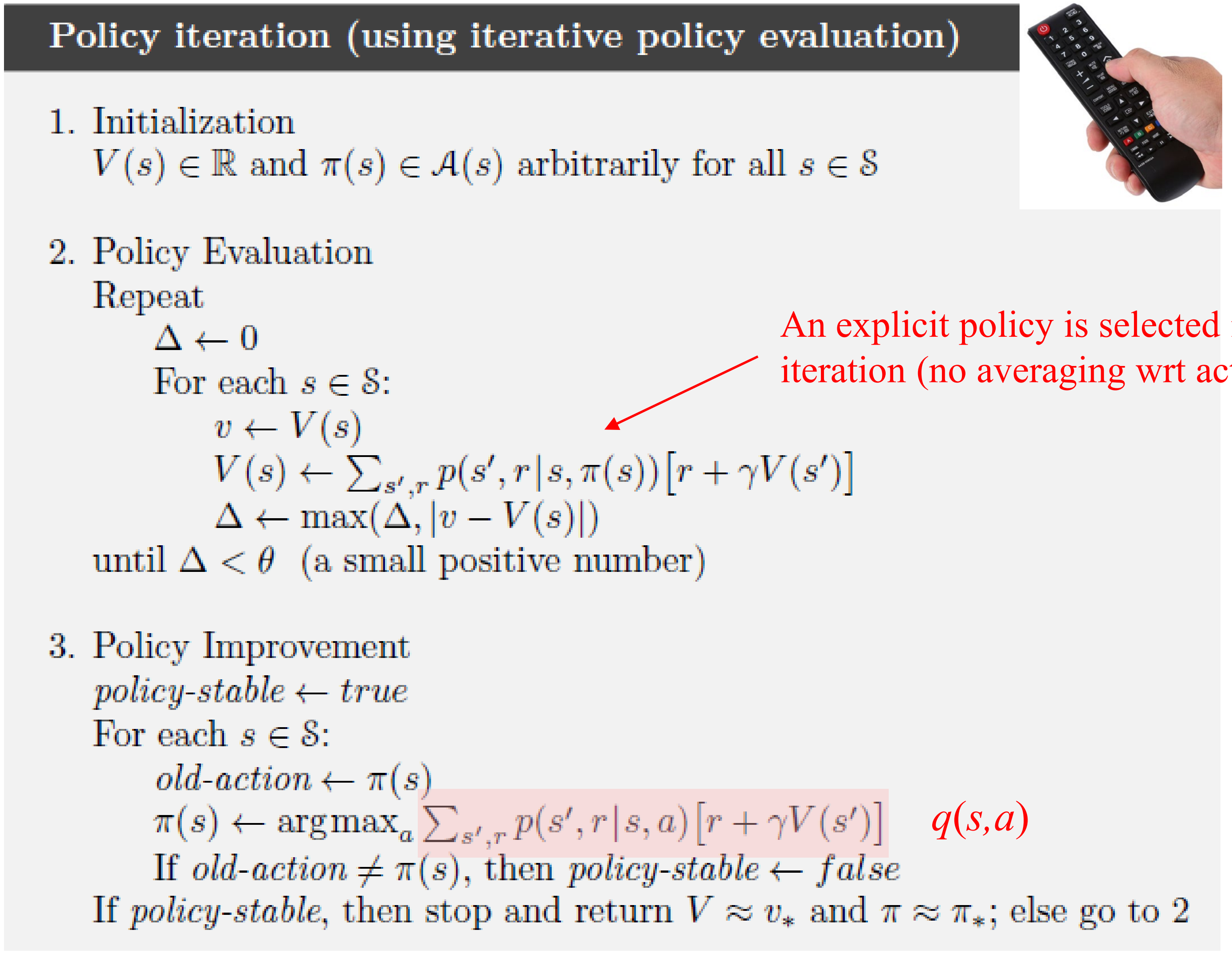
Value iteration:

- Evaluate policy only with single iteration and then improve policy
- Operations per cycle $O(|\mathcal{A}|\,|\mathcal{S}|^2)$
- Requires many cycles

Both use the policy evaluation and the policy improvement procedures seen so far, in a slightly different way. The procedures are…

Policy iteration

Evaluate policy until convergence and then improve policy:



Policy iteration (using iterative policy evaluation)

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

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

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       *old-action* $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \mathrm{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

An explicit policy is selected in outer iteration (no averaging wrt actions *a*)

$q(s,a)$

Policy iteration

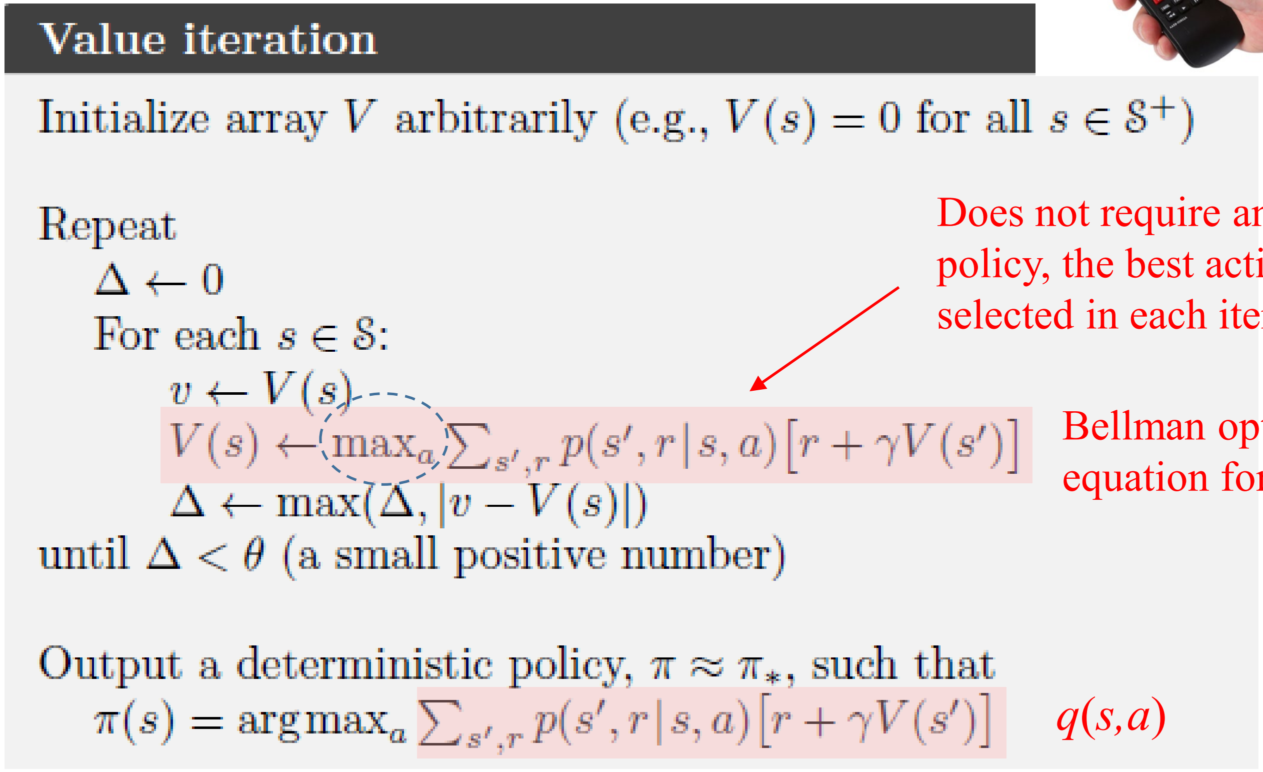
- If improvement stops, the Bellman optimality equation is satisfied.
- The final policy is an optimal policy:

$$v_{\pi}(s) = \max_{a \in \mathcal{A}} q_{\pi}(s,a) = v_{*}(s) \qquad \forall \, s$$

- Some questions for a faster convergence:
  - Does policy evaluation need to converge?
  - Why not updating policy every iteration? This would be equivalent to **value iteration.**
  - Or stop after *k* iterations of iterative policy evaluation?

Value iteration

Evaluate value function and then obtain policy:



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, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

Does not require an explicit policy, the best action is selected in each iteration

Bellman optimality equation for $v(s)$

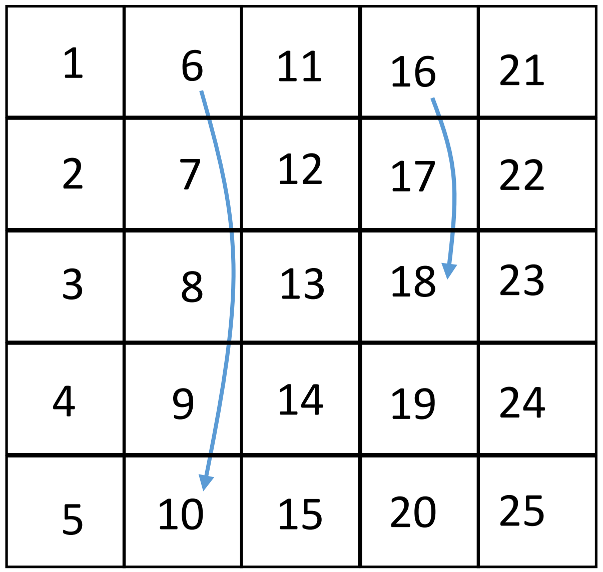
$q(s,a)$

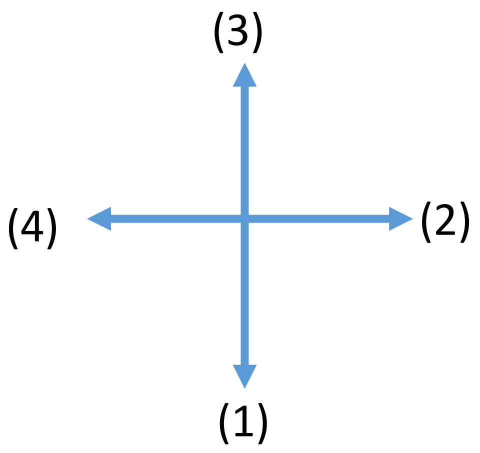
Value iteration

- Policy evaluation is stopped after just one sweep (one update of each state).
- Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.
- It is a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps.
- Iterative application of Bellman optimality backup.

Example 2.2. 5×5 Gridworld example (VI)

e) Program the **Value Iteration** and the **Policy Iteration** procedures to compute $\pi^*(s)$, for $s = 1,\dots,25$. Draw in square figures all the intermediate and the last obtained policies.





From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Example 3.1: Jack's car rental (I)

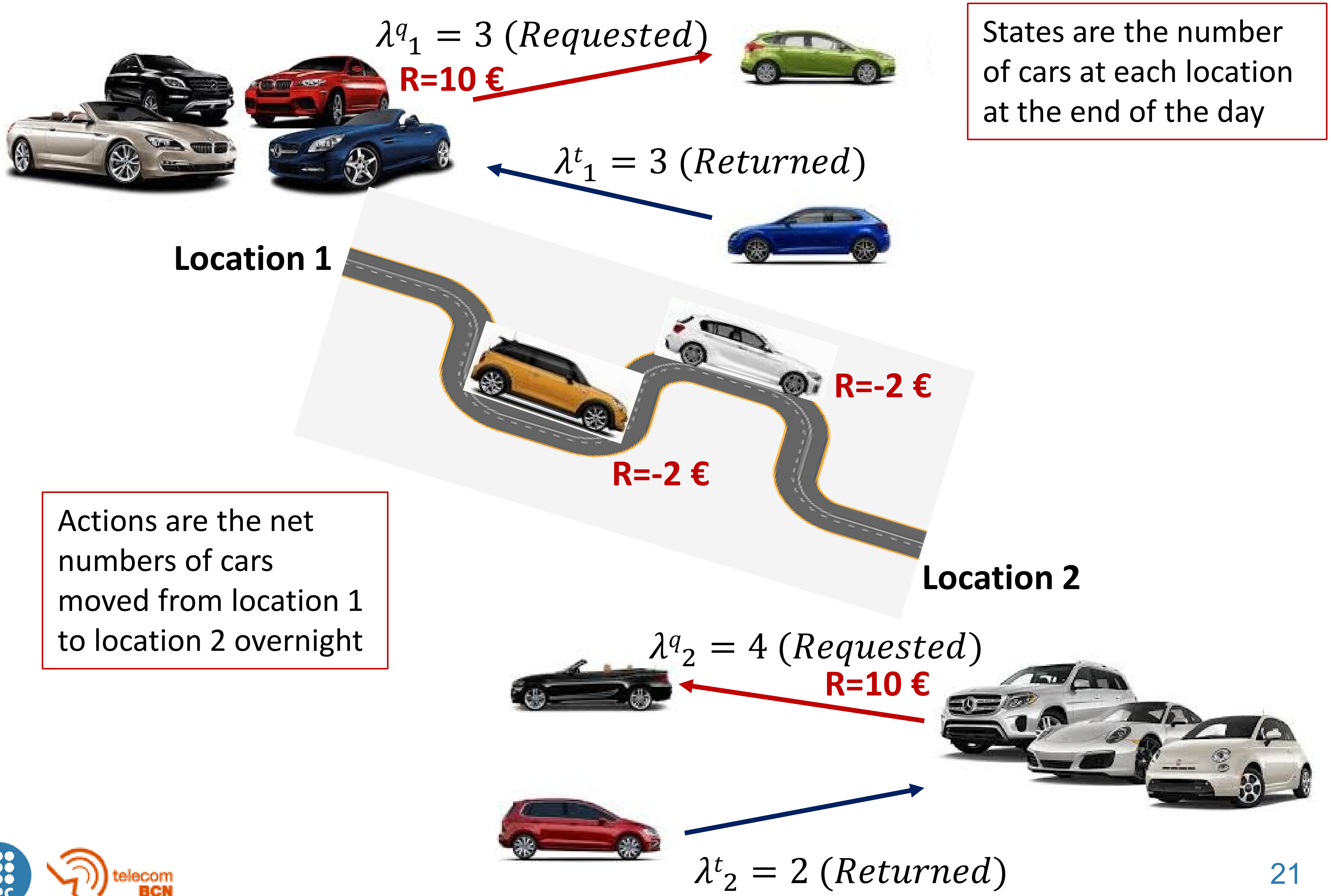
- Two locations of a car rental company.
- Jack is credited **10 €** by each rented car.
- Jack can move them between the two locations overnight, at a cost of **2 €** per car moved.
- The number of cars requested and returned at each location are Poisson random variables:

$$\Pr(n) = \frac{\lambda^n e^{-\lambda}}{n!}$$

- $\lambda^{q}_{1} = 3$ and $\lambda^{q}_{2} = 4$ for daily rental requests at the first and second location.
- $\lambda^{t}_{1} = 3$ and $\lambda^{t}_{2} = 2$ for daily rental returns.
- There can be no more than $N = 20$ cars at each location.
- A maximum of $A_{\max} = 5$ cars can be moved from one location to the other in one night.
- Cars become available for renting the day after they are returned.

From Sutton & Barto, Reinforcement Learning: An Introduction, 1998

Example 3.1: Jack’s car rental (II)



$\lambda^{q}{}_{1} = 3\ (Requested)$
R=10 €
$\lambda^{t}{}_{1} = 3\ (Returned)$
Location 1
R=-2 €
R=-2 €
Location 2
$\lambda^{q}{}_{2} = 4\ (Requested)$
R=10 €
$\lambda^{t}{}_{2} = 2\ (Returned)$

States are the number of cars at each location at the end of the day

Actions are the net numbers of cars moved from location 1 to location 2 overnight

Example 3.1: Jack's car rental (III)

- This is a continuing finite MDP
- The time steps are days
- The state is the number of cars at each location at the end of the day.
- The number of states is $N_s = (N+1)(N+1)$ $\mathcal{S} = \{s_i \,;\, i = 1,\ldots, N_s\}$
- The actions are the number of cars moved from location 1 to location 2 overnight $\mathcal{A} = \{-A_{max},\ldots, +A_{max}\}$
- At the end of the day Jack's car rental company is in state $s = (n_1\,,\,n_2)$ and moves a cars from location 1 to location 2.
- Afterwards, depending on the number of requested (returned) cars at location 1(2) next day the state is $s' = (n_1', n_2')$
- Depending on s not all the actions in $\mathcal{A}$ are possible:

$$\max([-A_{max}\,, -n_2, n_1 - N\,]) \leq a \leq \min([+A_{max}\,, n_1\,, N - n_2\,])$$

Example 3.1: Jack's car rental (IV)

An illustrative case…

- Let's suppose that at the end of the day the state is $s = (n_1, n_2) = (10,18)$
- Jack decides to move two cars at night from loc. 2 to loc. 1, so $a = -2$
- The probability of state $s' = (n'_1, n'_2) = (10,20)$ at the end of the next day can be factorized as:

$$p(s'|a,s) = p_1(n_1'|a,n_1)\,p_2(n_2'|a,n_2) = p_1(10|-2,10)\,p_2(20|-2,18)$$

i.e. given the action a and the state (n_1, n_2), the state is random: the final number of cars at loc. 1 (n'_1) is independent of the initial number of cars at loc. 2 (n'_2), and vice versa.

Location 1:
Starts with n_1 - a cars
At closing time: n'_1

Independent events given a

Location 2:
Starts with n_2 + a cars
At closing time: n'_2

Example 3.1: Jack's car rental (V)

How to obtain **transition probabilities?**

- Number of rented (returned) cars at **location 1**: q_1, (t_1)

$$n_1{}' = n_1 - a - q_1 + t_1$$

- Transition probability: $p_1(n'_1 \,|a, n_1) = p_1(10|{-2},10)$ can be computed as the sum of probabilities of some combinations regarding rented (q_1) & returned (t_1) cars

If $q_1 < 2$ final state cannot be n'_1=10

q_1	2	3	4	5	6	7	8	9	10	11	12
t_1	0	1	2	3	4	5	6	7	8	9	10

Maximum number of cars that can be rented

- So,

$$\boxed{p_1(n_1{}'\,|\,a,n_1) = \sum_{q_1=\max(0,n_1-n_1{}'-a)}^{n_1-a} \Pr(q_1)\Pr(t_1 = n_1{}' - n_1 + a + q_1) = \sum_{q_1=2}^{12} \Pr(q_1)\Pr(t_1 = q_1 - 2)}$$

where $\Pr(q_1) = \dfrac{\left(\lambda_1^q\right)^{q_1} e^{-\lambda_1^q}}{q_1\,!}$ except $\Pr(q_1 = n_1 - a) = \displaystyle\sum_{q_1=n_1-a}^{\infty} \dfrac{\left(\lambda_1^q\right)^{q_1} e^{-\lambda_1^q}}{q_1\,!}$

and $\Pr(t_1) = \dfrac{\left(\lambda_1^t\right)^{t_1} e^{-\lambda_1^t}}{t_1\,!}$ (note that if $n'_1 = N$ then $\Pr(t_1)$ is computed as cdf)

Example 3.1: Jack's car rental (VI)

How to obtain **transition probabilities**?

$$p(s'|a,s) = p_1(n_1'|a,n_1)\,p_2(n_2'|a,n_2)$$

$$= \sum_{q_1=q_{1\min}^{s,s'}}^{n_1-a} \Pr(q_1)\Pr(t_1 = n_1'-n_1+a+q_1) \sum_{q_2=q_{2\min}^{s,s'}}^{n_2+a} \Pr(q_2)\Pr(t_2 = n_2'-n_2-a+q_2)$$

$$= \sum_{q_1=q_{1\min}^{s,s'}}^{n_1-a} \sum_{q_2=q_{2\min}^{s,s'}}^{n_2+a} \Pr(q_1,t_1,q_2,t_2)$$

where

$$q_{1\min}^{s,s'} = \max(0, n_1 - a - n_1') \qquad q_{2\min}^{s,s'} = \max(0, n_2 + a - n_2')$$

and

$$\Pr(q_1,t_1,q_2,t_2) = \Pr(q_1)\Pr(t_1 = n_1'-n_1+a+q_1)\Pr(q_2)\Pr(t_2 = n_2'-n_2-a+q_2)$$

Join probability of four independent events

How to obtain **expected rewards**?

$$\boxed{r(s,a,s') = \sum_n \overbrace{\Pr(r_n|s,a,s')}^{\frac{\Pr(r_n,s'|s,a)}{p(s'|a,s)}} r_n = \frac{\sum_{q_1=q_{1\min}^{s,s'}}^{n_1-a} \sum_{q_2=q_{2\min}^{s,s'}}^{n_2+a} \Pr(q_1,t_1,q_2,t_2)(10(q_1+q_2)-2|a|)}{p(s'|a,s)}}$$

Example 3.1: Jack's car rental (VII)

Solving Bellman equation

- Deterministic policy: a (s) $s = 1 \ldots N_s$
- Discount factor γ
- Compute transition probability matrix

$$\boxed{[\mathbf{P}]_{ss'} = p(s'|s) = \sum_{a=-A_{MAX}}^{+A_{MAX}} p(a|s)\,p(s'|a,s) = p(s'|a(s),s)}$$

- Compute expected rewards

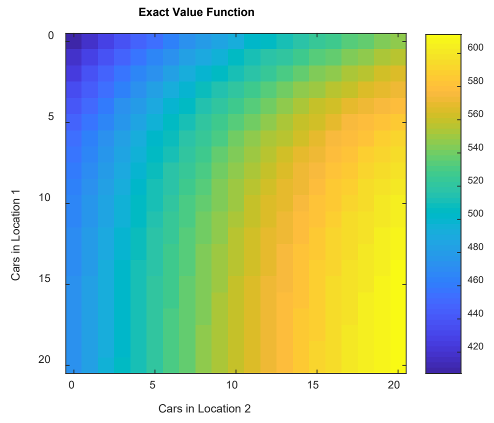
$$\boxed{[\mathbf{R}]_s = r(s) = \sum_{a=-A_{MAX}}^{+A_{MAX}} p(a|s) \sum_{s'=1}^{N_s} p(s'|s,a)\,r(s,a,s') = \sum_{s'=1}^{N_s} p(s'|s,a(s))\,r(s,a(s),s')}$$

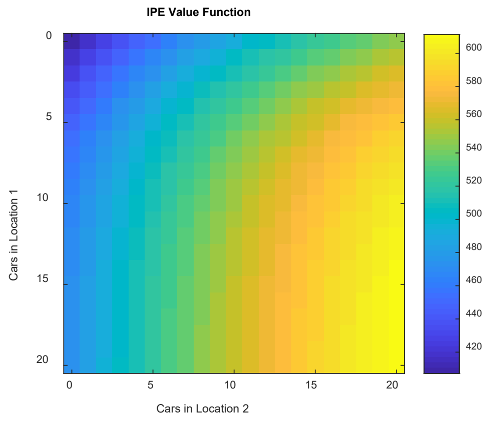
- Solve the Bellman equation

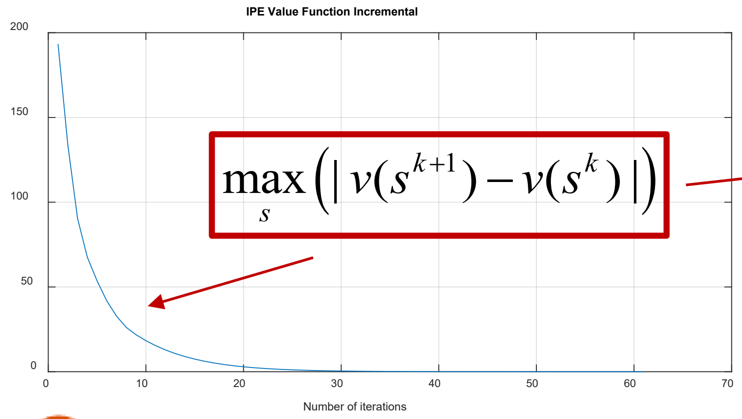
$$\mathbf{v} = \left(\mathbf{I} - \gamma \mathbf{P}\right)^{-1} \mathbf{R}$$

Example 3.1: Jack's car rental (VIII)

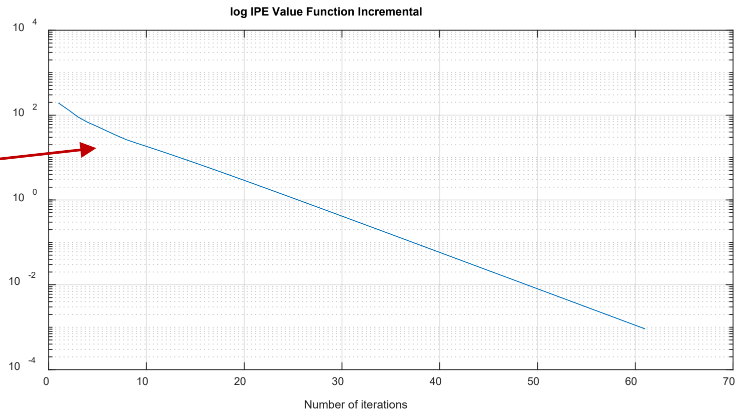
- Solving Bellman equation vs applying IPE (Iterative Policy Evaluation)
- Deterministic Policy: $a(s) = 0$, $s = 1,\ldots, N_s$. No cars are moved at night
- $\gamma = 0.9$, $\theta = 0.001$





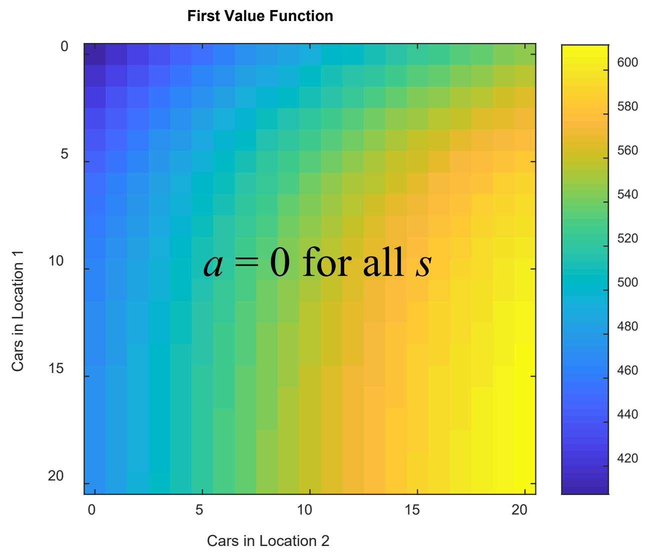


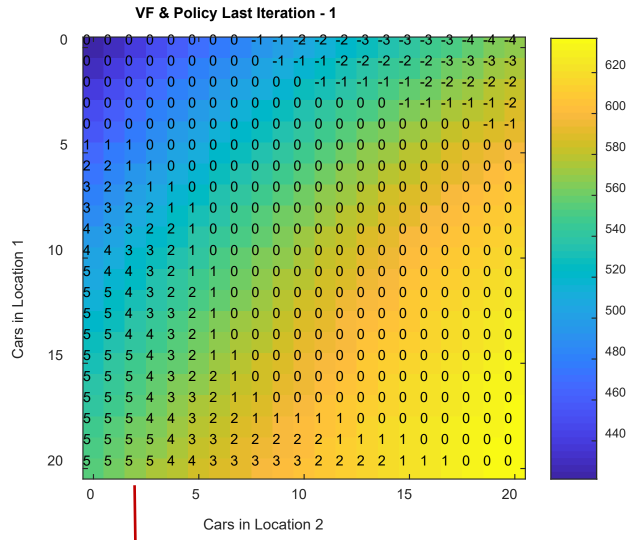
$$\max_{s}\left(\left| v(s^{k+1}) - v(s^{k}) \right|\right)$$

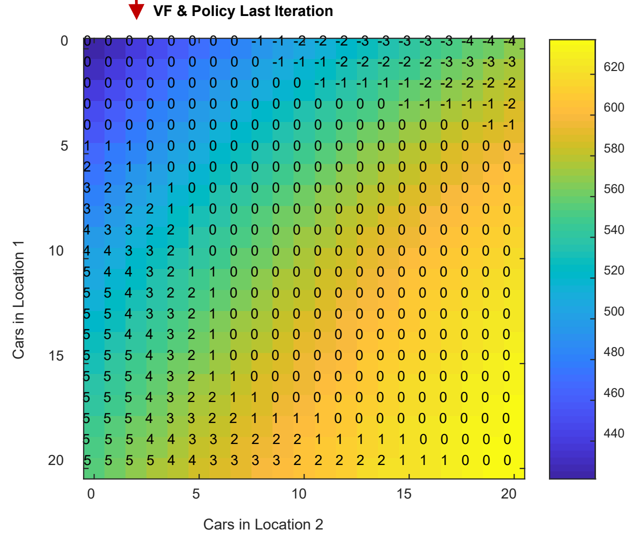


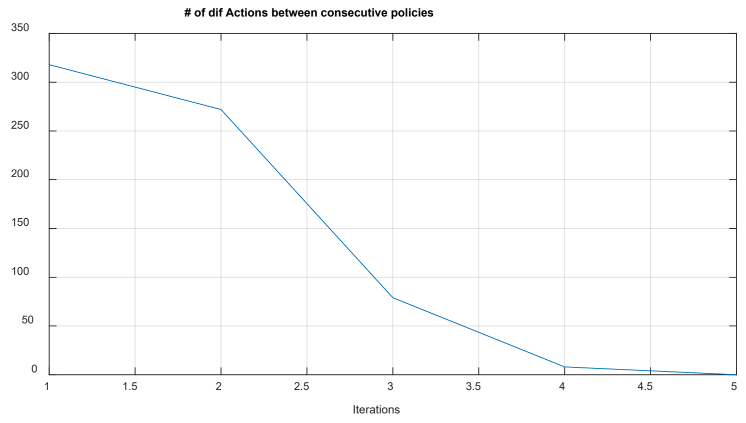
Example 3.1: Jack's car rental (IX)

- Policy iteration $\gamma = 0.9$, $\theta = 0.001$



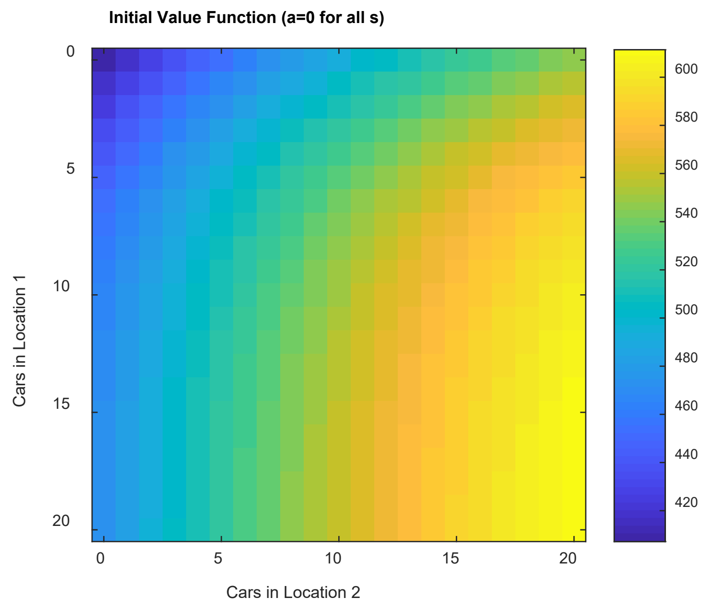


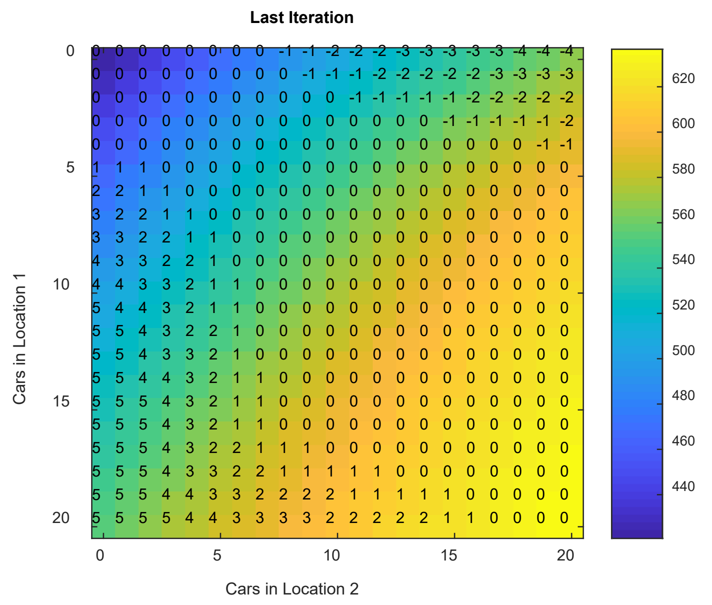




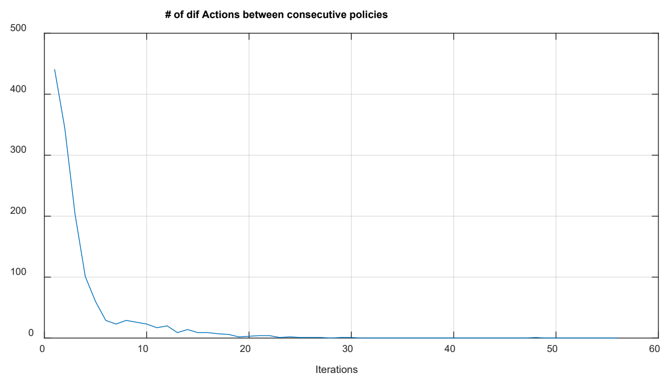
Example 3.1: Jack’s car rental (X)

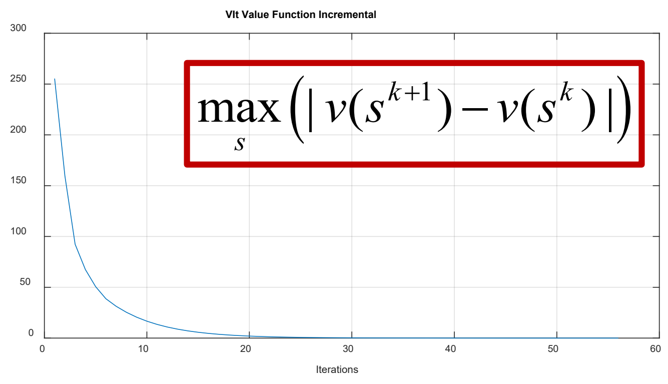
- Value iteration $\gamma = 0.9$, $\theta = 0.001$





0	0	0	0	0	0	0	0	-1	-1	-2	-2	-2	-3	-3	-3	-3	-3	-4	-4	-4
0	0	0	0	0	0	0	0	0	-1	-1	-1	-2	-2	-2	-2	-2	-3	-3	-3	-3
0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-2	-2	-2	-2	-2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	3	2	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	3	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	3	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	4	4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	4	3	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	4	3	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	4	4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	4	3	2	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0
5	5	5	4	3	3	2	2	1	1	1	1	1	0	0	0	0	0	0	0	0
5	5	5	4	4	3	3	2	2	2	2	1	1	1	1	1	0	0	0	0	0
5	5	5	5	4	4	3	3	3	3	2	2	2	2	2	1	1	0	0	0	0





$$\boxed{\max_{s}\left(\left| v(s^{k+1}) - v(s^{k}) \right|\right)}$$

Number of float operations when searching for the optimum policy

$|\mathcal{S}|=n \qquad |\mathcal{A}|=m$

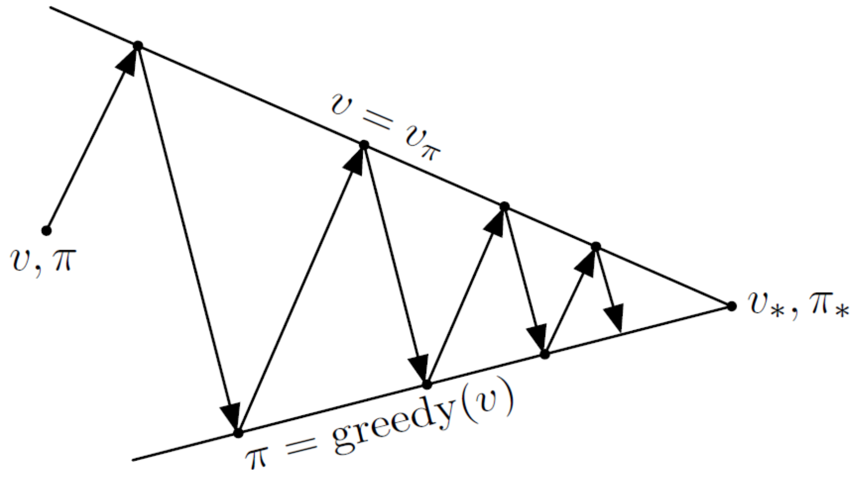
	Number of Cases	Matrix Inversion	Solving System of Equations	Total
Brute Force	$N_c = m^n$	$N_i = n^3$	$N_1 = N_i + 2\,n^2$	$N_T = N_c N_1$
	1.1259e+15	15625	16875	1.9000e+19

	Number of Iterations	Floats/Inner Iteration K loops	Total
Policy Iteration	N	$N_1 = Kn^2 + m\,n^2$	$N_T = N\,N_1$

	Number of Iterations	Floats/Iteration	Total
Value Iteration	N	$N_1 = m\,n^2 = 2500$	$N_T = N\,N_1$

General policy iteration (GPI)

- Key idea: let policy evaluation and policy improvement processes interact, independently of the granularity and other details of the two processes.
- If both the evaluation process and the improvement process stabilize, that is, no longer produce changes, then the value function and policy must be optimal.



Quiz #2