# Markov Processes

## Markov Process

A Markov Process (MP), or Markov Chain, is a directional node-edge graph which satisfies the Markov property. The Markov property requires that the probability of reaching any successor state depends ONLY on the current state (i.e. the states visited prior to the current state have no impact on future outcomes). Each node in the MP is a state , and each edge is some state transition that occurs with probability . MPs have a start state, a terminal state, and any number of intermediate states. Each state transition has a unique probability, and the sum of probabilities of all state transitions from any given state must equal 1. State transitions can be self-targeting.

Therefore, an MP sets forth the fundamental framework that we can use to establish a state navigation task with probabilistic state transitions. However, in an MP there is no maximizing objective per se (other than an objective related to network traversal, e.g. reaching the terminal state in the fewest number of steps).

## Markov Reward Process

An MRP has all of the properties of an MP, with two additions: First, we introduce the concept of a reward function which gives rewards to our agent for arriving at a particular state:

We also introduce a discount factor which reflects the time-value of the reward, i.e. a reward realized now is more valuable than an equivalent reward realized later in time.

In this formulation, the reward system now provides an objective function which can be maximized, that being the present value of all rewards accrued over a given simulation rollout.

## Markov Decision Process

A Markov decision process is a further extension of the MRP, in which actions are added to each given state. In other words, each state has a corresponding set of actions that can be taken from that state, and each state-action pair has a set of transition probabilities associated with a set of successor states.

NOTE: An MDP subject to a given policy (i.e. a set of prescribed actions for any possible state) simply reduces to an MRP!

## Value Function

Generally, the value function is meant to capture the expected value of the total discounted reward starting from any given state. The solution to the value function is obtained by decomposing the value function into the immediate reward, and the sum of all future rewards starting from the successor state. We note that the second part is in fact the value function for the successor state, and so solving this recurrence is the foundation of the Bellman equation.

In the case of an MRP, the actions are all prescribed, and so at any point the value function can be computed analytically by solving a system of matrix equations. In the case of an MDP, one must sum over all possible actions, and so the value function must be computed using iterative methods.

# Solution Techniques

## Policy Iteration

Policy iteration conceptually consists of two steps: First, a random policy is selected and the value function is evaluated for that policy (Policy Evaluation). Then, the policy is improved based on the result of that rollout. This iteration technique eventually converges to the optimal policy, that is, the policy that maximizes the value function.

**Policy Evaluation:** The value function for a given policy is evaluated at each state using the standard expression for the value function:

Note that the above is simply the expected reward in a given state, plus the discounted probability-weighted reward of future states, as measured by the value function computed at those successor states. It follows that solving this value function expression is done by recurrence.

**Policy Improvement:** Once the value function has been evaluated for our initial policy, we perform improvement by maximizing reward over our possible actions at every state:

Inspection of the above expression allows us to understand what we are conceptually doing with policy iteration. In our rollout, the policy tells us how to act at every state to reach some successor state. In the improvement step, we essentially “revisit” our states, and improve the policy by choosing the action that would maximize our reward out of all the possible actions we could take. In this manner, we correct the “sub-optimal” branches of our initial policy such that we **converge to a result where every action at every state maximizes the expected future reward as represented by the value function, which in turn is the optimal policy.**

## Value Iteration

Value iteration is similar to the policy iteration algorithm described above, except the algorithm explicitly maximizes the value function rather than optimizing the policy that produces the maximum value function. Accordingly, the algorithm is very similar to Policy Iteration, but the value function is what is being maximized at each of the states, i.e. the value function for a given state is selected as the maximum value function that can be obtained from that state. By substituting the max in this expression for an argmax, the optimal policy can also be obtained.

## Bellman Equation

The Hamilton-Jacobi-Bellman equation is a PDE that is used to analytically solve an optimal control problem for the value function corresponding to the optimal policy for that problem. For a standard optimal control problem with stage cost and terminal cost and dynamics on a time interval we have

Solving the above PDE yields the Bellman value function, which is an optimal value function derived from a corresponding optimal policy.

## Practice Problem: Midterm Q3

In this question we are asked to solve a continuous state/action MDP analytically. In this problem, our state transition outcomes are normally distributed upon taking some action . The cost associated with this action is given as . We want to minimize the infinite-horizon discounted sum of costs, and derive an expression for the optimal action in any state which will guarantee this.

From the above, we see that our cost is a monotonic function of our successor state. However, because we don’t know what the sign of will be at any given time step, a brute-force optimization does not work. Instead, we must reason in expectation about the behavior of and condition our formulation of the optimal action based on this result.

Generally, we know the optimal value function is a maximization over possible actions on our expected discounted reward. For some arbitrary discount factor , we write

However, since we are looking at the myopic case, we impose the simplifying assumption of a zero discount rate so we have

Generally, the expectation of a normally distributed variable is

In the above, the successor state is normally distributed, and we seek to calculate the expectation of the function , so we have

Now, we complete the square as , add and subtract this into the numerator, and note that the first term of this expression cancels out the now-orphaned so we have

Which by rearrangement reveals that we can resolve the integral term into the expectation of subject to a new Gaussian distribution on the variable of mean and variance , which is simply 1:

Now that we have simplified the expression, we revisit our original optimization problem:

Now that we have already formed our expectation on the stage cost, we can treat this as a direct optimization problem, i.e. we seek an action such that

We can check that this extremum is a maximum by seeing that the second derivative is negative everywhere. Since we have framed the cost as a negative reward, this means this action will minimize the cost.

# Absolute / Relative Risk Premia

## Preliminaries

Naively speaking, the decision of whether or not to invest in a particular game of chance is a simple cost-benefit calculus between the cost of participating, and the expected payoff of playing the game. The expected value here is the first moment of some probability distribution reflecting the probability of outcomes, weighted by the reward from each outcome.

However, in practice, we must also account for risk in this assessment as well. Generally, there are two factors that we consider in order to account for risk.

* The first is the inherent variability in the game itself, which is reflected by the variance of the distribution of outcomes we mentioned earlier.
* The second is the risk-aversion of the individual playing the game. Different actors will interpret risk differently.

Generally, we combine these two to form a gross-up on the economics of the game which we refer to as a **risk-premium.** The risk-premium is a dollarized representation of the game’s risk. Risk-premia of various kinds will either be reflected in the amounts of the rewards themselves (e.g. risky games will fold in higher premiums into their rewards), or in the actor’s behavior (e.g. a strongly risk-averse actor will pay relatively less to participate).

One way of accounting for the risk premium is to define a **certainty-equivalent value**, which is an adjusted measure of value which represents the extent to which risk scales down the expectation of value from the game relative to its statistical mean. This also lends to another definition of risk-premium as the difference between the certainty-equivalent value and the statistical mean:

Intuitively this makes sense: as drops, the risk-premium grows, i.e. the actor is willing to pay less to participate in response to the perceived risk of the game growing.

## Absolute and Relative Risk Aversion

The utility function is a concave function which describes total utility as a function of the amount of a particular good that is consumed. Generally speaking, the extent of concavity of the curve tells us the extent of risk-aversion (more concavity = more risk-averse). A linear utility function implies risk-neutrality (but more specifically means marginal utility is constant).

The absolute risk aversion measurement is related exclusively to the shape of the utility function, and is the ratio between the extent of concavity of the utility function and the slope of the function, given by the second and first derivatives respectively:

Where the risk premium is given as .

Comparatively, relative risk-aversion also takes into account the amount of consumption:

Where the risk premium is given as .

From the above, we see that in both cases the risk premium is the product of the intrinsic variability of the outcome distribution, as well as the mean risk-aversion of the actor (whether defined on an absolute or relative basis).

## Constant Absolute Risk Aversion (CARA)

In the CARA model, we define

Where the absolute risk aversion is a constant, hence the name. This leads to a risk-premium of

Which we can cast as an optimization problem which is focused on maximizing (note that this is assuming that the variance is a function of the mean)

## Constant Relative Risk Aversion (CCRA)

In the CCRA model, we define

Where now the relative risk aversion is constant, . This yields a risk-premium of

Which we can cast as an optimization problem which is focused on maximizing (note that this is assuming that the variance is a function of the mean).

# Applications: Merton’s Portfolio Optimization Problem

## Problem Statement

Fundamentally, Merton’s portfolio optimization problem deals with investing some amount of initial wealth into a number of risky assets and a risky asset in continuous-time. Any fractional amount of wealth can be consumed at any time (i.e. allocated to any mix of the assets). The objective is to maximize the lifetime-aggregated utility of consumption.

## MDP Formulation

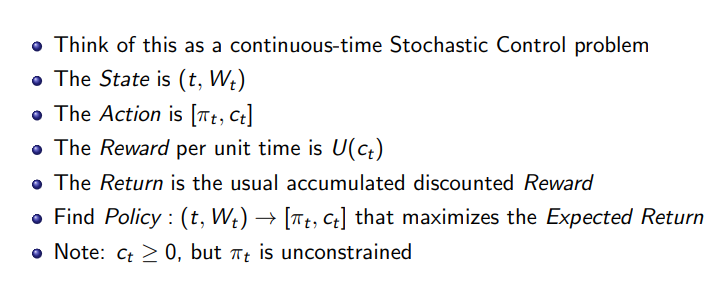
Based on the above description, it is possible for us to formulate the problem as an MDP. At a high level, we consider the state to be the amount of wealth we have at any given time, our actions as the choice of allocation of that wealth among different assets, and the reward as the return generated from our resulting portfolio (see Figure below).

Figure : Casting Merton’s Portfolio Optimization problem as an MDP.

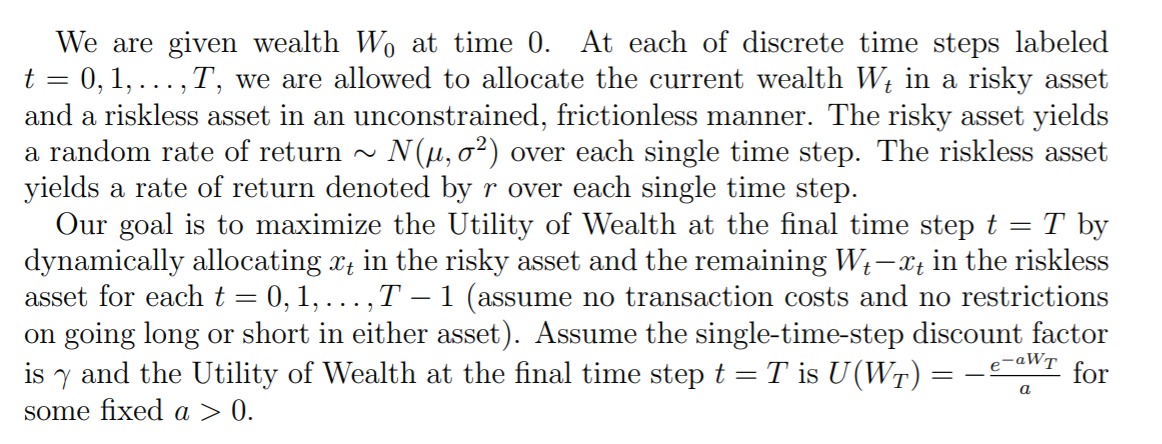
To guide our understanding, it is helpful to more explicitly write how the model would be simulated.

* We begin at some initial state with our initial amount of wealth

## Solution

The Merton portfolio problem has an analytical solution which is given in the course slides. This solution is derived from an HJB formulation of the problem. In the analytical solution, we see that the optimal allocation and expected portfolio return are both constant with time, and the fractional consumption depends only on time. The key insight here is that **the optimal portfolio allocation, subject to this formulation, does not depend on the total amount of wealth at any given time at all.** Although we know that risk-aversion plays a factor, this property manifests itself directly in our formulation based on the amount allocated into the riskless asset, rather than directly by saving wealth (in other words, we assume that liquidity does not play a role).

## Midterm Practice Problem: Optimal Portfolio Allocation



### Part 1

First, we formulate the above problem as an MDP:

* Our state-space is spanned by , i.e. the point in time and the amount of wealth we have at that time.
* Our action-space is spanned by , i.e. the amount of our wealth at any point in time we choose to allocate into the risky asset
* Our reward will be zero at all points from , and the reward at time T will be the utility derived from our wealth at that time, i.e. .

### Part 2

We start from the fundamental definition of the value function, which is simply the expected value of the total discounted reward we accrue from following actions prescribed by a policy starting from a given state:

From the above, we know that we only realize rewards at the final time step, so we have

Where .

To find the optimal value function, we simply apply a policy which maximizes the expected discounted reward that is used to calculate the value function, i.e.

We know the generic form of the Bellman Optimality Equation is given as

# Applications: Derivative Pricing / Hedging

## Problem Statement

Fundamentally, we seek to maximize our risk-adjusted return by taking decisions at each time step to trade on hedges for derivatives. Although the financial instruments are different, this problem is analogous in nature and solution to the Merton portfolio optimization problem in that assets are being apportioned to different investment vehicles for the purpose of maximizing a portfolio of returns across hedge positions.

## MDP Formulation

Analogously to the Merton problem, we can cast this as an MDP. Our state includes all relevant information about prices and positions over time, and the actions taken at each step are units of hedges traded. The reward is then the return resulting from the portfolio of investments. We also assume a complete market, i.e. the payoffs of all derivatives can be replicated, as well as other assumptions which are detailed in the slides.

## Solution

The traditional solution to this problem is given by the Black-Scholes formulas (see e.g. <https://www.investopedia.com/terms/b/blackscholes.asp>). Deep RL approaches have also been attempted for this type of problem. At a high level, each time step involves performing a set of transactions for hedge instruments and recalculating a P&L which accounts for returns from hedge positions and transaction costs.

# Applications: Optimal Market-Making

## Background

Fundamentally, development of an algorithm for optimal market-making involves developing a model of the marketplace itself. Here, we consider buyers and sellers who express intent to buy or sell through Limit Orders (LO), for some amount of shares N at a price P. Buyers provide *bids* of the value they are willing to pay, while sellers provide *asks* on some amount they are willing to sell for.

## Algorithm

# Model-Free: Monte Carlo methods

## Overview

Recall that in the model-based methods above, one of the things we needed to compute the optimal solution by solving the Bellman equation was the transition probabilities between every state pair, given an action. In reality, this is not always the case – we may not know the dynamics of our system, in which case the above approaches do not work.

When we do not know the dynamics, one approach to determine an optimal policy is to learn the dynamics of the system and compute the relevant missing parameters from our observed data. This is what Monte-Carlo methods are fundamentally premised on.

## Algorithm

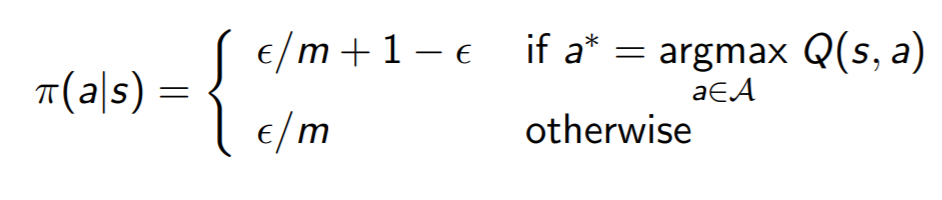
Effectively, the two steps of Monte-Carlo are to 1) roll-out some exploration policy to collect data about the game, and 2) to retrace your steps and compute cumulative rewards stemming from each path, which serve as the basis for computing the value functions V and Q for each state. These two steps can be broadly referred to as the “forward pass” and “backwards pass”, respectively.

## Epsilon-Greedy Policy Improvement

Although the Monte-Carlo method allows us to derive information about our state-space and value functions from rolling out episodes using a policy, it does not directly allow us to improve that policy in any way. To do this, we can expand Monte-Carlo to make use of Epsilon-Greedy policy improvement. To motivate this technique, we will prove the following:

*Theorem: For any -greedy policy , the -greedy policy with respect to is an improvement on the policy with respect to the value function, i.e.*

To prove the above result, we first present the generic form of our greedy policy, which will choose a greedy action with probability and choose an action at random with probability , presented in mathematical notation below:



From the above, we can begin our proof by beginning with the definition of the state-action value function applied to a given state by the improved policy , which is obtained by marginalizing the action out of the stochastic policy and summing across all possible actions:

Now, since we know the general form of the epsilon-greedy policy from above, we can split out the possible action outcomes according to the probabilities of each. If we have actions, we choose the greedy action (the one which maximizes the state-action value function) with probability and each of the other actions with probability . So we have

In the second term, we note that we are taking the greedy action by maximizing the state-action value function over all possible actions we can take. For any maximizing action like this, we know that it will be greater than or equal to the expected reward (i.e. the probability-weighted sum of actions), so we can write

Cancelling terms in the above, we can then simplify to

Which allows us to conclude that  **i.e. that the epsilon-greedy policy improvement step necessarily produces an improved policy (i.e. a policy whose value function is equal to or greater than the value function of the original policy).**

## GLIE (Greedy in the Limit with Infinite Exploration) Policy Improvement

The GLIE approach to policy improvement is similar to the epsilon-greedy technique discussed above, but works slightly differently in that optimality is driven by the end-behavior of visiting all state-action pairs infinitely many times. If we denote as the number of times we have visited a certain state-action pair, then for each policy rollout we increment our counts and state-action value functions as

And perform policy improvement as

From the above, we see that as the number of visitations becomes very large (k), the value of begins to be driven down to zero. Revisiting the original definition of our original greedy algorithm, we see that this value corresponds to the probability of taking a non-greedy action (choosing randomly among all possible actions). This mathematical result therefore implies that as the number of visitations becomes infinitely large, the policy converges to one that exclusively chooses the greedy action at any given state.

# Policy Gradient ALgorithms

## Background

In the “tabular” methods of reinforcement learning (e.g. Value Iteration), we optimize a policy for our problem by selecting actions at each state which maximize expected forward-looking reward. This, in turn, requires iterating through each state and at each state, assessing each possible action that can be taken from that state in a recursive fashion to optimize the solution. When our state-space or action-space is continuous or large, however, this approach rapidly becomes computationally intractable.

The motivation behind Policy Gradient is to perform optimization in scenarios with a large action-space, without having to directly perform a maximization operation over all possible actions. Instead, we treat the action space as continuous, and “step” iteratively towards the optimal solution using gradient descent. By using this mode of optimization, we have flexibility on the computational load of the algorithm, and can approach a result that is very very close to the “true” optimum derived from traditional policy iteration, but at a fraction of the computational cost.

## Policy Gradient Theorem

(see <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#policy-gradient-theorem>)

In order to prove the policy gradient theorem, we must first motivate the above problem. Since we plan to use gradient descent, we need to a) parametrize the function we wish to optimize, and b) select an appropriate loss function to measure convergence.

For part a), since we are trying to optimize our policy, we must simply cast our policy as a parameterized stochastic policy with respect to some arbitrary parameter :

For part b) we will express our objective function simply as the total expected returns, which once again must be parameterized according to :

With the above, we can formulate our “weight” update for some learning rate as

By expanding out the above expectations, we can arrive at the following more precise definition for the reward function:

Where is referred to as the *discounted-aggregate state-visitation measure,* and effectively represents a conditioned probability distribution over possible states (based on the probability of moving from some start state to any target state in time steps, subject to our policy ).

Since we note that the second integral in the reward function resembles the value function, we begin by taking the gradient of the value function

Which by product rule gives

In the second term we can expand the state-action value function (ignoring discounting):

In the above, we observe that we have a recursive expression. If we continue to unroll recursions of this expression, we can simplify to the general form

Where represents the probability of going from state to state in time steps subject to policy . We see that this term closely matches the form of our state-visitation measure in the reward function we previously defined. From visual inspection of the two expressions, we then see that

If we let denote the number of time steps spent, on average, in a given state in a single episode, we have

Which is algebraically equivalent to

Noting that is the on-policy state distribution , we then see that

Which is the final result of the policy gradient theorem. The key implication of this finding is that, **when computing the gradient of our loss function, we do NOT need to evaluate the gradient of the state distribution with respect to the parameter** . Practically, this derivative would indicate the effect of a policy update on the state distribution, which is difficult to estimate in an unknown environment.

## Softmax Policy

## Gaussian Policy