Master M2 MVA 2015/2016 - Reinforcement Learning - TP 1

By: Lilian Besson (`lilian.besson at ens-cachan.fr`). Attached programs: The programs for this TP are included in the `zip` archive I sent, and are regular **MATLAB/Octave** programs (tested with Octave only).

Problem 1 : Retail store management

Question 1

I chose to keep the default parameters, as given in the template `mainTP1.m` file: $M = 15$ (size of the stock), $K = 0.8$ (delivery cost), $h = 0.3$ (maintenance cost), $c = 0.5$ (buying price), $p = 1$ (selling price), $\gamma = 0.98$ (discount factor ie. inflation rate). The distribution of customers is assumed to be a truncated geometric distribution (of parameter $q = 0.1$).

I also tried to change them a little bit to check that the behavior is coherent (for instance, a bigger stock does not change much the situation, increasing the costs reduce the optimal cumulated profit while increasing selling price does, and a higher inflation can suppress all possibility of profit for the store manager – all this is logical).

Both the **VI** and **PI** methods are iterative, with a number of steps T, that I chose[1] to be 300 for **VI**, and 10 for **PI**. Increasing this T increases the time complexity, and at one point does not improve accuracy of the answer (as shown in Question 2). Both give the same optimal policy and value vector (as expected[2]):

- **VI** gave $\pi^\star_{VI}$ = `[ 10, 9, 8, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]`.
- **PI** gave $\pi^\star_{PI}$ = `[ 10, 9, 8, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]`.

They both have the same shape (as illustrated below): the manager buys $M - x$ objects when his stock is almost empty ($x < \theta \stackrel{\text{def}}{=} 4$), and does not buy any object if the stock is "full enough":

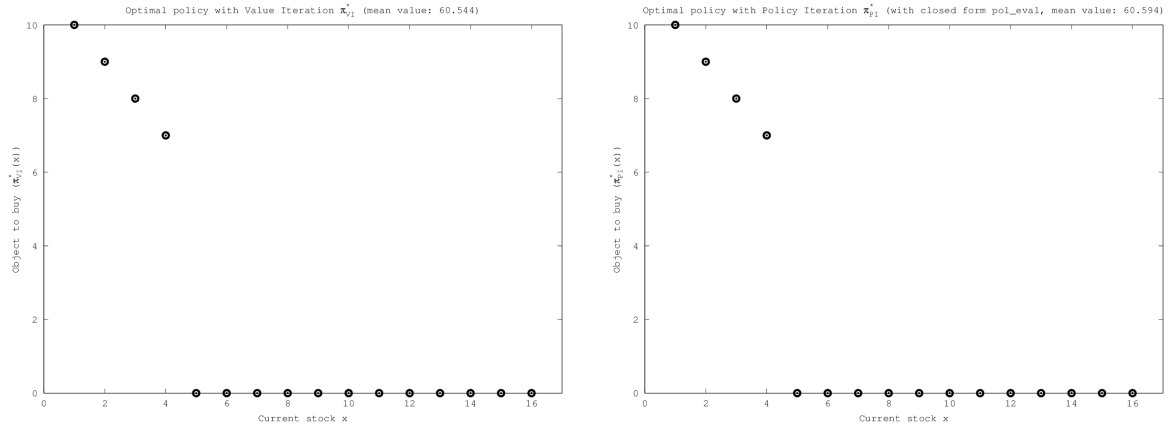

Figure 1: Optimal policies $\pi^\star_{VI}$ and $\pi^\star_{PI}$

[1] It seems logical: in class we saw that each step of **VI** is quicker but it requires more steps (and it works well).

[2] I kept a struggling with a *vicious* bug for hours: `[V pol] = max(Q, [], 2)` was returning `pol` as a vector of *indeces* (in $[1, M+1]$) while our implementation considered states and actions to be in $[0, M]$! **VI** and **PI** were giving different results! Just adding `pol -= 1` after each evaluation of `pol` fixed that.

This strategy is quite logical, under the assumption we made on (D_t), it prevents the manager from purchasing too much and from having an empty stock. The fact that there is a fixed cost of delivery explains also why the manager should buy a lot of objects when needed and nothing otherwise (it was not easy to tweak this aspect of the model, I did not try to).

The value function (vector $V^\star$) has that form: `[ 57.6, 57.8, 58.0, 58.2, 58.5, 58.9, 59.3, 59.6, 59.9, 60.1, 60.4, 60.6, 60.7, 60.9, 61.0, 61. ]`. It's almost constant equals to 60, and increasing when the initial stock increases[3], which is logical (if the initial stock is full, the store wins more money).

I checked that changing the inflation rate (γ, e.g. from `0.98` to `0.92`) does not change the optimal policy but does change the optimal value: decreasing γ decreases the reward of our clever manager, this is also very logical. If γ is small enough (like `0.70`), **PI** converges in **one** step! $\pi^\star$ has the same shape, the threshold θ is reduced, and the optimal reward is really reduced. I did not try to play with the other parameters of our MDP, but I'm sure we would observe the same kind of "logical" behavior.

Question 2

For **PI**, I implemented the three different version of `pol_eval` (Monte-Carlo simulation, closed form by inverting the Bellman equation, or fixed point method by iterating the Bellman operator $\mathcal{T}$). In practice, I found that the fixed point method is very close to the closed form (ie the exact solution) from about 200 steps of iteration of $\mathcal{T}$ on V_0, but is slower (M is not big enough for $200M^2 < M^3 \dots$), so **I used the closed form option here**.

As requested, here is a plot of the error rates for both **VI** and **PI**, done a posteriori by using the final V_K as $V^\star$ (cf. the functions `VI_with_plots.m` and `PI_with_plots.m`).

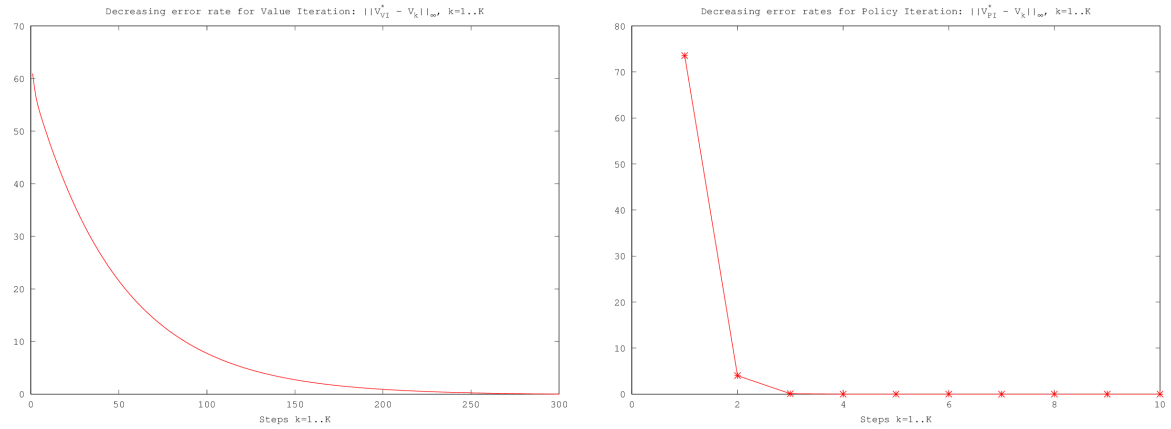

Figure 2: Decreasing error rates $\|V^\star - V_k\|_\infty, k = 1 \dots T$, for **VI** (left) and **PI** (right).

For a fixed number of iteration, **PI** is clearly the more efficient approach (the scale of the x axis is different for the right plot, T is only 10 steps for **PI** while it is 300 for **VI**). But for big MDPs, each step of **PI** is way costlier than a step for **VI**. Here, $M = 15$ so the number of states and actions is (really) too small to be able to compare (asymptotic) complexity.

[3]Remember that $V^\star : x_0 \mapsto V^{\pi^\star}(x_0)$ quantifies the long-term γ-discounted cumulative reward if we start from x_0 and follow $\pi^\star$.

Problem 2 : Q-Learning as a first RL algorithm

Question 3

The **Q-Learning** algorithm has been implemented in the file `Qlearning.m`.

- For a real world reinforcement learning, the distribution D_t would really be unknown, and just observed, but here we need it to be able to perform the step "*observe a transition: get a reward* R_t *and a new state* X_{t+1}".
- For the "suitably exploratory policy", I tested with a cycling one (selecting `a_t = mod(t,M+1)`) and a random one (`a_t = randi(M+1)-1`), and the random one seems to be quicker.
- For the initial state, I tested $x_0 = 0$ or $x_0 = M$ or a random one, and it does not change the results at all. That's logical because the next oen x_1 is already random.

And here are the policies found, for different numbers of iterations (`nbit` in the code):

Iterations	Time	Approximated optimal policy	
10^2	$0.9s$	`[ 12, 6, 0, 15, 0, 13, 3, 8, 8, 5, 4, 6, 13, 1, 11, 3 ]`	(*bad!*)
10^3	$1.52s$	`[ 10, 8, 15, 15, 14, 2, 10, 9, 1, 5, 4, 11, 1, 2, 1, 2 ]`	(*bad*)
10^4	$7.27s$	`[ 13, 12, 14, 12, 11, 9, 6, 4, 0, 0, 0, 1, 0, 9, 13, 5 ]`	(*meh?!*)
10^5	$66s$	`[ 10, 9, 7, 9, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ]`	(*better*)
10^6	$11m51s$	`[ 10, 7, 9, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]`	(*almost!*)
For $\pi^\star$	**VI** or **PI**	`[ 10, 9, 8, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]`	(*optimal*)

As we can see the Q-Learning algorithm needs a lot of iterations to find the optimal policy $\pi^\star$ (10^7 might not be enough) but it is able to tell us quite quickly that the manager should only order big commands or order nothing, and this is mainly because of the fixed cost of delivery.

The action was chosen by cycling in $\mathcal{A} = \{0, \dots, M\}$, another option is to choose a_t *randomly*, but we would very likely get better results if we were choosing the action a in a way to explore where we have bigger incertitude.

Conclusion

Except Octave specific coding mistakes (like indeces starting in 0 when actions/states were starting at 0), and the "observe a transition" part of the Q-Learning algorithm, that TP was not too hard, but I found it interesting and fun to do! I'm looking forward for the next one.