CS 287, Fall 2011
Problem Set #1 : Markov Decision Processes, Value Iteration, Linear Programming, LQR, Function Approximation

---

Deliverable: pdf write-up by Monday September 24th, 23:59pm to be emailed to pabbeel@cs.berkeley.edu, with subject: PS1. Some of the starter code will generate plots. Don't include all of them, include whatever is sufficient to show that you correctly solved the problem. Whenever including a figure, make sure it is accompanied by a discussion of that figure. Ball-park figure for number of pages: 6.

Refer to the class webpage for the homework policy.

Various starter files are provided on the course website.

You are free to use any linear programming solver of your choice for this assignment. If you don't have in-depth experience with a particular solver already, I recommend you try out CVX. CVX can solve a broad class of convex optimization problems, including linear programs, but also quadratic programs (which you will need for future assignments!). The file cvx_intro.m provides download/installation instructions, as well as a few examples of problems being solved with CVX.

1. Markov Decision Processes – Modeling

(a) **Shortest Path in a Graph.** Consider the problem of finding the shortest path in a graph (V, E) (with V the set of vertices, and E the set of directed edges), and where v_G is the destination vertex. Describe how this problem can be encoded into a Markov decision process, (S, A, T, γ, R), such that the optimal solution in the Markov decision process is the shortest path in the graph.

(b) **Queuing.** Consider a server that has to serve three queues denoted by a, b, c. Each queue can be of length zero through five. At each time, for each queue that is shorter than five (already to start out with, or per a request being serviced in that queue) a request is added to the queue with probability p_a, p_b, p_c respectively. This happens independently for each of the queues. At any given time, the server gets to choose between continuing to serve the queue it is currently serving, in which case it can serve one request in that queue, or to switch to another queue. Switching queues takes up an entire time-step, hence when switching no request is serviced in that time-step. Serving a request results in a reward of $+1$, which is valued as γ^t if it happens at time t. Describe an MDP, (S, A, T, γ, R), that models this setting.

2. Value Iteration

(a) Implement value iteration into value_iteration.m. Run value iteration for the provided grid-world and report the values obtained for each state. (You can check your results against the ones provided in the starter-code to ensure correctness!)

(b) Implement an MDP for the queuing problem from 1 (a) for $p_a = 0.1, p_b = 0.2, p_c = 0.5, \gamma = 0.9$. Report the values and the optimal actions for the following states: $(0, 0, 0, a), (2, 0, 4, b), (5, 2, 1, c)$.

3. Solving MDPs with Linear Programming

(a) **Implementation.** Implement the linear programming based solution to MDPs into linear_programming.m. Solve again the two MDPs you solved in the previous question, and verify that you obtain the same results. Report that you indeed get the same results, or, if not, report the results you obtained with LP formulations the same way you did for value iteration.

(b) **Theory.** In class we covered the linear programming formulation (both primal and dual) that solves infinite horizon MDPs with discounting, which you have used for the previous questions. Write down both the primal and the dual LP formulations for the finite horizon setting, with horizon H.

4. Function Approximation

We will study the application of linear programming with function approximation to building a control policy for the game of tetris. The file main_ALP_starter.m illustrates the use of the starter files available to you. There should be no need to investigate other files, though you might have some fun with main_play_and_collect_data.m.

The game of tetris has a very large state space. We will use a linear function approximator with the standard set of 22 features. A function computing them from a board configuration is provided (tetris_standard_22_features.m). Our resulting value function approximation for a board configuration b is of the form $V(b) = \theta^\top \phi(b)$. As this function approximation ignores the current block, we are going to use a slightly altered Bellman equation:

$$\begin{aligned} V(b) &= \frac{1}{K}\sum_{k=1}^{K} V(b,k) \\ V(b,k) &= \max_a R(b'(b,k,a)) + \gamma V(b'(b,k,a)) \end{aligned}$$

where $k = 1, \ldots, K$ indexes over the different blocks, $V(b,k)$ is the value of board configuration b with current block k, a indexes over all possible actions (the product space of 4 rotations and 10 translations), and b' is the next board configuration achieved when starting from b with block k and taking action a.

(a) Write out the approximate linear program you obtain with this new formulation.

(b) Implement the approximate linear programming (ALP) approach. Per the constraint sampling: your starter file loads a log of a couple of games I played. Investigate how well the ALP approach works as a function of the number of constraints being sampled (when subsampling, it would be most natural to subsample equally spaced, maximizing the diversity of board configurations),[1] As your evaluation metric for each of the settings, run the resulting policy (i.e, the greedy policy w.r.t. the found value function) 20 times and report the mean and standard error of the number of blocks placed. To make your comparisons more meaningful, compare on the same 20 sequences of blocks.

(c) Discuss performance as a function of the choice of discount factor. Same evaluation metric as above.

[1] On my laptop, it took 200s for CVX to parse (most of the time) and solve the ALP when sampling every 5'th board configuration for the constraint set, and 68s when sampling every 10'th board.

(d) For this subquestion only, change the reward function. The default reward function gives a reward of 20 − maxheight, i.e., the distance of the highest filled square from the top of the board, and also zero for game-over. Max height is feature $\phi(20)$ in the standard list of 22 features. Change it to reward of 1 for every block being placed, a reward of zero when the "game-over" state has been reached. With the default set of game states I provided you with, you should see that this did not work–how can you tell from the values you found as the solution of the LP? Would there be a choice of states that would still enable learning with this simpler reward function? Discuss why you believe so, but no need to implement with such set of states.

(e) Investigate one of the following three extensions:

(i) **Bootstrapping.** After solving the ALP use the greedy policy w.r.t. the obtained value function to sample states. Then re-solve the ALP with the newly sampled states and iterate.

(ii) **Feature selection.** Generate a very large vocabulary of features. Investigate how the ALP performs with an increasing number of features. (You might want to add some form of regularization to avoid overfitting.)

(iii) **Smoothed ALP.** Recently the "smoothed ALP" has been proposed. Modify your code to use this smoothed version and compare performance with the original formulation. (See, `http://web.mit.edu/~vivekf/www/papers/salp.pdf` for details on the smoothed ALP.)

5. LQR

(a) **LQR for a Linear System.** In this question you will implement and evaluate LQR for a linear system. See p_a_starter.m for detailed instructions.

(b) **LQR-based Stablization for a Nonlinear System, Cartpole.** In this question you will implement and evaluate the application of LQR to stablization of a nonlinear system. See p_b_starter.m for detailed instructions.

(c) **LQR-based Stabilization of a Helicopter in Hover.** In this question you will implement and evaluate the application of LQR to stablization of a helicopter in hover. See p_c_starter.m for detailed instructions.

(d) Consider the following optimal control problem, which considers a linear system with additive noise and quadratic cost:

$$\begin{aligned} \min_{x,u} \quad & \mathrm{E}[\sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t] + \mathrm{E}[x_T^\top Q x_T] \\ \text{s.t.} \quad & x_{t+1} = Ax_t + Bu_t + w_t, \quad \forall t = 0, 1, 2, \ldots, T-1 \end{aligned}$$

with w_t independent random vectors with $\mathrm{E}[w_t] = 0$, and $\mathrm{E}[w_t w_t^\top] = \Sigma_w$.

Find an LQR-like sequence of matrix updates that computes the optimal cost-to-go at all times and the optimal feedback controller at all times. Describe the expected cost incurred in excess of the expected cost in the case when there is no noise.

(e) Consider the following optimal control problem, which considers a linear system with multiplicative noise and quadratic cost:

$$\begin{aligned} \min_{x,u} \quad & \mathrm{E}[x_T^\top Q x_T] \\ \text{s.t.} \quad & x_{t+1} = Ax_t + (B + W_t)u_t, \quad \forall t = 0, 1, 2, \ldots, T-1 \end{aligned}$$

Here $Q \in \mathbb{R}^{n_X \times n_X}, A \in \mathbb{R}^{n_X \times n_X}, B \in \mathbb{R}^{n_X \times n_U}$ are given and fixed. $W_t \in \mathbb{R}^{n_X \times n_U}, t = 0, 1, \ldots, T-1$ are independent random matrices with $\mathrm{E}[W_t] = 0$. Higher-order expectations involving W_t will show up. These higher-order expectations are *not* assumed to be zero, and you should just keep these expectations around—no need to try to simplify these (and not possible anyway unless additional assumptions were made).

Find an LQR-like sequence of matrix updates that computes the optimal cost-to-go at all times and the optimal feedback controller at all times.