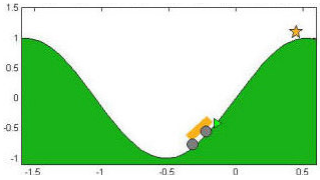
TP 4: Mountain Car

emilie.kaufmann@inria.fr

December 15th, 2015

1 Position of the problem

In the moutain car problem, a car is stuck in a valley and the goal is to 'escape' this valley by climbing up the right side. We assume that the gravity is stronger



than the car's engine, and even at full throttle the car cannot accelerate up the steep moutain road. Thus the only solution is to first move away from the goal and up opposite slope on the left. Then by applying full throttle the car can build up enough inertia to carry it up the steep slope. This problem can be modeled as finding the optimal policy in the following MDP:

- The state space $\mathcal{S} = [-1.2; 0.6] \times [-0.07; 0.07]$ where the first dimension represent the position and the second the velocity of the car.
- The action space $\mathcal{A} = \{-1, 0, 1\}$ where 1 means full throttle forward, -1 for full throttle reverse and 0 for zero throttle.
- The reward is always $r(s, a, s') = r(s) = -1$ except when the position is 0.6. The position 0.6 is a terminal state.
- The dynamic of the system (transition) is described as follows. Assume that you are in state $s_t = (x_t, v_t)$ and you did action a. Then you move to the next state $s_{t+1} = (x_{t+1}, v_{t+1})$ given by :

$$\begin{cases} v_{t+1} &= \max\{\min\{v_t + \epsilon_t + 0.001 a_t - 0.0025\cos(3x_t), 0.07\}, -0.07\}, \\ x_{t+1} &= \max\{\min\{x_t + v_t, 0.6\}, -1.2\}, \end{cases}$$

  with the extra condition that the position $x = -1.2$ is an inelastic wall: if the car reaches this point, its speed is set to zero.

One can consider both the deterministic case in which $\epsilon_t = 0$ and the stochastic case in which ϵ_t is drawn uniformly at random in $[-0.0005, 0.0009]$.

Controlling this car to drive it out of the valley is a reinforcement learning problem since the driver is in general not aware of the precise dynamics of the problem and the transition can be regarded as unknown. Note that even in the plannification problem (where you know the transition, as we do), the usual algorithms (IP,IV) cannot be applied since the state space is infinite (or very big if discretized).

The goal of the TP is to compute and plot the optimal value function V^*.

Q : What do you think the optimal value function looks like?

2 Modeling

1. Write a function

   `[sn,rec]=simulator(s,a)`

   that returns the next state and the rewards obtained when we are in state $s = (x, v)$ and choose action a.

2. Our goal is to build an approximation of the Q-value function in the following space:

$$\mathcal{E}_Q = \{f : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \ : \ f(s,a) = \sum_{i=1}^{k} \alpha_i \phi_i(s,a), \text{ where } \alpha \in \mathbb{R}^k\}$$

   where k is the number of features. We choose to work with gaussian features:

$$\phi_i(s,a) = \frac{1}{2\pi\sqrt{|\Sigma_{i,a}|}} \exp\left(-\frac{1}{2}(s-\mu_{i,a})\Sigma_{i,a}^{-1}(s-\mu_{i,a})'\right).$$

   In our implementation, each feature ϕ is represented by a lign vector *theta*, as you can check in the given function $phiQ(s, a, theta)$ that return the value in (s, a) of the feature function parametrized by *theta*.

   Q: How are the caracteristics of a feature function ϕ stored in its corresponding vector *theta*?

3. The given function $[thetas] = randfeatureQ(k)$ builds k features chosen at random. Note that there are other possible choices for the features.

4. The given function $createQ(alpha, thetas)$ returns a function Q which is the linear combination of the features stored in *thetas* with coefficients given by the vector *alpha*.

   One can visualize functions in $\mathcal{E}_Q$ which the command *plotf.m* (given).

3 Approximate Value Iteration: Fitted-Q

Implement the Fitted-Q iteration algorithm using the space $\mathcal{E}_Q$ to approximate the optimal Q-value function. Especially, you have to understand that the algorithm reduces to solving a linear regression problem at each round.

The optimal value function V^* can be obtained from Q^* by using the command

```
V = @(s) max(arrayfun(@(a)Q(s,a),[-1,0,1]))
```

You can plot this function using the *plotf.m* function.

4 Appoximate Policy Iteration: LSTD

Approximate Policy Iteration is based on an estimate V_k of V^{π_k} (or an estimate Q_k of Q^{π_k}), often based on observing trajectories under policy π_k. Then π_{k+1} is the greedy policy with respect to this estimate.

The LSTD algorithm computes Q_{k+1} as the solution of $\min_{\mathcal{F}} ||T^{\pi_k}Q - Q||_{\pi_k}$, which leads to the resolution of a linear system to find the coefficient α for Q_{k+1}; $A\alpha = b$ with

$$\left\{ \begin{array}{rcl} A_{i,j} & = & \langle \phi_i \mid \phi_j - \gamma P^\pi \phi_j \rangle_{\mu_\pi} \\ b_i & = & \langle \phi_i \mid r^\pi \rangle_{\mu_\pi}, \end{array} \right.$$

with $\pi = \pi_k$. These coefficients can be estimated along a trajectory of length n.

Implement the LSTD algorithm with the approximation space $\mathcal{E}_Q$ and plot the optimal value function.

Assignment: For both algorithms, send me a 'reasonable' approximation of V^* that you obtained. Mention the number of features used (the way they were chosen) and the number of iterations of the algorithm needed to obtain the graphic. Which algorithm is easiest to implement/gives the best results?