HW LOG

CME241

Justin Lundgren, 06289145
justinlundgren@stanford.edu
Stanford University

HW – Jan 11

- Write out the MP/MRP definitions and MRP Value Function definition (in LaTeX) in your own style/notation (so you really internalize these concepts)
- Think about the data structures/class design (in Python 3) to represent MP/MRP and implement them with clear type declarations
- Remember your data structure/code design must resemble the Mathematical/notational formalism as much as possible
- Specifically the data structure/code design of MRP should be incremental (and not independent) to that of MP
- Separately implement the $r(s, s')$ and the $R(s) = \sum_{s'} p(s, s') * r(s, s')$ definitions of MRP
- Write code to convert/cast the $r(s, s')$ definition of MRP to the $R(s)$ definition of MRP (put some thought into code design here)
- Write code to generate the stationary distribution for an MP

MP/MRP definition

MP: A markov process is a chain that is memory less, i.e. it only cares about about the current state and not the past. The mathematical definition is

$$\mathbb{P}(S_{t+h} = s_{t+h} \big| S_0 = s_0, S_1 = s_1, \ldots, S_{t-1} = s_{t-1}, S_t = s_t) = \mathbb{P}(S_{t+h} = s_{t+h} \big| S_t = s_t)$$

or equivalently

$$\mathbb{E}[S_{t+h} \big| S_0 = s_0, S_1 = s_1, \ldots, S_{t-1} = s_{t-1}, S_t = s_t] = \mathbb{E}[S_{t+h} \big| S_t = s_t].$$

The Markov process is defined as $\{s, P_s\}$ where $s \in \{s_0, \ldots, s_k\}$ is the state spaces and P_s is the probability distribution in each state.
MRP: A Markov reward process is a Markov process that has a reward $R(s)$ associated with each state and some discounting factor $\gamma \in [0, 1]$.
Value function: The value function is the accumulated expected reward associated with the current known state s. It is defines as

$$v(s) = \mathbb{E}[\sum_{i=0}^{T} R(s_{t+i})\gamma^i \big| S_t = s],$$

where T is the time of termination for the process.

Data structures

- *State* – TypeVar('State')
- *States* – List[State]
- $\mathcal{R}(s)$ – List[float] (*)
- $r(ss')$ – List[List[float]] (*)
- P_{MP} – Dict[State,Tuple[State,float]]
- P_{MRP_A} – Dict[P_{MP},float]
- P_{MRP_B} – Dict[State,Dict[State,Tuple[float,float]]]
- γ – float.

Thus we see that

$$\begin{aligned}
\mathcal{R}(s) &= \mathbb{E}[R_t|S_{t-1}=s] \\
&= \sum_{s'} R_t(\{\texttt{reward after state } s'\})\mathbb{P}(S_t=s'|S_{t-1}=s) \\
&= \sum_{s'} \mathbb{E}[R_t|S_{t-1}=s \ \cap \ S_t=s']\mathbb{P}(S_t=s'|S_{t-1}=s) \\
&= \sum_{s'} r(s,s')p(s,s')
\end{aligned}$$

HW – Jan 16

- Write the Bellman equation for MRP Value Function and code to calculate MRP Value Function (based on Matrix inversion method you learnt in this lecture)
- Write out the MDP definition, Policy definition and MDP Value Function definition (in LaTeX) in your own style/notation (so you really internalize these concepts)
- Think about the data structure/class design (in Python 3) to represent MDP, Policy, Value Function, and implement them with clear type definitions
- The data structure/code design of MDP should be incremental (and not independent) to that of MRP
- Separately implement the $r(s, s', a)$ and $R(s, a) = \sum_{s'} p(s, s', a) * r(s, s', a)$ definitions of MDP
- Write code to convert/cast the $r(s, s', a)$ definition of MDP to the $R(s, a)$ definition of MDP (put some thought into code design here)
- Write code to create a MRP given a MDP and a Policy
- Write out all 8 MDP Bellman Equations and also the transformation from Optimal Action-Value function to Optimal Policy (in LaTeX)

Data structures

- *Action* – TypeVar('Action')
- *Policy* – Dict[State,Tuple[Action,(float or int)]]
- MDP_A – Dict[State,Dict[Action,Dict[Tuple[State,(float or int)],(float or int)]]]
- MDP_B – Dict[State,Dict[Action,Tuple[State,Tuple[float,float]]]

Bellman Equations

(1) Basic Bellman for MRP (A)

$$\begin{aligned} v(s) &= \mathbb{E}[\sum_{i=0}^{T} R_{t+i+1}\gamma^i \big| S_t = s] \\ &= \mathbb{E}[R_{t+1} \big| S_t = s] + \gamma \mathbb{E}[v(S_{t+1}) \big| S_t = s] \\ &= \mathcal{R}_s + \gamma \sum_{s'} v(s')\mathbb{P}(S_{t+1} = s' \big| S_t = s). \end{aligned}$$

(2) Basic Bellman for MRP (A) in matrix form is then

$$v = \mathcal{R} + \gamma \mathcal{P} v.$$

(3) For the action-value function with policy π we have

$$q_\pi(s, a) = \mathbb{E}[\sum_{i=0}^{T} R_{t+i+1}\gamma^i \big| S_t = s \,\cap\, A_t = a]$$

which have the same solution in

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[R_{t+1} \big| S_t = s \,\cap\, A_t = a] + \gamma \mathbb{E}[q_\pi(S_{t+1}, A_{t+1}) \big| S_t = s \,\cap\, A_t = a] \\ &= \mathcal{R}_s^a + \gamma \sum_{s', a'} q_\pi(s', a')\mathbb{P}(S_{t+1} = s' \,\cap\, A_{t+1} = a' \big| S_t = s \,\cap\, A_t = a) \end{aligned}$$

where $\mathcal{R}_s^a$ is $\mathcal{R}_s$ for action a.

(4) Likewise for a MDP with a policy π we can create a value MRP with value function

$$v_\pi(s) = \sum_{a'} \pi(a'|s) q_\pi(s, a')$$

where $\pi(a'|s)$ is the probability of taking action a' in state s.

(5) Now, we can combine (3) and (4) to express $v_\pi(s)$ as

$$v_\pi(s) = \sum_{a'} \pi(a'|s)\Big(\mathcal{R}_s^a + \gamma \sum_{s'} \mathbb{P}(S_t = s'|S_t = s \,\cap\, A_t = a')v_\pi(s')\Big)$$

(6) Combining (4) and (5) we can express $q_\pi(s, a)$ as

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathbb{P}(S_{t+1} = s'\big|S_t = s \,\cap\, A_t = a) \sum_{a'} \pi(a'|s')q_\pi(s', a').$$

HW – Jan 18

- Write code for Policy Evaluation (tabular) algorithm
- Write code for Policy Iteration (tabular) algorithm
- Write code for Value Iteration (tabular) algorithm
- Those familiar with function approximation (deep networks, or simply linear in featues) can try writing code for the above algorithms with function approximation (a.k.a. Approximate DP)

All done – check

HW – Jan 23

- Work out (in LaTeX) the equations for Absolute/Relative Risk Premia for CARA/CRRA respectively
- Write the solutions to Portfolio Applications covered in class with precise notation (in LaTeX)

CARA

For CARA we have

$$U(x) = -\frac{1}{a}e^{-ax},\ a \neq 0.$$

Thus we have

$$\frac{dU(x)}{dx} = e^{-ax} \text{ and } \frac{d^2U(x)}{dx^2} = -ae^{-ax}.$$

For the Arrow-Pratt risk aversion coefficient A we have

$$\begin{aligned} A &= -\frac{U''(x)}{U'(x)} \\ &= a. \end{aligned}$$

CRRA

For CRRA we have

$$U(x) = \frac{x^{1-\gamma}}{1-\gamma}\ \gamma \neq 1.$$

Thus we have

$$\frac{dU(x)}{dx} = x^{-\gamma} \text{ and } \frac{d^2U(x)}{dx^2} = -\gamma x^{-\gamma-1}$$

For the relative Arrow-Pratt risk aversion coefficient A we have

$$\begin{aligned} A &= -\frac{xU''(x)}{U'(x)} \\ &= \gamma. \end{aligned}$$

Portfolio Application Solution

CARA: We have two assets $r_a \sim N(\mu, \sigma^2)$ and $r_f \sim N(r, 0)$. We invest a fraction ρ_a in r_a and ρ_f in r_f. The objective is then to

$$\begin{aligned} \max\ & \mathbb{E}[U(N(\rho_a\mu + \rho_f r, \rho_r^2\sigma^2))] \\ \text{s.t. } & \rho_a + \rho_f = 1. \end{aligned}$$

Now, substituting $\rho_f = 1 - \rho_a$ and using the PDF of the normal distribution we can set this up as

$$\max_{\rho_a}\Big\{ -\frac{1}{a}\int_{\mathbb{R}} \exp(-ax)\frac{1}{\sqrt{2\pi\rho_a\sigma^2}}\exp\big(\frac{\big(x-(\rho_a\mu+(1-\rho_a)r)\big)^2}{2\rho_a^2\sigma^2}\big)\Big\}. \tag{1}$$

Differentiating (1) wrt ρ_a and setting to zero gives

$$\rho_a^* = \frac{\mu - r}{a\sigma^2}.$$

CRRA: The setup is very similar but now we assume that $\log(r_a) \sim N(\mu, \sigma^2)$ instead. This gives the solution

$$\rho_a^* = \frac{\mu - r}{\gamma\sigma^2}$$

as optimal allocation.

HW – Jan 25

- Model Merton's Portfolio problem as an MDP (write the model in LaTeX)
- Implement this MDP model in code
- Try recovering the closed-form solution with a DP algorithm that you implemented previously

Discretization of the model:

For a stock S we have that

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where W_t is a Brownian motion, i.e. a simple random walk with infinitely small steps size. Solving this sde gives for a future time T given a current time t gives

$$S_T = S_t \exp\Big((\mu - \frac{\sigma^2}{2})(T-t) + \sigma W_{(T-t)}\Big)$$

or more simply

$$S_T = S_t \exp\Big((\mu - \frac{\sigma^2}{2})(T-t) + \sigma\sqrt{T-t}Z\Big) \tag{2}$$

where $Z \sim N(0,1)$. Now, if we have the wealth W_t at time t, we consume c_t thus and out of the remaining wealth $W_t - c_t$ we invest π_t fractions in a portfolio of risky assets with homoscedastic variance[1] σ^2 and return μ. Consequently we invest $1-\pi_t$ fractions in a risk free asset with constant return r and no variance. With the logic of (2) we can set up the distribution of the future wealth as a time-discrete function of the current wealth and the actions c_t and pi_t as

$$W_{t+\tau} = (W_t - c_t)\exp\Big((\pi_t\mu + (1-\pi_t)r - \frac{\pi_t^2\sigma^2}{2})\tau + \sigma\sqrt{\tau}Z\Big). \tag{3}$$

This will be the discrete setup for the development of the wealth. Furthermore the goal is to find the optimal π_t and c_t at each time, i.e. the π_t and c_t that maximizes the expected utility of the consumption. Thus our goal is to

$$\max_{\pi_t, c_t} \mathbb{E}\Big[\sum_{\tau=0}^{T} e^{-\rho\tau}\frac{c_\tau^{1-\gamma}}{1-\gamma}\Big].$$

Setup:

- States – The state is a tuple $\Big\langle t, W_t\Big\rangle$ of time r and wealth W_t at time t. The times is here discrete with equidistant time partition $\tau = t_i - t_{i-1}$ up until time T of maturity. The wealth is also discrete but it follows the distribution in (3) – more on that later
- Action(s) – The action is also a tuple $\Big\langle c_t, \pi_t\Big\rangle$ of consumption c_t and fraction of risky assets π_t. To make this discrete I will define c_t as $c_t \in \{0, 0.01W_t, 0.02W_t, \ldots, 1.99W_t, 2W_t\}$ with a fixed upper boundary at $2W_t$. The fraction π_t is supposed to be unconstrained, but in order of modelling this with a finite state space, I will probably have to set some upper and lower boundary, e.g. $\pi_t \in \{-2, -1.99, \ldots, 1.99, 2\}$.
- Transitions

[1]Perhaps these parameters should not be constant to make the setup more realistic.

HW – Jan 30

- Model a real-world Portfolio Allocation+Consumption problem as an MDP (including real-world frictions and constraints)
- Exam Practice Problem: Optimal Asset Allocation in Discrete Time

Step 1:

The continuous state space is a tuple $\Big\langle t, W(t) \Big\rangle$ where t denotes the time and $W(t)$ the value of the portfolio. In the discrete case we would have that

$$\begin{aligned} &\frac{W_t}{W_{t-1}} - 1 \sim N(\mu \frac{x_{t-1}}{W_{t-1}} + r\frac{W_{t-1}-x_{t-1}}{W_{t-1}}, (\frac{x_{t-1}}{W_{t-1}})^2\sigma^2) \\ \Rightarrow &W_t \sim N\Big(\big(W_{t-1} + \mu x_{t-1} + r(W_{t-1}-x_{t-1})\big), x_{t-1}^2\sigma^2\Big) \\ \Rightarrow &W_t \sim N\Big((\mu - r)x_{t-1} + (1+r)W_{t-1}, x_{t-1}^2\sigma^2\Big) \end{aligned} \tag{4}$$

In the continuous case this corresponds to

$$S(t) = S(0)\exp(\mu t + \sqrt{t}\sigma Z)$$

where Z is a standard normal distributed variable. The action is $x_t \in \mathbb{R}$ and the discount factor is γ.

Step 2:

For the Bellman optimally equation we have that

$$V_*(\langle t, W_t \rangle) = \max_{x_t} \big\{\mathcal{R}^{x_t}_{\langle t, W_t\rangle} + \gamma \int_{\langle t+1, W'_{t+1}\rangle} \mathcal{P}_{\langle t, W_t\rangle\langle t+1, W'_{t+1}\rangle} V_*(\langle t+1, W'_{t+1}\rangle)\, dW_{t+1}\big\}.$$

However, the rewards at each time $t \in \{0, 1, \dots, T-1\}$ are all zero. Hence $\mathcal{R}^{x_t}_{\langle t, W_t\rangle} = 0 \ \forall \ t \in \{0, 1, \dots, T-1\}$. We also need to take the utility into account. Thus we have that

$$V_*(\langle t, W_t\rangle) = \gamma \max_{x_t}\{\mathbb{E}\big[V_*(\langle t+1, W_{t+1}\rangle)\big]\}$$

where the expected value only depends on the action x_t in each $t \in \{0, 1, \dots, T-1\}$.

Step 3:

If we know that

$$V_*(\langle t, W_t\rangle) = -b_t e^{-c_t W_t},$$

we can now use this and (4) to solve this expected value as

$$\begin{aligned} &- b_{t+1}\int_{\mathbb{R}} e^{-c_{t+1}W_{t+1}} \frac{1}{\sqrt{2\pi x_t^2\sigma^2}} \exp(-\frac{\big(W_{t+1} - ((\mu - r)x_t + (1+r)W_t)\big)^2}{2x_t^2\sigma^2}) dW_{t+1} \\ =&- b_{t+1}\int_{\mathbb{R}} \exp(-\frac{\big(W_{t+1} - ((\mu - r)x_t + (1+r)W_t)\big)^2 + 2W_{t+1}c_{t+1}x_t^2\sigma^2}{2x_t^2\sigma^2}) / \sqrt{2\pi x_t^2\sigma^2}\, dW_{t+1} \\ =&\{\text{Completing the square: } (y-c)^2 + 2yq = \big(y - (c-q)\big)^2 + 2cq - q^2\} \\ =&- b_{t+1}\exp(\frac{-2\big((\mu - r)x_t + (1+r)W_t\big)(c_{t+1}x_t^2\sigma^2) + (c_{t+1}x_t^2\sigma^2)^2}{2x_t^2\sigma^2}) \\ =&- b_{t+1}\exp\big(-\big((\mu - r)x_t + (1+r)W_t\big)c_{t+1} + \frac{1}{2}c_{t+1}^2x_t^2\sigma^2\big). \end{aligned} \tag{5}$$

Step 4:

Differentiating the final step of (5) wrt x_t now gives

$$\frac{\partial V_*(\langle t, W_t\rangle)}{\partial x_t} = \big(c_{t+1}^2 x_t \sigma^2 - (\mu - r)c_{t+1}\big)V_*(\langle t, W_t\rangle)$$

and setting to zero gives that

$$x_t^* = \frac{\mu - r}{c_{t+1}\sigma^2}.$$

Step 5:

Now, using the definitions

$$V_*(\langle t, W_t\rangle) = -b_t e^{-c_t W_t}$$

and

$$\begin{aligned} V_*(\langle t, W_t\rangle) &= -\gamma b_{t+1} e^{-c_{t+1}W_{t+1}} \\ &= -\gamma b_{t+1} \exp\big(-\big((\mu - r)x_t^* + (1+r)W_t\big)c_{t+1} + \frac{1}{2}c_{t+1}^2 x_t^{*2}\sigma^2\big) \\ &= -\gamma b_{t+1} \exp\big(-c_{t+1}(1+r)W_t - \frac{(\mu - r)^2}{2\sigma^2}\big) \end{aligned}$$

we see that

$$c_{t+1}(1+r) = c_t$$

and

$$b_t = \gamma b_{t+1} \exp(\frac{-(\mu - r)^2}{2\sigma^2}).$$

Step 6:

Since

$$U(W_T) = -\frac{e^{-aW_T}}{a}$$

we know that

$$c_T = a$$

and

$$b_T = \frac{1}{a}.$$

Thus we can find a general expression for c_t as

$$c_t = a(1+r)^{T-t}$$

whereas for b_t we have that

$$b_t = \frac{1}{a}\gamma^{T-t} \exp(\frac{-(\mu - r)^2(T-t)}{2\sigma^2}).$$

To conclude, this yields the optimal policy

$$x_t^* = \frac{\mu - r}{a\sigma^2}(1+r)^{t+1-T}.$$

HW – Feb 1

- Implement Black-Scholes formulas for European Call/Put Pricing (jan 30)
- Implement standard binary (jan 30) tree/grid-based numerical algorithm for American Option Pricing and ensure it validates against Black-Scholes formula for Europeans
- Implement Longstaff-Schwartz Algorithm and ensure it validates against binary tree/grid-based solution for path-independent options (jan 30)
- Explore/Discuss an Approximate Dynamic Programming solution as an alternative to Longstaff-Schwartz Algorithm (jan 30)
- Work out (in LaTeX) the solution to the Linear Impact model we covered in class
- Model a real-world Optimal Trade Order Execution problem as an MDP (with complete order book included in the State)

LIM

- $t \in \{t_1, t_2, \ldots, t_n = T\}$
- price at time $t := P_t$
- # shares sold at time $t := N_t$
- # shares remaining at time $t := R_t = N - \sum_{i=1}^{t-1} N_t$, where N is the amount of shares we start of with.
- price is function of previous price and previous amount of shares sold, i.e. $P_{t+1} = f(P_t, N_t) + \epsilon_t$ where ϵ_t is i.i.d. random noise.
- Goal: Maximize $\mathbb{E}[\sum_{i=1}^{T} U\Big(N_i f_i(P_{t-1}, N_{t-1})\Big)]$ where U is the utility.
- MDP-formulation
  - State: $\langle t, P_t, R_t \rangle$
  - Action: N_t
  - Discount factor γ
- Solution:
  - Assume model: $P_{t+1} = P_t + -\alpha N_t + \epsilon_t$
  - Then we can find an optimal policy $N_t^* = R_t/(T - t + 1)$
  - and an optimal value function

$$V^*(t, P_t, R_t) = R_t P_t - \frac{R_t^2}{2}\left(\frac{2\beta + (T-t)\alpha}{T - t + 1}\right)$$

where $Q_t = P_t - g_t(P_t, N_t) = P_t - \beta N_t$.

HW – Feb 6

- Write code for the interface for tabular RL algorithms. The core of this interface should be a mapping from a (state, action) pair to a sampling of the (next state, reward) pair. It is important that this interface doesn't present the state-transition probability model or the reward model.
- Implement a tabular Monte-Carlo algorithm for Value Function prediction
- Implement a tabular TD algorithm for Value Function prediction
- Test the above implementation of Monte-Carlo and TD VF prediction algorithms versus DP Policy Evaluation algorithm on an example MDP
- Prove that fixed learning rate (step size alpha) for MC is equivalent to an exponentially decaying average of episode returns

Assume we initialize $V(S_t)$ to be zero. Then, for the first episode we will have that

$$V(S_t) = \alpha G_t^{\langle 1\rangle} = \alpha\Big(R_1^{\langle 1\rangle} + \gamma R_2^{\langle 1\rangle} + \cdots + \gamma^{t-1} R_t^{\langle 1\rangle}\Big)$$

where $\langle 1\rangle$ denotes the subscript of the first episode. For the second iteration we will have

$$\begin{aligned} V(S_t) &= \alpha G_t^{\langle 1\rangle} + \alpha\Big(G_t^{\langle 2\rangle} - \alpha G_t^{\langle 1\rangle}\Big) \\ &= \alpha G_t^{\langle 2\rangle} + \alpha(1-\alpha) G_t^{\langle 1\rangle}. \end{aligned}$$

For the third iteration we have

$$\begin{aligned} V(S_t) &= \alpha G_t^{\langle 2\rangle} + \alpha(1-\alpha) G_t^{\langle 1\rangle} + \alpha\Big(G_t^{\langle 3\rangle} - (\alpha G_t^{\langle 2\rangle} + \alpha(1-\alpha) G_t^{\langle 1\rangle})\Big) \\ &= \alpha G_t^{\langle 3\rangle} + \alpha(1-\alpha) G_t^{\langle 2\rangle} + \alpha(1-\alpha)^2 G_t^{\langle 1\rangle} \\ &= \alpha \sum_{i=1}^{3} (1-\alpha)^{3-i} G_t^{\langle i\rangle}. \end{aligned}$$

Thus we see a general pattern. Namely that

$$V(S_t) = \alpha \sum_{i=1}^{n} (1-\alpha)^{n-i} G_t^{\langle i\rangle},$$

where n is the amount of episodes.

HW – Feb 13

- Implement Forward-View TD(Lambda) algorithm for Value Function Prediction
- Backward View TD(Lambda), i.e., Eligibility Traces algorithm for Value Function Prediction
- Implement these algorithms as offline or online algorithms (offline means updates happen only after a full simulation trace, online means updates happen at every time step)
- Test these algorithms on some example MDPs, compare them versus DP Policy Evaluation, and plot their accuracy as a function of Lambda
- Prove that Offline Forward-View TD(Lambda) and Offline Backward View TD(Lambda) are equivalent. We covered the proof of Lambda = 1 in class. Do the proof for arbitrary Lambda (similar telescoping argument as done in class) for the case where a state appears only once in an episode.

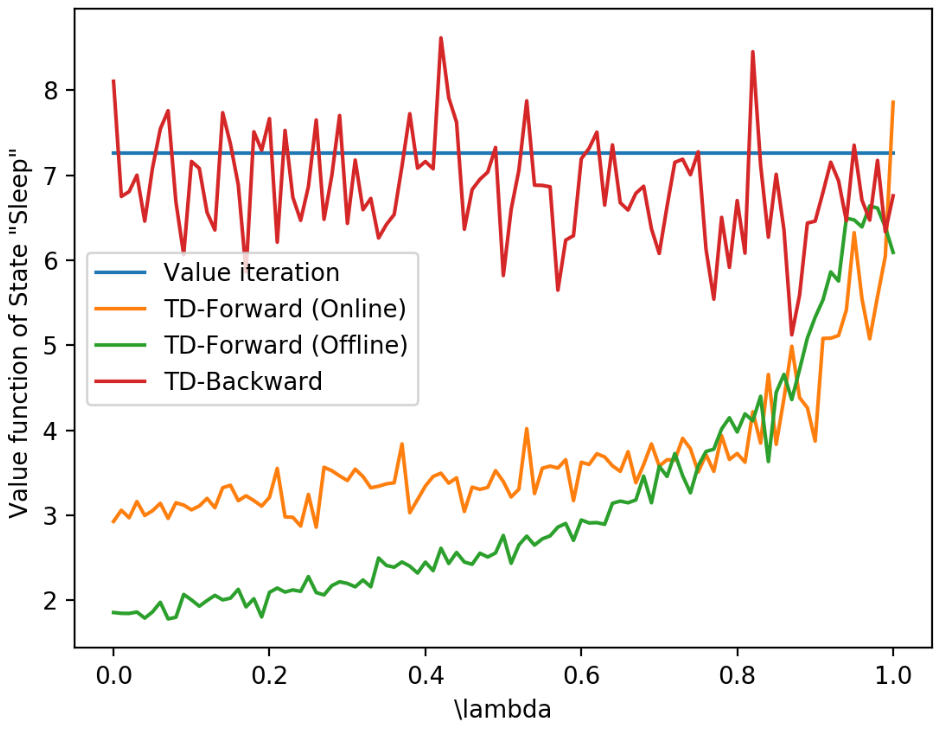


Figure 1: TD(λ) in comparison with value iteration of a simple three state scenario.

Theorem:

$$\sum_{t=1}^{T} \delta_t E_t(s) = \sum_{t=1}^{T} \Big(G_t^\lambda - V(S_t)\Big)\mathbb{I}_{(S_t=s)} \ \forall\ s \in S$$

Proof by induction:

for $T = 1$ we have that

$$\begin{aligned} LHS &= \delta_1 E_1(s) \\ &= \Big(R_2 + \gamma V(S_2) - V(S_1)\Big)\mathbb{I}_{(S_t=s)} \end{aligned}$$

and

$$\begin{aligned} RHS &= \Big(G_1^\lambda - V(S_1)\Big)\mathbb{I}_{(S_t=s)} \\ &= \Big(G_1^1 - V(S_1)\Big)\mathbb{I}_{(S_t=s)} \\ &= \Big(R_2 + \gamma V(S_2) - V(S_1)\Big)\mathbb{I}_{(S_t=s)}. \end{aligned}$$

For $T = T + 1$ we have that

$$\sum_{t=1}^{T} \delta_t E_t(s) + \delta_{T+1} E_{t+1}(s) = \sum_{t=1}^{T} \Big(G_t^\lambda - V(S_t)\Big)\mathbb{I}_{(S_t=s)} + \Big(G_{T+1}^\lambda - V(S_{T+1})\Big)\mathbb{I}_{(S_{T+1}=s)},$$

i.e. if we have that

$$\delta_{T+1} E_{t+1}(s) = \Big(G_{T+1}^\lambda - V(S_{T+1})\Big)\mathbb{I}_{(S_{T+1}=s)},$$

it would conclude the proof. without any further derivation, this holds just as in the first case. Which would conclude the proof if I wrote down all the algebra[2].

[2]http://xuhan.me/2016/10/18/eligibility-trace/

HW – Feb 15

- Prove the Epsilon-Greedy Policy Improvement Theorem (we sketched the proof in Class)
- Provide (with clear mathematical notation) the defintion of GLIE (Greedy in the Limit with Infinite Exploration)
- Implement the tabular SARSA and tabular SARSA(Lambda) algorithms
- Implement the tabular Q-Learning algorithm Test the above algorithms on some example MDPs by using DP Policy Iteration/Value Iteration solutions as a benchmark

ϵ-greedy

This proof is pretty much covered from the slides. However, for an ϵ-greedy policy $\pi_\epsilon(s)$ we have that

$$\begin{aligned} q(s,\pi_\epsilon(s)) &= \sum_{a\in A(s)} \pi_\epsilon(a|s)q(s,a) \\ &= \frac{\epsilon}{|A(s)|}\sum_{a\in A(s)} q(s,a) + (1-\epsilon)\max_{a\in A(s)} q(s,a) \\ &\geq \frac{\epsilon}{|A(s)|}\sum_{a\in A(s)} q(s,a) + (1-\epsilon)\sum_{a\in A(s)} q(s,a)\Big(\frac{\pi(a|s)-\epsilon/|A(s)|}{1-\epsilon}\Big), \end{aligned} \tag{6}$$

where π is some other previously derived ϵ-greedy policy. The claim here is that

$$\max_{a\in A(s)} q(s,a) \geq \sum_{a\in A(s)} q(s,a)\Big(\frac{\pi(a|s)-\epsilon/|A(s)|}{1-\epsilon}\Big).$$

Assume that $A(s) = \{a_1, a_2\}$ and

$$\arg\max_{a\in A(s)} q(s,a) = a_2,$$

then we have that

$$\begin{aligned} \sum_{a\in A(s)} q(s,a)\Big(\frac{\pi(a|s)-\epsilon/|A(s)|}{1-\epsilon}\Big) &= q(s,a_1)\Big(\frac{\epsilon/2-\epsilon/2}{1-\epsilon}\Big) + q(s,a_2)\Big(\frac{1-\epsilon/2-\epsilon/2}{1-\epsilon}\Big) \\ &= q(s,a_2) \end{aligned}$$

and this will be the same for any any length of $A(s)$. The inequality however, will hold when this new iteration has a different maximum action from that of the previous iteration. Since this is justified, we can finally express (6) as

$$\begin{aligned} \frac{\epsilon}{|A(s)|}\sum_{a\in A(s)} q(s,a) + (1-\epsilon)\sum_{a\in A(s)} q(s,a)\Big(\frac{\pi(a|s)-\epsilon/|A(s)|}{1-\epsilon}\Big) &= \sum_{a\in A(s)} \pi(a|s)q(s,a) \\ &= v_\pi(s), \end{aligned}$$

which concludes the proof[3].

GLIE

The acronym is greedy in the limit with infinite exploration. It says that all state-action pairs should be explored infinitely many times and and that all actions should be chosen with a greedy policy wrt to the current estimated Q-function.

[3]https://stats.stackexchange.com/questions/248131/epsilon-greedy-policy-improvement

HW – Feb 20 and 22

- Write code for the interface for RL algorithms with value function approximation. The core of this interface should be a function from a (state, action) pair to a sampling of the (next state, reward) pair. It is important that this interface doesn't present the state-transition probability model or the reward model.
- Implement any Monte-Carlo Prediction algorithm with Value Function approximation
- Implement 1-step TD Prediction algorithm with Value Function approximation
- Implement Eligibility-Traces-based TD(lambda) Prediction algorithm with Value Function approximation
- Implement SARSA and SARSA(Lambda) with Value Function approximation
- Implement Q-Learning with Value Function approximation
- Optional: Implement LSTD and LSPI
- Test the above algorithms versus DP Policy Evaluation and DP Policy Iteration/Value Iteration algorithm on an example MDP

All done – check

HW – Feb 27 and mar 2

- Write with proper notation, the derivations to solutions of Linear Systems for Bellman Error-minimization and Projected Bellman Error-minimization (lecture slides have the derivation, but aim to do it yourself)
- Optional: Write with proper notation, the derivation of Gradient TD (the last two slides of the lecture have the derivation, but aim to do it yourself)

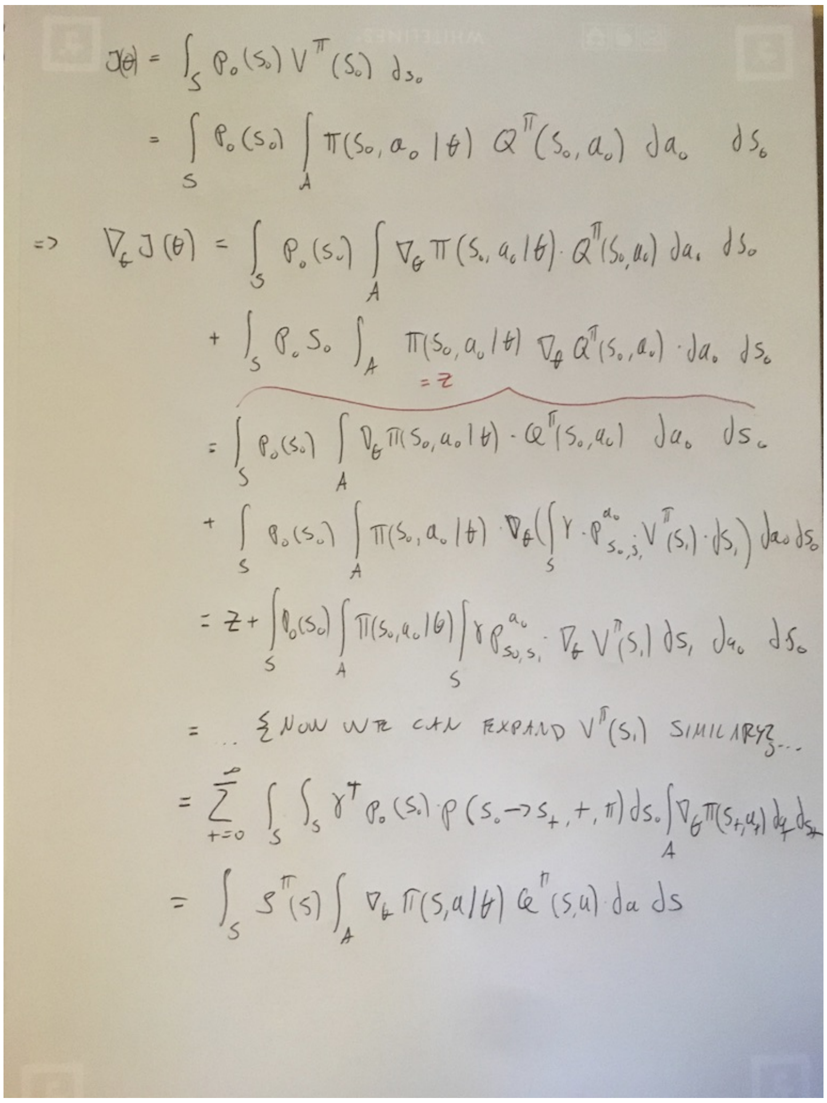
$$\begin{aligned}
\mathbf{w}^*_{BE} &= \arg\min_{\mathbf{w}} \big(B_\pi V - V\big)^\top D\big(B_\pi V - V\big) \\
&= \arg\min_{\mathbf{w}} \big(R_\pi + \gamma P_\pi \phi \mathbf{w} - \phi\mathbf{w}\big)^\top D\big(R_\pi + \gamma P_\pi \phi \mathbf{w} - \phi\mathbf{w}\big) \\
&= \arg\min_{\mathbf{w}} \big(R_\pi + (\gamma P_\pi \phi - \phi)\mathbf{w}\big)^\top D\big(R_\pi + (\gamma P_\pi \phi - \phi)\mathbf{w}\big) \\
&= \arg\min_{\mathbf{w}} \Big((R_\pi I)^\top + \mathbf{w}^\top(\gamma P_\pi\phi - \phi)^\top\Big) D\big(R_\pi + (\gamma P_\pi \phi - \phi)\mathbf{w}\big) \\
&= \arg\min_{\mathbf{w}} (R_\pi I)^\top D R_\pi + (R_\pi I)^\top D(\gamma P_\pi\phi - \phi)\mathbf{w} + \mathbf{w}^\top(\gamma P_\pi \phi - \phi)^\top D R_\pi \\
&\qquad + \mathbf{w}^\top(\gamma P_\pi\phi - \phi)^\top D(\gamma P_\pi \phi - \phi)\mathbf{w}) \\
&= \frac{\partial}{\partial \mathbf{w}}\{\text{expression above}\}\Big|_{=0} \\
&= (R_\pi I)^\top D(\gamma P_\pi\phi - \phi) + (\gamma P_\pi\phi - \phi)^\top D R_\pi + \mathbf{w}^\top\Big((\gamma P_\pi\phi - \phi)^\top D(\gamma P_\pi\phi - \phi)\Big) \\
&\Rightarrow \mathbf{w}^*_{BE} = -\Big((\gamma P_\pi\phi - \phi)^\top D(\gamma P_\pi\phi - \phi)\Big)^{-1}\Big((R_\pi I)^\top D(\gamma P_\pi\phi - \phi)\Big) \\
&= \Big((\phi - \gamma P_\pi\phi)^\top D(\phi - \gamma P_\pi\phi)\Big)^{-1}\Big((R_\pi I)^\top D(\phi - \gamma P_\pi\phi)\Big) \\
&= \Big((\phi - \gamma P_\pi\phi)^\top D(\phi - \gamma P_\pi\phi)\Big)^{-1}\Big((\phi - \gamma P_\pi\phi)^\top D^\top R_\pi\Big) \\
&= \Big((\phi - \gamma P_\pi\phi)^\top D(\phi - \gamma P_\pi\phi)\Big)^{-1}\Big((\phi - \gamma P_\pi\phi)^\top D R_\pi\Big).
\end{aligned}$$

Note that $D^\top = D$ since D is symmetric.

HW – Mar 6

- Write Proof (with precise notation) of the Policy Gradient Theorem
- Derive the score function for softmax policy (for finite set of actions)
- Derive the score function for gaussian policy (for continuous actions)
- Write code for the REINFORCE Algoithm (Monte-Carlo Policy Gradient Algorithm, i.e., no Critic)
- Optional: Write code for Actor-Critic Policy Gradient Algorithms (with TD(0) and with Eligiblity-Traces-based TD(Lambda))
- Write Proof (with proper notation) of the Compatible Function Approximation Theorem
- Optional: Write code for Natural Policy Gradient Algorithm based on Compatible Linear Function Appeoximation for Critic

PGT



$$J(\theta) = \int_S \rho_0(s_0) V^\pi(s_0)\, ds_0$$

$$= \int_S \rho_0(s_0) \int_A \pi(s_0, a_0 | \theta)\, Q^\pi(s_0, a_0)\, da_0 \; ds_0$$

$$\Rightarrow \quad \nabla_\theta J(\theta) = \int_S \rho_0(s_0) \int_A \nabla_\theta \pi(s_0, a_0 | \theta) \cdot Q^\pi(s_0, a_0)\, da_0 \; ds_0$$

$$+ \int_S \rho_0 s_0 \int_A \pi(s_0, a_0 | \theta)\, \nabla_\theta Q^\pi(s_0, a_0) \cdot da_0 \; ds_0$$

$$\overbrace{= \int_S \rho_0(s_0) \int_A \nabla_\theta \pi(s_0, a_0 | \theta) \cdot Q^\pi(s_0, a_0)\, da_0 \; ds_0}^{=Z}$$

$$+ \int_S \rho_0(s_0) \int_A \pi(s_0, a_0 | \theta) \cdot \nabla_\theta \left(\int_S \gamma \cdot \rho^{a_0}_{s_0, s_1} V^\pi(s_1) \cdot ds_1 \right) da_0\, ds_0$$

$$= Z + \int_S \rho_0(s_0) \int_A \pi(s_0, a_0 | \theta) \int_S \gamma\, \rho^{a_0}_{s_0, s_1} \cdot \nabla_\theta V^\pi(s_1)\, ds_1 \; da_0 \; ds_0$$

= … {NOW WE CAN EXPAND $V^\pi(s_1)$ SIMILARY} …

$$= \sum_{t=0}^{\infty} \int_S \int_S \gamma^t \rho_0(s_0) \cdot p(s_0 \to s_t, t, \pi)\, ds_0 \int_A \nabla_\theta \pi(s_t, a_t)\, da_t\, ds_t$$

$$= \int_S \rho^\pi(s) \int_A \nabla_\theta \pi(s, a | \theta)\, Q^\pi(s, a) \cdot da\, ds$$

Softmax Score

We have the softmax policy

$$\pi(s,a|\theta) = \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a)\Big)^{-1} \exp(\theta^\top \phi(s,a)) \;\forall\; a \in A(s) \;\forall\; s \in S.$$

The score function is

$$\begin{aligned}
\nabla_\theta \log \pi(s,a|\theta) &= \nabla_\theta \log \left(\Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big)^{-1} \exp(\theta^\top \phi(s,a) \right) \\
&= \nabla_\theta \left(\theta^\top \phi(s,a) - \log \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big) \right) \\
&= \phi(s,a) - \nabla_\theta \log \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big) \\
&= \phi(s,a) - \nabla_\theta \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big) \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big)^{-1} \\
&= \phi(s,a) - \sum_{a\in A(s)} \phi(s,a) \exp(\theta^\top \phi(s,a)) \Big(\sum_{a\in A(s)} \exp(\theta^\top \phi(s,a) \Big)^{-1} \\
&= \phi(s,a) - \mathbb{E}[\phi(s,a)].
\end{aligned}$$

Gaussian Score

Likewise we have[4]

$$\begin{aligned}
\nabla_\theta \log \pi(s,a|\theta) &= \nabla_\theta \log \left(\frac{1}{\sqrt{2\boldsymbol{\pi}\sigma^2}} \exp\big(- \frac{(a-\theta^\top \phi(s))^2}{2\sigma^2} \big) \right) \\
&= \nabla_\theta \log(\frac{1}{\sqrt{2\boldsymbol{\pi}\sigma^2}}) + \nabla_\theta(-\frac{(a-\theta^\top \phi(s))^2}{2\sigma^2}) \\
&= \frac{a-\theta^\top \phi(s)}{\sigma^2} \phi(s)
\end{aligned}$$

[4] $\boldsymbol{\pi}$ is the numeric, i.e. 3.14..

Project

This short project revolves around investigating pricing methods for American put options. This problem can be expressed as an MDP where the state is the time to maturity, price of the underlying asset and also the strike price (although the latter is constant). Thus the value function can be expressed as

$$V(s,t,k) = \max\{k - s, \gamma \mathbb{E}[V(s', t', k)]\}.$$

The goal is to implement LSPI in order to achieve a price that is similar to that of the Longstaff Schwartz (LSM) and the binomial pricing model (BPM). To investigate this, we use a GBM with the following parameters

σ	$\mu = r_f$	Maturity	Δ	S	K
0.25	0.005	5	1/40	100	110

Table 1: Parameters for GBM

First we start of by comparing LSM and BPM. Figure 2 shows the price as a function of the underlying volatility. This simply shows that both simple models seem to work well.

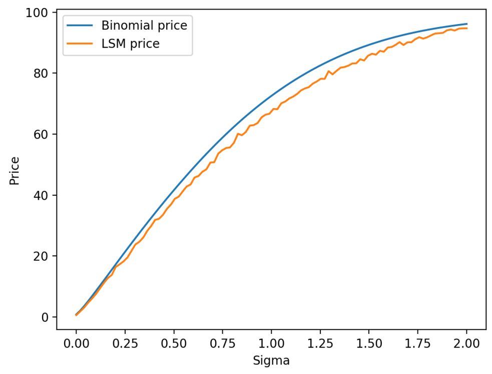


Figure 2: Price as a function of σ.

Now, implementing the LSPI-algorithm to price American put option gives figure 3. As we see the algorithm does not converge. Instead it revolves around the closed form solution from the binomial price, with a lot of variance. In order to correct this, different batch sizes have been tried, but they all give similar results.

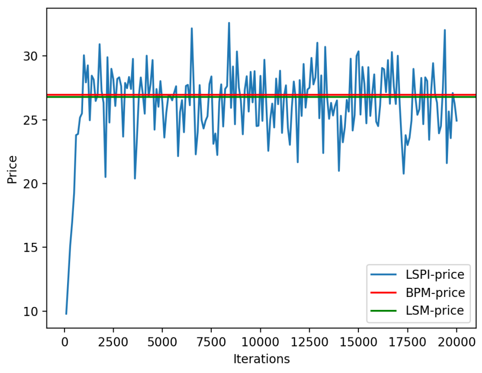


Figure 3: Price from iterations on LSPI-algorithm