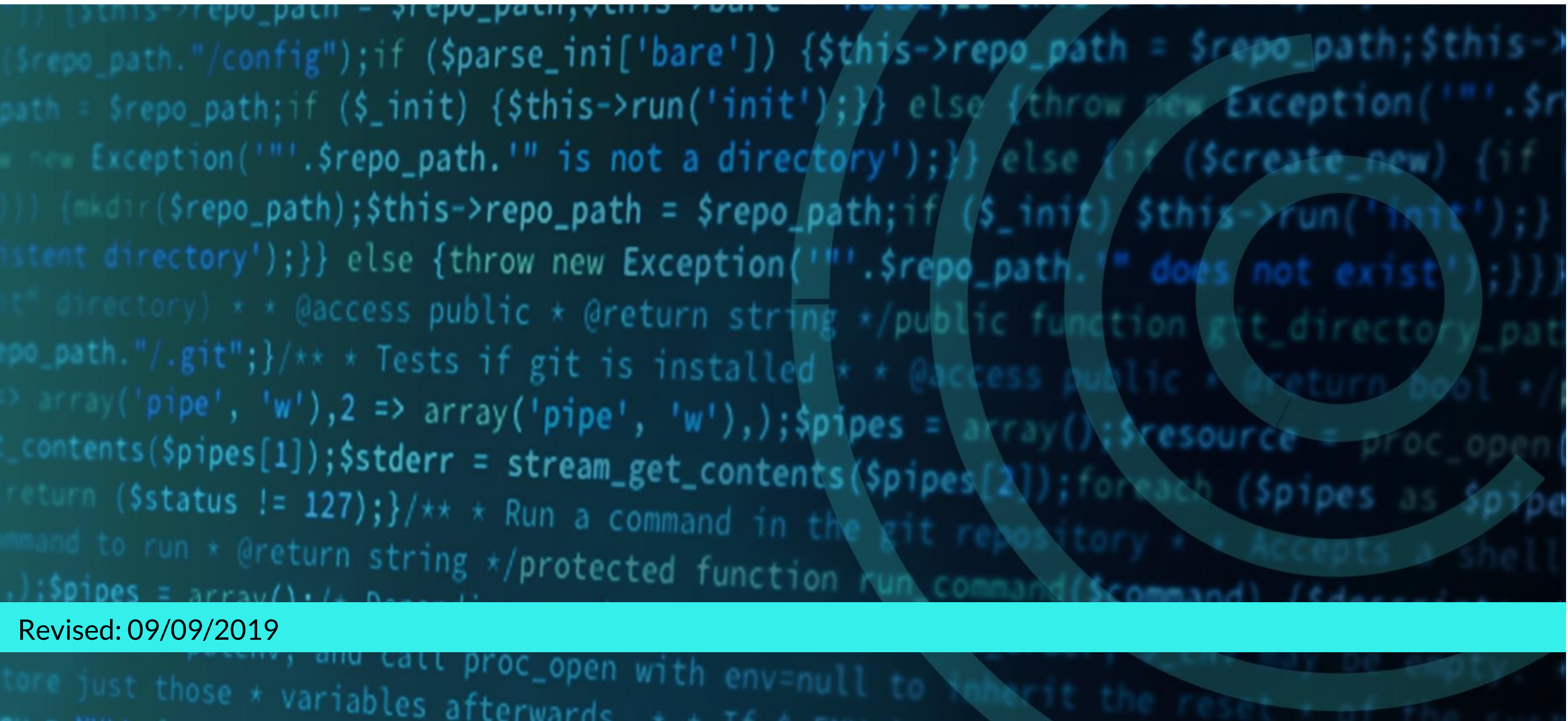
WORLDQUANT UNIVERSITY

Compiled Content Module 2

MScFE 640

Portfolio Theory and Asset Pricing



Revised: 09/09/2019

Table of Contents

Module 2: Introduction to Stochastic Dynamic Control 3

Unit 1: Dynamic Programming and Optimization Problems.................................. 4

Unit 2: Markov Decision Processes and the Markov Property................................9

Unit 3: The Bellman Equation..14

Unit 4: Discrete Versus Continuous Time and Diffusion Processes...................18

Bibliography 23

Collaborative Review Task 24

Module 2: Introduction to Stochastic Dynamic Control

This module begins by introducing dynamic programming and optimization problems. The module continues by introducing Markov decision processes and the Markov property and by explaining the Bellman equations. The module ends with a discussion on discrete time versus continuous time processes and diffusion processes.

Unit 1: Dynamic Programming and Optimization Problems

One of the best-known books on dynamic programming was written by Dmitri Bertsekas: *Dynamic Programming and Optimal Control, 4th Edition* (2017). It can be broadly applied to many fields. Specifically for financial portfolio optimization, Gérard Cornuéjols, Javier Peña, and Reha Tütüncü give great examples in their text, *Optimization Methods in Finance, 2nd Edition* (2018). The material in this chapter is largely adapted from these books.

Dynamic is a programming technique for solving problems. It is used for optimization problems where you can find a solution with an optimal value. By **optimal**, we mean a minimization or a maximization. For example, if we wanted to find the least risky portfolio for a given return, we would perform a portfolio minimization. Conversely, if we wanted to find the highest expected return of a portfolio given a fixed amount of risk we are willing to accept, we would perform a portfolio maximization.

Why is it called programming?

The word programming is not used in the sense of computer programming but rather in the sense of scheduling by *filling in a table*. For example, the term television programming means building a table of channels and showtimes and then filling in each combination of what each channel broadcasts at each particular time. Another term for this would be television scheduling.

Similarly, athletic programming in a gym refers to the classes that are offered at specific times of day on specific days of the week. The result is a table that would have days of the week as columns and times of days as rows. The athletic programmer would schedule events by completing such a table.

Why is it called dynamic?

Unlike our television or athletic programs, the dynamic programs Bellman created were not meant to be filled in all at once but rather over time. Most likely, a television station’s programming could likely be known a month in advance so that cable companies could publish their schedule – online or as a printed copy – ahead of time. A dynamic program, however, is determined recursively. In order to determine the system’s value at some time, the system’s value should be known at a point adjacent in time. For example, pricing an option one period prior to expiration is an example of a dynamic program, because we know the option’s value at the adjacent time – i.e. one period immediately afterwards. The binomial tree example you studied in discrete time stochastic processes is an example of this sort.

The name dynamic programming, while general, gives a good sense of the methodology. Note that dynamic programming is focused on finding *a* solution rather than finding *all possible* solutions. Nevertheless, the solution that it finds is optimal, meaning that there is no solution that is better. There may be solutions that are equally optimal, though, because there can be more than one optimal solution to a problem.

Different problems

Optimal problems require an optimization, which means that there is some value that is being minimized or maximized. Let's consider two examples. An example of a minimization problem would occur if we were trying to minimize transaction costs associated with rebalancing an investment portfolio. An example of a maximization problem would occur if we were trying to maximize utility as a function of wealth associated with an investment portfolio.

Note, however, that a maximization problem could be turned into a minimization problem by taking the negative of the values associated. Thus, even a utility maximization could be thought of as minimizing the negative of the utility function. (This idea is explicitly mentioned here because some software have optimizers that tend to work better on minimization problems rather than maximization problems.)

Let's consider the parts of a dynamic program, beginning with the idea of a sequential system.

Sequential system

We can think of a sequential system as having four elements:

1. Stages
2. States
3. Decisions
4. Law of motion.

Let's examine each one in turn:

1. **Stage**: a point in time at which a decision is made. We will work in discrete time. (Note that the differences between discrete time and continuous time will be discussed later).

Discrete time is much easier to work with than continuous time because when we implement a solution in software it is almost always in discrete time.

The notation used to represent times is typically $t = 0,\ 1, 2,\ 3, \dots T$. Typically, it is used as a subscript to one of the other system elements. In other words, we will have states represented as $s_t, s_{t+1}, s_{t+2}, \dots$, and decisions represented as $x_t, x_{t+1}, x_{t+2}, \dots$. Note that sometimes the starting time may be given as 1, but the final time is consistently represented as T.

2 **State**: for a given time, the information contained in the system in order to make a decision.
The notation used to represent states is typically written as $s_t, s_{t+1}, s_{t+2}, \dots, s_T$.

3 **Decision**: at each state, an action is undertaken that affects the state of the system. Decisions change states and influence decisions that may precede or follow them.

The notation used to represent decisions is typically $x_t, x_{t+1}, x_{t+2}, \dots, x_T$.

4 **Law of motion**: the means by which the state of the system evolves.The notation used to represent laws of motion at a specified state is

$$s_{t+1} = f_t(s_t, x_t) \;\; for\ t\ in\ \{0, 1, \dots, T\}.$$

A sequential decision problem can be written as:

$$\sum_{t=0}^{T} g_t(s_t, x_t) + \; g_{T+1}(s_{T+1}).$$

The function g is a cost or utility of being in the state s_t and making an action or decision x_t. Note that the first term is in the summation but the second term is not. The second term is a function that may relate to benefit or utility – i.e. something positive – or, alternatively, a cost or penalty – i.e. something negative. Using the notation we defined previously, the sequential problem is to find the decision that maximizes our utility function (or minimizes our cost function) with respect to x.

Let's consider an example where we have to maximize the utility of our consumption. Assume that at time 0 you have an initial portfolio value P_0. In t-years time, you have a choice to consume C_t

dollars now and invest the rest of your portfolio in risk-free bonds. The decision you make is how much you will consume: anywhere from nothing to the entire amount. If you consume C, there is an associated utility with that, $U(C)$. Postponing your consuming, you can maintain the funds invested in the risk-free rate to earn an interest of $(1+r)$ each year.

This is an example of a dynamic programming problem.

Applications

Let's apply the framework listed above to this example.

The stages can be thought of as a number of years, starting with an initial time of 0 – when you first invest the funds and let it grow for a year. At time = 1, you have earned some interest and can now make your first decision: how much will I consume? Whatever you spend is reduced from the portfolio, and whatever is left will continue to earn interest at a rate of r, so that at time = 2 you will have $(1+r)$ times the amount in the portfolio. At that time, you will again decide how much to consume, which then determines how much gets reinvested that year. This continues until time t_{T-1}. The state at each stage is the **portfolio balance**. It encapsulates everything you need to know about the system. Note that it is not important to know the portfolio value two or more periods ago, as the current portfolio value is only dependent on the value one period ago and the current consumption decision. Therefore, the interest that is earned depends only upon the amount that is in the account at that time.

The decision at a stage t is the amount we choose to consume, which must be in the range of 0 to P_t

The law of motion is the wealth at a stage, represented by the portion of the portfolio from the previous stage that was not spent, accrued with interest. In other words, it is *principal plus interest*.

Now that we have qualitatively described the system, let's put the framework into formulas with the notation we discussed earlier.

Stages:	$t = t_0, t_1, t_2, \ \dots t_T$.
States:	$P_t, P_{t+1}, P_{t+2}, \dots, P_T$
Decisions:	$S_t\colon\ 0 \le \ S_{t+1} \le \ P_T$
Law of motion:	$P_{t+1} = \ (P_t - S_t)(1+r), t = t_0, t_1, t_2, \ \dots t_T$.

The objective is to maximize the total utility of consumption.

$$\max_{S_0, \dots S_T} \left\{ \sum_{t=0}^{T} U(S_t) \right\}.$$

Let's try the same approach that we used to find a strategy to optimize the matches game – working backwards.

Let's use P_t to represent the portfolio available at the beginning of year t. Let's also use $U(S_t)$ to represent the total utility of consumption from year t to year T. The value function should satisfy the following backwards recursion for $t = t_0, t_1, t_2, \dots t_T$.

Why do we stop at t_{T-1}? We stop because that is the last stage at which we make a decision. Our decision at t_{T-1} will affect how much our portfolio will have at time t_T. That is the final stage. The reason is that we'll assume that we spend everything that remains at time t_T, so that $P_{T+1} = 0$. We've spent it all!

Unit 2: Markov Decision Processes and the Markov Property

A Markov Process is a random process with two main characteristics:

1. It is stochastic
2. It is memory-less.

Let's elaborate on each of these two terms and then provide some context.

On being stochastic

A process that is stochastic cannot be predicted with accuracy. It is not deterministic. Rather, we need a set of probabilities to describe how to move from one state to another. So how do these probabilities behave? In order for a stochastic process to be classified as a Markov Process, it must satisfy the Markov Property condition – i.e. for every time t, we have:

$$P\big(x(t_n) \leq x_n \,|x(t_{n-1}), x(t_{n-2}), \dots, x(t_1)\big) = P\big(x(t_n) \leq x_n \,|x(t_{n-1})\big).$$

In words, this says that additional values of the past *do not* improve the ability to predict the future. To put it differently, all the information contained in the system is present in the most recent value of the system.

Let's consider a random walk. Suppose that at each point in time, a person flips a coin. If the toss results in heads, then she can take one step to the right. If the toss results in tails, she takes one step to the left. This continues repeatedly.

After a number of flips, suppose she is standing four steps to the right from the point where she started. What is the probability of her moving to the right? To keep it simple, let's assume the probability of stepping to the left is 50% and probability of stepping to the right is 50%. This means that after the next coin toss:

- If she flips heads, then she moves one step to the right. Thus, she has a 50% chance to be at +5.
- If she flips tails, then she moves one step to the left. Thus, she has a 50% chance to be at +3.

Let's note a few points:

1. She has a 0% probability of remaining where she is, because any outcome moves her from her current spot.
2. She also has a 0% probability of being at +2, or +6 or any other number further away from 4 because she simply takes one step at a time.
3. Thus, at every time point, she only has two choices.
4. Assigning probabilities to the locations where she could be depends entirely on where she is now and not where she had been in the past.

On being memory-less

This last point specifically makes this a Markov process. This stochastic process has a Markovian property in that it is memory-less. It doesn't matter if she came directly from 0 to +4 by flipping four heads in a row, or by flipping the coin 100 times and winding up at +4, or by any other combination such that she arrives at +4. The important point is that the only values we use to predict the next location are the current values of the state.

Written more quantitatively,

$$P\big(x(t_n) = +5 \,|x(t_{n-1}) = 4, x(t_{n-2}), \dots, x(t_1)\big) = P(x(t_n) = +5 \,|x(t_{n-1}) = 4)$$

and

$$P\big(x(t_n) = +3 \,|x(t_{n-1}) = 4, x(t_{n-2}), \dots, x(t_1)\big) = P(x(t_n) = +5 \,|x(t_{n-1}) = 4).$$

Just as you saw in the video, we can create a matrix that shows us the probability of going from one state to any other state. So, what is the defining characteristic of a Markov Process? It is that the probabilities can be determined by where we currently are and not by everywhere we have been.

Matrices

Let's look at the table below to see how we create this transition matrix – the matrix of probabilities of moving from one state to another.

		Next position						
		1	2	3	4	5	6	7
Current position	2	$\frac{1}{2}$		$\frac{1}{2}$				
	3		$\frac{1}{2}$		$\frac{1}{2}$			
	4			$\frac{1}{2}$		$\frac{1}{2}$		
	5				$\frac{1}{2}$		$\frac{1}{2}$	
	6					$\frac{1}{2}$		$\frac{1}{2}$

Each row corresponds to a set of the current positions. In reality, there should be an infinite number of rows because any integer is valid. So, what we are seeing is just a small subset of the possible positions. Each column shows what the next position is.

Now that we understand what we are seeing, we can make a few observations. The probabilities in every row sum to 1. This comes from the Law of Total Probability: stepping to the left and stepping to the right constitute a full set of possibilities, with probabilities that sum to 1.

Also, notice you get a bidiagonal matrix, that is, there is a diagonal just below the main diagonal. This lower diagonal contains ½ everywhere. Likewise, there is another diagonal just above the main diagonal that contains ½ everywhere. This comes from the symmetrical probability of flipping heads or tails. What if we were to use a biased coin? Suppose there were a 60% chance of heads and a 40% chance of tails. Do we still have a Markov process? We do indeed, but now the transition matrix would look like this:

		Next position						
		1	2	3	4	5	6	7
Current position	2	$\frac{2}{5}$		$\frac{3}{5}$				
	3		$\frac{2}{5}$		$\frac{3}{5}$			
	4			$\frac{2}{5}$		$\frac{3}{5}$		
	5				$\frac{2}{5}$		$\frac{3}{5}$	
	6					$\frac{2}{5}$		$\frac{3}{5}$

Observe also that in both of these tables the main diagonal itself is always 0. Why? In the random walk, there is a zero chance of remaining in the same position.

Where would we be in two steps of time? Well, because the process is Markovian, we can simply multiply the transition matrix twice to see. Let's return to the transition matrix of our fair coin and multiply the matrix by itself. For simplicity, let's use a square matrix that contains the levels from 2 through 6, and we'll only show the probabilities for starting at a current position of 4.

	2	3	4	5	6
4	0.25	0	0.5	0	0.25

After two flips, we have a 25% probability of being 2 steps down; a 25% probability of being 2 steps up; and a 50% probability of returning to where we started. Indeed, at any point in time, the probabilities of going from 1 period to another remain the same. In these cases, repeatedly multiplying a transition matrix by itself *m*-times gives you the probabilities of where you will be in *m*-steps.

Markovian or not?

So, are all stochastic processes Markovian? Certainly not. Suppose you are a technical analyst, and you compute 20-day moving averages as a trading signal. In order to predict the next direction, you require the previous 20 days of historical data. The last value alone is insufficient to determine your price. In that case, this model is non-Markovian. Indeed, all technical analysis relies on the underlying stock prices having non-Markovian properties. Any type of extrapolation of previous history is saying that there is information contained in the historical set that can improve the ability to forecast future levels.

Summary

How does that compare with the efficient market hypothesis? You studied the efficient market hypothesis in Financial Markets and Econometrics. If stock markets are weakly efficient, then this random walk is a stochastic process by which stock prices move. That means technical analysts are trying to make sense out of a market for which randomness prevails.

This is why the random walk is one of the most popular random processes studied in financial engineering. It describes what many perceive as the prevailing model for stock prices. It is so popular that the best-selling book about this, *A Random Walk Down Wall Street: The Time-Tested Strategy for Successful Investing* by Burton Malkiel, is now in its 12th Edition (January 2019). Let us remember, though, there is another book, *A Non-Random Walk Down Wall Street* by Andrew Lo, which has never been revised and has probably not sold as many copies. Popularity and book sales, however, do not determine whether markets are random or not. Let us look at empirical data.

According to SPIVA S&P Indices Versus Active (https://us.spindices.com/spiva/#/), over the 15-year investment horizon 92.33% of large-cap managers, 94.81% of mid-cap managers, and 95.73% of small-cap managers failed to outperform on a relative basis.

Is this why index funds are so popular? Is this why active managers tend to underperform the market? We need to remind ourselves of these issues as we travel down the road of portfolio optimization.

Unit 3: The Bellman Equation

Introduction

Dynamic programming's central equation comes from Richard Ernest Bellman, who is considered the father of dynamic programming. Recall that a dynamic program has different states, different stages, and different decisions. These can be interpreted as space, time, and actions, respectively.

For a portfolio optimization problem, the states could be whether the portfolio is up 1% on the year, 2% on the year, etc.; they could also be whether the portfolio is down 1% on the year, 2% on the year, etc. The stages would be 1 year, 2 years, and so on. The decisions would be whether to purchase, say, Stock A or Stock B, or to put the funds into a risk-free bond. Let's keep stages, states, and decisions in mind as we examine the Bellman equation.

Bellman equation

The Bellman equation can be written as follows:

$$U(s_0) \overset{\text{def}}{=} \max_{x_0, \dots x_T} \left\{ \sum_{t=0}^{T} g_t(s_t, x_t) + g_{T+1}(s_{T+1}) \right\}.$$

There is a lot going on in this equation: summations, maximizations, subscripts. Pay attention to subscripts. Their font size is smaller but understanding them means you get the gist of the formula. Why do actions have subscripts? Because different actions are performed at different times. Likewise, states have subscripts because they occur at different times. There is even a time subscript for the function that tells us how much utility we are adding, which is based on the combination of the state and the action we take.

Now, notice that there are three references to states in the Bellman equation, and they all occur at different times. On the right-hand side, the first state has a subscript that is an index variable, which varies from time zero (0) to time T. The second state on the right-hand side has a time subscript of $T + 1$, which is the state the system will be in after our decision is made. The state on the left-hand side is the value of the state at the beginning of the problem – time zero.

The right-hand side of the Bellman equation contains two terms. The first term is a function that gives the value based on the state we are in and the decision we make at that time. Sometimes in dynamic programming this is referred to as a reward, or value, function. There is an index t, and we sum over all time stages. Note that the summation term only includes the first term on the right-hand side. The second term on the right-hand side is simply the value function, evaluated at time $t+1$. It is evaluated once – at time $T+1$. However, the value that is getting maximized is the sum of *both* the utility of the action we take and the utility of the state that follows our decision. Why do we add in the last term? Our decisions at time T puts us in a state at time $T + 1$. If that decision, say in the matches game, made us win, that will then maximize our utility. Therefore, the reward or utility of the last decision is included in the overall utility function.

Recall that the function g is a cost or utility of being in the state s_t and making an action or decision x_t. Notice that utility appears on both sides of the equation, albeit in different formats. It appears on the left side as the U function because that is what we are trying to solve. It also appears on the right side as the g function because that indicates our utility is a function of the state we are in and the decision we make. In other words, we look at the utility of the prior time period based on the utility of what comes next. This is where the recursion comes from.

Therefore the Bellman equation is a recursive equation in discrete time. We make a decision – i.e. we take an action. We get some utility for it. This takes us to the next state. However, we do not want to just take any action. We want to take an action that results in us performing optimally – i.e. maximizing our utility. That is basically what the Bellman equation tells us.

Playing the match game

Suppose you were playing the match game. Does the decision you make at time t_t depend on the state? Absolutely. If your opponent had five matches on the table and chose three, there would be two matches left. Therefore, your optimal decision is to take one match. But if your opponent chose two matches, there would be three matches left, and you would have to choose two matches. That is why the utility function $g_t(s_t, x_t)$ depends on *both* the state and the decision. The effectiveness of the decisions depends entirely on what the current state is. Taking one match is a good decision only if it leaves one match. Taking one match is optimally bad if it leaves two matches. If you took one match, then your opponent would also take one match, and one match would be left on the table – which would force you to lose.

Let's run through states, states, and decisions in the game of matches.

The stage: Say we are at time $t = t_{T-4}$ (and the game ends at time t_T).

The state: There are 6 matches left.

Player A's decision: We take 1 match.

The new stage: $t = t_{T-3}$

The new state: There are 5 matches left.

What is the utility with this new state? Well, for the matches game we don't have a specific function, but the utility is optimal because Player A has a guaranteed win. If Player B takes n matches (where n could be 1, 2, or 3), Player A takes $4 - n$ (which could be 3, 2, or 1) matches.

Player B's decision: Take n matches.

The new stage: t_{T-2}.

The new state: There are $5 - n$ matches left.

Player A's decision: Take $4 - n$ matches.

The new stage: t_{T-1}.

The new state: There are $(5 - n) - (4 - n) = 1$ match left.

Player B's decision: Take 1 match (the last one).

The new stage: $t = t_T$

The new state: There are no matches left. Game over. Player A wins.

Bellman equations, matches and, utility

The game of matches is a simple example that illustrates the point of recursion and working backwards, but it does not define a specific utility function. In order to calculate the Bellman equation, we would need a utility function that would tell us, for example, how a certain portfolio balance would translate to a utility measure. (There are various ways that utility can be defined, and you are likely to have discussed these in previous course modules.) We would then be able to calculate the left-hand side of this equation. We would be able to do so for each state, which is characterized by a stage (time), state level (state), and set of decisions (e.g. an action or choice to make). The utility of a state is the maximum over all the decisions that we can make that get applied to a summation. It is a function of the states we are in.

Considering the match game again, what was the problem-solving method we used? We broke the problem into smaller sub-problems and solved each of those. What was the order in which we solved those? We worked backwards. Consider the utility maximization problem. How did we solve it? The same way. What was the order? The same way – backwards. We call such processes **tail problems**. That is, we begin at time T and work backwards towards the present time. If you ever solved for an option price by using a binomial tree, you used the same backwards approach.

Indeed, many problems in financial engineering are solved backwards because we know the value we want at expiration or maturity, and then we work backwards to find an optimal solution at the previous time step, and then the previous time step from that, and so on.

In the particular case of the utility maximization problem, we would write the general equation above as follows:

$$U_t(s_t) = \max_{x_t, \dots x_T} \left\{ \sum_{\tau=t}^{T} g_\tau(s_\tau, x_\tau) + g_{T+1}(s_{T+1}) \right\}$$

Unit 4: Discrete Versus Continuous Time and Diffusion Processes

In your previous coursework on discrete time stochastic processes and continuous time stochastic processes, you examined the difference between discrete time and continuous time. In the area of derivative pricing, if you were to solve for an option's price you could choose to work in discrete time or in continuous time. Typically, working in discrete time is easier to implement in computers. In discrete time, you set up a binomial tree (or some other finite difference grid), and at each point you calculate a value for the underlying from which, by working forwards, you could price the option at expiration. Then, by working backwards, you could price the option one time step before expiration, and then one time step before that, and so on.

In continuous time, the approach was very different: you solved for an option price by writing a stochastic differential equation for the stock process, applying Ito's Lemma, no-arbitrage assumptions, and a boundary condition to derive a closed-form solution that removed the stochastic component, and turned this into a partial differential equation with a closed-form solution. When we assumed the stock process was a log-normal Brownian motion, we wound up with the Black-Scholes model.

In the area of portfolio theory, the ideas are similar but the decisions are different: you must decide what amount to put in stock 1, what amount to put in stock 2, etc. If we want to examine decision-making using binomial trees, or on a finite difference grid, we will use discrete time. It is possible to also make decisions in continuous time but is harder in practice because of the constant rebalancing required.

Continuous time

Whether in derivative pricing or portfolio optimization, continuous time requires much more difficult mathematics. It is also not as easily implemented as a solution in discrete time. Think back to Black-Scholes. We did not solve for Black-Scholes in continuous time on the computer. What we most likely did was to solve for the Black-Scholes *formula*, which is the solution to the Black-Scholes partial differential equation. The formula is a relatively straightforward calculation. This works well for European calls and puts because there are closed-form solutions for these. It does not work for American puts, however, because there are no closed-form solutions for these. Therefore, it is easier to work in discrete time to price these American options. Likewise, we will work in discrete time here.

Discrete time

Working in discrete time, we want to know what type of random increment we have from one time step to the next. Suppose we are trading stocks. We might assume that stock returns over a very small window of time follow a Gaussian distribution, characterized by a constant, known mean and a constant, known standard deviation. This translates into the following (continuous time) diffusion equation for stock returns:

$$dS/S = mu * dt + sigma * dz,$$

where:

$dS =$ the incremental change in the stock price

$S =$ the current stock price

$mu =$ the drift term

$dt =$ an increment of time

$sigma =$ the volatility

$dz =$ a random increment from a Gaussian distribution.

This diffusion is a geometric Brownian motion, which in turn implies that log stock returns are normally distributed. However, once again, we are in continuous time. How do we navigate between continuous time and discrete time? It's easy: we simply discretize the random elements by replacing incremental changes with differences: dt becomes Δ_T and dS becomes Δ_S.

Distribution

When you look at the return's distribution, you notice that there are more outliers than the Gaussian distribution would assume. That is, in a normal distribution, "tail events" are not very likely. Here are the percentiles for example data.

Left Z score	% Data in-between	Right Z score
-1	68.269%	1
-2	95.450%	2
-3	99.730%	3

Left Z score	% Data outside	Right Z Score
-1	31.731%	1
-2	4.550%	2
-3	0.270%	3

Recall that the **z-score** is the number of standard deviations. When you calculate how much of your return data is in outside -2 and +2 z-scores you get a number more than 4.5%, indicating that there are more extreme values in the observed data than the Gaussian distribution admits.

Here is where we may want to change the diffusion. Perhaps our assumption of Gaussian distribution is not appropriate. What would suggest that this is the case?

Well, a Gaussian distribution is completely specified by just two numbers: the mean and the standard deviation. But there are other metrics, called **moments**, that can summarize a distribution. Let's examine the first four moments of a distribution.

The first moment of a distribution relates to a measure of *central tendency* and is represented by the mean.

The second moment of a distribution relates to how far away the data is likely to be from the first moment: in other words, the *uncertainty* about the mean. This is represented by the standard deviation or variance. Standard deviation is in the same units as the distribution as it represents the average distance we expect to be from the measure of central tendency.

The third moment of a distribution relates to whether we are distributed symmetrically around the mean. This relates to a distribution's *skewness* and will be discussed in a later module.

Kurtosis

Finally, let's consider the fourth moment of a distribution. It relates to the kurtosis (or more precisely *excess* kurtosis). Most people associate high kurtosis distributions as having heavy tails. That concept is certainly true. Let's think of a kurtosis another way, though. Remember that kurtosis is the 4th moment. Another way to write the number 4 is 2 squared. Why might we think of it this way?

Every time we see a 2nd moment, we should think uncertainty. In kurtosis, we see the number 2 twice, so it's helpful to think of kurtosis as the *uncertainty about the uncertainty*; and that is exactly what it is. **Kurtosis** represents the uncertainty about estimating the standard deviation.

Suppose we calculate the returns of AAPL stock and determine that it has excess kurtosis significantly bigger than 0. This means that the returns are not normally distributed.

In summary, the 4th moment reminds us that our estimate of volatility itself has noise in it, so we are not certain what the expected distance from the mean truly is.

```
# Let's download and install an R package that has the kurtosis
function ----
#install.packages("PerformanceAnalytics")
library(PerformanceAnalytics)
# Let's fix a starting seed so we all generate the same random
numbers ----
set.seed(2018)
# Let's simulate 950,000 numbers from a Normal(mean=0, sd=1)
data1 = rnorm(9500000, mean=0, sd=1)
# Let's simulate 50,000 numbers from a Normal(mean=0, sd=4)
data2 = rnorm( 500000, mean=0, sd=4)
# Let's calculate the kurtosis of each distribution
kurtosis(data1)
## [1] 0.0005573789
kurtosis(data2)
## [1] -0.01208992
# Let's combine the data in a single distribution and compute its
kurtosis
data = c(data1, data2)
kurtosis(data)
## [1] 10.38978
```

The first distribution is normal with mean 0 and a standard deviation of 1. Its excess kurtosis is effectively 0. The second distribution is normal with mean 0 and standard of 5. Its excess kurtosis is effectively 0. Therefore, each individual distribution has no excess kurtosis. However, when we pool the data into a single distribution, the excess kurtosis becomes more than 10. This is then a very heavy-tailed distribution.

In this example, we know what the standard deviation of the distribution is: 95% of the time, there is a standard deviation of 1 and 5% of the time, there is a standard deviation of 4. The standard deviation of the pooled data is 1.32. We have an excess kurtosis larger than 10. Thus, there is uncertainty in estimating the standard deviation.

Summary

What all of this means is that if we wanted to simulate some behavior about stocks for a portfolio optimization we could choose a diffusion that is a *mixture of normals* rather than a normal distribution. This diffusion would allow us to preserve a standard deviation that has uncertainty associated with its estimation. Indeed, when you priced options using models besides Black-Scholes – such as Heston, SABR, or Variance-Gamma – you are using a volatility that is not constant, but stochastic. Stochastic volatility models can better incorporate excess kurtosis because they treat variance as a variable, rather than as a constant. We need to keep this in mind because when we input values into a portfolio optimization we may assume the variances or covariances are constant when in fact they may not be.

Bibliography

Bertsekas, D. (2017) Dynamic Programming and Optimal Control. 4th edn. Athena Scientific.

Cornuéjols, G., Peña, J., and Tütüncü, R. (2018) Optimization Methods in Finance, 2nd edn. Cambridge University Press.

Collaborative Review Task

In this module, you are required to complete a collaborative review task, which is designed to test your ability to apply and analyze the knowledge you have learned during the week.

Questions

1 Returns

a. Download 1-2 years of price history of a stock.

b. Compute its log return.

c. Compute the mean, standard deviation, skewness, and excess kurtosis of its log return.

d. Repeat for a second stock.

e. Compute the covariance and the correlation. Explain their difference. How do you convert one to the other?

2 Build your own transition

a. Divide the data into 2 uneven parts: the first part is 80% of your data, and the second part is 20%.

b. Categorize each day in the 1-2 year price history as belonging to one of four categories:

i. Both stocks up

ii. Stock #1 up, stock #2 down

iii. Stock #1 down, stock #2 up

iv. Both stocks down

c. Build a transition matrix of portfolio direction that shows your portfolio in four scenarios:

i. From moving together to moving together

That means starting from uu or dd & going to uu or dd

ii. From moving together to moving apart

That means starting from uu or dd & going to ud or du

iii. From moving apart to moving together

That means starting from ud or du & going to uu or dd

iv. From moving apart to moving apart

That means starting from ud or du & going to ud or du

d. How similar is the transition matrix from the first group to the second group?

e. Is the process Markovian?