CS 533 Homework 2
Optimizing Finite-Horizon Expected Total Reward

Eugene Seo

April 17, 2017

Part I: Build a Planner.

- Run file: python finite.py *MDPFile H*

  INPUTs: *MDPFile* and *H*
  1. *MDPFile* - text file name that describes an MDP (e.g. MDP.txt, MDP1.txt, ...)
  2. *H* - a time horizon (positive integer)

  OUTPUTs: A text file named ' *output_H_MDPFile*' containing followings
  1. Optimal non-stationary value function (V), n $\times$ *H* matrix
  2. Optimal non-stationary policy, n $\times$ *H* matrix
  ($\star$ n is the number of states)

- Sample run: python finite.py sampleMDP.txt 5

  Inputs: sample MDP and *H* (=5)

```
3 2

0.2 0.8 0.0
0.0 0.2 0.8
1.0 0.0 0.0

0.9 0.05 0.05
0.05 0.9 0.05
0.05 0.05 0.9

-1.0 -1.0 0.0
```

  Outputs: output_5_sampleMDP.txt containing optimal V and policy

```
V
-1.0000 -1.9500 -2.3500 -2.5260 -2.6748
-1.0000 -1.2000 -1.3200 -1.4620 -1.6174
0.0000  -0.1000 -0.2475 -0.4063 -0.5650

Policy
0 2 1 1 1
0 1 1 1 1
0 2 2 2 2
```

($\star$ column i means i steps-to-go)

- Finite-Horizon Policy Optimization Algorithm

Algorithm 1 Finite-Horizon Policy Optimization

```
def bellman_backup(state_size, action_size, T, s, V, t):
    future_values_by_actions = [] # list of expected future values with each actions
    for a in range(0, action_size):
        future_values = 0 # accumulated future value by an action
        for s_next in range(0, state_size):
            value_s_next = T[a,s,s_next] * V[s_next,t-1] # value by the next state
            future_values = future_values + value_s_next
        future_values_by_actions.append(future_values)

    max_value = np.max(future_values_by_actions)
    action = np.argmax(future_values_by_actions) + 1 # action number starting from 1

    return max_value, action

def policy_optimization(state_size, action_size, T, R, H):
    V = np.zeros((state_size, H))
    policy = np.zeros((state_size, H))

    V[:,0] = R # rewards for 0 step-to-go

    for t in range(1, H):
        for s in range(0, state_size):
            max_value, action = bellman_backup(state_size, action_size, T, s, V, t)
            V[s,t] = R[s] + max_value
            policy[s,t] = action
```

The complexity of computation of this algorithm is $O(Hmn^2)$ where H is the time horizon, m is the size of actions, and n is the size of states.

Part 2: Create your own MDP.

In my simple MDP, there are 20 states (s1 - s20), and two actions (a1 and a2). Just like the sample MDP seen in the previous problem, every state in my MDP has a higher probability to go to the next state, the state number of which is one greater than the current state number, when we take action 1 (a1). When we take action 2 (a2) on each state, all states have a higher probability to stay in the current state than to move to any other states.

- The transition function for action 1.

$$T(s_i, a1, s_j) = \begin{cases} 0.1, & j = i \\ 0.9, & j = i + 1 \\ 0, & \text{otherwise} \end{cases}$$

a1	s1	s2	s3	...	s9	s10	s11	...	s18	s19	s20
s1	0.1	0.9	0		0	0	0		0	0	0
s2	0	0.1	0.9		0	0	0		0	0	0
s3	0	0	0.1		0	0	0		0	0	0
...											
s8	0	0	0		0.9	0	0		0	0	0
s9	0	0	0		0.1	0.9	0		0	0	0
s10	0	0	0		0	0.1	0.9		0	0	0
...											
s18	0	0	0		0	0	0		0.1	0.9	0
s19	0	0	0		0	0	0		0	0.1	0.9
s20	0.9	0.1	0		0	0	0		0	0	0

- The transition function for action 2.

$$T(s_i, a2, s_j) = \begin{cases} 0.9, & j = i \\ 0.1, & j = i + 1 \\ 0, & \text{otherwise} \end{cases}$$

a2	s1	s2	s3	...	s9	s10	s11	...	s18	s19	s20
s1	0.9	0.1	0		0	0	0		0	0	0
s2	0	0.9	0.1		0	0	0		0	0	0
s3	0	0	0.9		0	0	0		0	0	0
...											
s8	0	0	0		0.1	0	0		0	0	0
s9	0	0	0		0.9	0.1	0		0	0	0
s10	0	0	0		0	0.9	0.1		0	0	0
...											
s18	0	0	0		0	0	0		0.9	0.1	0
s19	0	0	0		0	0	0		0	0.9	0.1
s20	0.1	0	0		0	0	0		0	0	0.9

- The reward for each state.

  The highest reward, 1 is assigned to the last state 20, and small amount of reward, 0.01 is assigned to state 10.

	s1	s2	s3	...	s9	s10	s11	...	s18	s19	s20
R	0	0	0	0	0	0.01	0	0	0	0	1

- The expected optimal policy

  Since the highest reward is given us when we are in state 20, the goal should be to get to state 20 and stay there as much as possible. Thus, the optimal policy will be taking action 2 when in state 20 and taking action 1 otherwise in order to move to the next state, number of which is one greater than the current state, and to get closer to the state 20, assuming we have a plenty of time to arrive at the state 20 from any starting points. However, if we start from state 10 and the time horizon is too limited to get to the state 20, we are rather better to stay at state 10 as earning the small amount of reward, 0.01, than moving forward without earning any rewards during the remaining time. That is, the expected optimal solution will be taking action 1 when we are in state 10 and we have at least 10 remaining time, otherwise we will stay in the state 10 by taking action 2.

- The results of optimal value function and policy

  1. 1st horizons H1 (=5): Run "python finite.py myMDP.txt 5"

```
V
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0066
0.0000  0.0000  0.0000  0.0073  0.0160
0.0000  0.0000  0.0081  0.0170  0.0253
0.0000  0.0090  0.0180  0.0262  0.0336
0.0100  0.0190  0.0271  0.0344  0.0410
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000  0.6561
0.0000  0.0000  0.0000  0.7290  1.6038
0.0000  0.0000  0.8100  1.7010  2.5272
```

```
0.0000  0.9000  1.8000  2.6190  3.3570
1.0000  1.9000  2.7100  3.4390  4.0951

Policy
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 2 2 2 2
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 2 2 2 2
```

2. 2nd horizons H2 (=20): Run "python finite.py myMDP.txt 20"
(* V values have been rounded to account for space.)

```
V
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 3 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 3 4 5
0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 3 4 5 5
0 0 0 0 0 0 0 0 0 0 0 1 1 2 3 3 4 5 5 6
0 0 0 0 0 0 0 0 0 0 1 1 2 3 4 4 5 5 6 6
0 0 0 0 0 0 0 0 0 1 1 2 3 4 4 5 5 6 6 6
0 0 0 0 0 0 0 0 1 2 2 3 4 4 5 5 6 6 6 7
0 0 0 0 0 0 0 1 2 2 3 4 4 5 5 6 6 6 7 7
0 0 0 0 0 0 1 2 3 3 4 4 5 5 6 6 6 7 7 7
0 0 0 0 0 1 2 3 3 4 4 5 5 6 6 7 7 7 7 8
```

```
0 0 0 0 1 2 3 3 4 5 5 5 6 6 7 7 7 7 8 8
0 0 0 1 2 3 3 4 5 5 6 6 6 7 7 7 7 8 8 8
0 0 1 2 3 4 4 5 5 6 6 6 7 7 7 7 8 8 8 8
1 1 2 3 4 4 5 5 6 6 6 7 7 7 7 8 8 8 8 8

Policy
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

As I expected, in state 10, the optimal policy is taking action 2 when we have less than 10 remaining steps. Otherwise we are taking action 1 to move to the state 20 and keep staying state 20 once we get there by taking action 2.

Part 3: More Testing.

Run my code on given MDPs (MDP1.txt & MDP2.txt) and provide the resulting policies and value functions for a horizon of 10.

1. MDP1: Run “python finite.py MDP1.txt 10”

```
V
0.0000  1.0000  1.0000  1.0330  2.0000  2.0000  2.0330  3.0000  3.0000  3.0330
0.0000  0.0000  0.8944  0.9998  1.0184  1.8944  1.9998  2.0184  2.8944  2.9998
0.0000  0.0074  0.7987  0.8982  1.0126  1.7992  1.8982  2.0126  2.7992  2.8982
0.0000  0.0000  0.8944  0.9039  1.0184  1.8944  1.9039  2.0184  2.8944  2.9039
1.0000  1.0000  1.0330  2.0000  2.0000  2.0330  3.0000  3.0000  3.0330  4.0000
0.0000  0.0000  0.6595  0.8944  0.9998  1.6527  1.8944  1.9998  2.6524  2.8944
0.0000  0.6595  0.8806  0.9812  1.6527  1.8784  1.9804  2.6524  2.8783  2.9804
```

```
0.0000  0.0000  0.8522  0.8960  1.0125  1.8524  1.8958  2.0125  2.8524  2.8958
0.0000  0.0330  1.0000  1.0000  1.0330  2.0000  2.0000  2.0330  3.0000  3.0000
0.0000  0.8944  0.9039  1.0184  1.8944  1.9039  2.0184  2.8944  2.9039  3.0184

Policy
0 4 4 4 4 4 4 4 4 4
0 1 4 2 4 4 2 4 4 2
0 3 3 2 3 3 2 3 3 2
0 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1
0 1 1 3 3 1 3 3 1 3
0 2 2 2 2 2 2 2 2 2
0 1 3 3 3 3 3 3 3 3
0 4 2 2 2 2 2 2 2 2
0 4 1 4 4 1 4 4 1 4
```

2. MDP2: Run "python finite.py MDP2.txt 10"

```
V
0.0000  0.0569  1.0000  1.0569  2.0000  2.0569  3.0000  3.0569  4.0000  4.0569
0.0000  0.0000  0.9925  0.9950  1.9919  1.9952  2.9919  2.9952  3.9919  3.9952
0.4931  1.2527  1.5991  2.2970  2.6193  3.3062  3.6236  4.3081  4.6244  5.3085
0.0000  0.0006  0.0665  0.9936  1.0419  1.9929  2.0420  2.9928  3.0419  3.9927
0.0000  0.5715  1.0000  1.5715  2.0000  2.5715  3.0000  3.5715  4.0000  4.5715
0.0000  1.0000  1.0002  2.0000  2.0003  3.0000  3.0003  4.0000  4.0003  5.0000
1.0000  1.0000  2.0000  2.0002  3.0000  3.0003  4.0000  4.0003  5.0000  5.0003
0.0000  0.0775  0.0816  0.9657  1.0813  1.9648  2.0813  2.9648  3.0813  3.9648
0.0000  0.0112  0.5715  1.0000  1.5715  2.0000  2.5715  3.0000  3.5715  4.0000
0.0000  0.9999  1.0000  2.0000  2.0003  3.0000  3.0003  4.0000  4.0003  5.0000

Policy
0 4 1 4 1 4 1 4 1 4
0 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1
0 2 3 2 2 2 2 2 2 2
0 3 2 3 2 3 2 3 2 3
0 3 1 3 1 3 1 3 1 3
0 1 3 3 3 3 3 3 3 3
0 4 2 2 2 2 2 2 2 2
0 4 2 1 2 1 2 1 2 1
0 3 3 3 3 3 3 3 3 3
```