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)

The first three are pretty much done (at least for MDP_A)

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