Chris Kelliher

A Practical Guide to Investment Management, Trading and Financial Engineering

To my amazing daughter Sloane, my light and purpose.
To my wonderful, loving wife Andrea without whom none of my achievements would be possible.
To my incredible, supportive parents and sister and brother, Jen and Lucas.

Contents

Section IV Portfolio Construction & Risk Management

Author Bio

Chris Kelliher is a Senior Quantitative Researcher in the Global Asset Allocation group at Fidelity Investments. In addition, Mr. Kelliher is a Lecturer in the Masters in Mathematical Finance and Financial Technology program at Boston University's Questrom School of Business. In this role he teaches multiple graduate level courses including Computational Methods in Finance, Fixed Income & Programming for Quant Finance. Prior to joining Fidelity in 2019, Mr. Kelliher served as a portfolio manager for RDC Capital Partners. Before joining RDC, Mr. Kelliher served as a principal and quantitative portfolio manager at a leading quantitative investment management firm, FDO Partners. Prior to FDO, Mr. Kelliher was a senior quantitative portfolio analyst and trader at Convexity Capital Management and a senior quantitative researcher at Bracebridge Capital. He has been in the financial industry since 2004. Mr. Kelliher earned a BA in Economics from Gordon College, where he graduated Cum Laude with Departmental Honours, and an MS in Mathematical Finance from New York University's Courant Institute.

Foreword

In March 2018, the Federal Reserve ("Fed") was in the midst of its first hiking cycle in over a decade, and the European Central Bank ("ECB"), still reeling from the Eurozone debt crisis, continued to charge investors for the privilege of borrowing money. US sovereign bonds ("Treasuries") were yielding 3% over their German counterparts ("Bunds"), an all-time high, and unconventional monetary policy from the two Central Banks had pushed the cost of protection to an all-time low.

Meanwhile, across the pond, a sophisticated Canadian pension flipped a rather esoteric coin: A so-called digital put-on Euro/Dollar, a currency pair that trades over a trillion dollars a day. On this crisp winter morning, the EURUSD exchange rate ("spot") was 1.2500. If the flip resulted in heads, and spot ended below 1.2500 in 2 years, the pension would receive $10 million. If the flip were tails, and spot ended above 1.2500, the pension would have to pay $2.5 million. Naturally, the 4 to 1 asymmetry in the payout suggests that the odds of heads were only 25%. Turns out, the flip yielded heads, and in 2 years, spot was below 1.2500.

After the trade, I called Chris, reiterated the pitch, and explained that since January 1999, when EURUSD first started trading, the market implied odds of heads had never been lower. As macroeconomic analysis and empirical realizations suggest that the coin is fair, and there is about 50% chance of getting heads, should the client perhaps consider trading the digital put in 10x the size? In his quintessentially measured manner, Chris noted "we must have a repeatable experiment to isolate a statistical edge". Ten separate flips, for instance, could reduce the risk by 2/3. Moreover, as investors are perhaps loath to lose money with 100% probability, negative bund yields incentivize capital flows to treasuries, and the anomalous rates "carry is a well-rewarded risk premium". Furthermore, as "investors value $1 in risk-off more than $1 in risk-on", does the limited upside in the payout of the digital put also harness a well-rewarded tail risk premium?

I wish I were surprised by Chris's nuance, or objectivity, or spontaneity. Having known him for 8 years though, I have come to realize that he is the most gifted quant I have had the privilege to work with, and this book is testament to his ability to break complex ideas down to first principles, even in the treatment of the most complex financial theory. The balance between rigor and intuition is masterful, and the textbook is essential reading for graduate students who aspire to work in investment management. Further, the depth of the material in each chapter makes this book indispensable for derivative traders and financial engineers at investment banks, and for quantitative portfolio managers at pensions, insurers, hedge funds and mutual funds. Lastly, the "investment perspectives" and case studies make this a valuable

guide for practitioners structuring overlays, hedges and absolute return strategies in fixed income, credit, equities, currencies and commodities.

In writing this book, Chris has also made a concerted effort to acknowledge that markets are not about what is true, but rather what can be true, and when: With negative yields, will Bunds decay to near zero in many years? If so, will $1 invested in Treasuries, compound and buy all Bunds in the distant future? Or will the inflation differential between the Eurozone and USA lead to a secular decline in the purchasing power of $1? One may conjecture that no intelligent investor will buy perpetual Bunds with a negative yield. However, even if the Bund yield in the distant future is positive, but less than the Treasury yield, market implied odds of heads, for a perpetual flip, must be zero. As the price of the perpetual digital put is zero, must the intelligent investor add this option to her portfolio?

Since the global financial crisis, the search for yield has increasingly pushed investors down the risk spectrum, and negative interest rates, and unconventional monetary policy, are likely just the tip of the iceberg. This book recognizes that unlike physics, finance has no universal laws, and an asset manager must develop an investment philosophy to navigate the known knowns, known unknowns and unknown unknowns. To allow a portfolio manager to see the world as it was, and as it can be, this book balances the traditional investment finance topics with the more innovative quant techniques, such as machine learning. Our hope is that the principles in this book transcend the outcome of the perpetual flip.

– *Tushar Arora*

Contributors

Tushar Arora
INSEAD
Paris, France

George Kepertis
Boston University
Boston, Massachusetts

Lingyi Xu
Boston University
Boston, Massachusetts

You Xie
Boston University
Boston, Massachusetts

Maximillian Zhang
Boston University
Boston, Massachusetts

I

Foundations of Quant Modeling

CHAPTER 1

Setting the Stage: Quant Landscape

1.0.1 Coding Example: Option Payoff

The following python code can be used to generate a payoff function for a vanilla call and put options:

```
def call_payoff(S,K):
  return np.maximum(S-K,0)

def put_payoff(S,K):
  return np.maximum(K-S,0)
```

1.0.2 Coding Example: Valuing a Swap Contract

In the following coding example we show how to compute the present value of a swap assuming that the values of the reference variables are known:

```
def price_swap(coupon,reference_rates,rf_rate,delta):
  n_periods = len(reference_rates)

  fixed_leg = 0
  for period in range(1,n_periods+1):
    disc_factor = np.exp(-rf_rate*delta*period)
    fixed_leg += delta*coupon*disc_factor

  floating_leg  = 0
  for period, ref_rate in enumerate(reference_rates,1):
    disc_factor = np.exp(-rf_rate*delta*period)
    floating_leg += delta*reference_rates[period-1]*disc_factor

  return floating_leg - fixed_leg
```

This approach can be used either after the fact, when all fixings for the reference variable have been posted, or when the values for the floating leg can be inferred from a forward or futures curve for the asset.

EXERCISES

1.1 Write a piece of pseudo-code that calculates the payoff for a condor option

1.2 Compare the payoff of a futures contract to a cash position in an equity index.

1.3 Build an option strategy that replicates a short position in the underlying asset.

1.4 Build an options strategy that replicates selling a put option using a call option and a position in the underlying asset

1.5 Compare selling a put option to selling insurance

1.6 Using put-call parity devise a strategy using a put option and the underlying asset that replicates a long position in a call.

1.7 You are asked to build a statistical arbitrage strategy. Describe in detail how you would approach each of the steps in this project, highlighting the assumptions and challenges that you might encounter.

CHAPTER 2

Theoretical Underpinnings of Quant Modeling: Modeling the Risk Neutral Measure

2.1 CALCULUS REVIEW

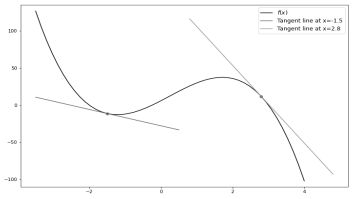
2.1.1 Differentiation

In calculus, the reader may recall that a derivative can be viewed as the rate of change, or slope of a function with respect to a particular variable. It is the change in value of some function, $f(x)$, when the input value is increased by Δx. More formally, it is the limit to this quantity as Δx approaches zero. If we consider the case of driving a car, we are clearly interested in our position, and the distance we can drive, which in this context is $f(x)$, but also are clearly interested in our speed. This quantity, which is also known as the slope or velocity, is the first derivative of the position, or function $f(x)$. Integrating this velocity then tells us the distance that we travel have traveled over a particular time period. Further, when modeling our speed we might want to account for our acceleration, that is how fast we are increasing or decreasing our velocity.

In the following list, we briefly summarize the interpretation of these derivatives:

- **Position or Level**: The value of the function: $f(x)$
- **Slope or Velocity**: The first derivative of the function: $\frac{df}{dx}$
- **Acceleration**: The second derivative of the function: $\frac{d^2f}{dx^2}$

A large component of calculus is learning how to compute these derivatives in different contexts and for different functions, $f(x)$. In the following chart we provide an intuitive visualization of how a derivative calculates a rate of change:



Note that a derivative is the local change to the function and incorporates only the linear behavior of the function. Thus, a derivative can be found by finding the line that the is tangent to the function at a given point. The non-linear behavior in the function, if it exists, would then be captured in higher order derivatives. These higher order derivatives will be an important part of quant finance and options modeling.

The power rule for derivatives, for example, helps us to differentiate a function of the form x^n. In the following list, we summarize some of the most relevant derivative rules, including how to differentiate powers, exponential functions, and logarithms:

- **Power Rule**:

$$\frac{d\left(x^n\right)}{dx} = nx^{n-1} \tag{2.1}$$

- **Derivative of** $\log(x)$:

$$\frac{d\left(\log\left(x\right)\right)}{dx} = \frac{1}{x} \tag{2.2}$$

- **Derivative of** e^x:

$$\frac{d\left(e^x\right)}{dx} = e^x \tag{2.3}$$

- **Chain Rule**:

$$\frac{dz(x,y)}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.4}$$

- **Product Rule**:

$$\frac{d(uv)}{dx} = \frac{du}{dx}v + \frac{dv}{dx}u \tag{2.5}$$

The chain rule in particular, is a foundational concept as it enables us to differentiate complex functions by taking derivatives of the components of the function. This is a concept that we will continue to refer to as we move from the ordinary calculus setting that we review here, to stochastic calculus where we need to model stochastic processes. The above list is by no means an exhaustive list of derivative rules. A more extensive list of these derivative rules can be found in [6] or another calculus textbook.

2.1.2 Derivatives of a Multivariate Function

In many practical settings, the functions that we are working with will be of many variables, and calculus shows us how to extend the context of a derivative to a multivariate setting. In this case, we want to be able to isolate the sensitivity of the function to each underlying variable, which we can accomplish via a so-called partial derivative . In this case of a partial derivative, the rate of change with respect to a given variable is computed while assuming that all other variables do not change. So-called total derivatives, by contrast, compute the changes for a given function as multiple variables are allowed to vary. In the list below, we summarize these two types of multi-variate derivative operations:

- **Partial Derivative**: Measures the rate of change of a function with respect to a particular variable while other variables are held constant [6]. Mathematically, this can be expressed as:

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \{ \frac{f(x + \Delta x, y, \dots) - f(x, y, \dots)}{\Delta x} \} \tag{2.6}$$

- **Total Derivative**: Estimates the rate of change of a given function with respect to its entire parameter set [6]. Mathematically, this can be thought of as the aggregate of the partial derivatives with respect to each variable, and can be written as:

$$df = \sum_{i=1}^{N} \frac{\partial f}{\partial x_i} dx_i \tag{2.7}$$

In many practical cases that we encounter, calculating derivatives analytically using standard table of derivatives may be quite challenging. In this cases, we may instead need to resort to numerical methods for estimating derivatives, a topic that we discuss in chapter 10. As the readers makes their way through the text, the concepts of multivariate calculus, and in particular partial derivatives, will be prevalent throughout. Therefore, the reader is encouraged to ensure that these concepts are clear before moving forward and extending calculus to a stochastic setting. Readers interested in further review on these calculus concepts should consult [6].

2.1.3 Integration

Readers may also recall from their calculus courses that integration is another core concept, and provides us with the tools to calculate the area of a given function, $f(x)$. It can be thought of as the reverse operation of a derivative. More formally, recall that the indefinite integral of a function $f(x)$ can be expressed via the following equation:

$$F(x) = \int f(x)dx \tag{2.8}$$

We say that $F(x)$ is the anti-derivative of $f(x)$ is the following condition holds:

$$F'(x) = f(x) \tag{2.9}$$

That is, if we recover $f(x)$ by taking the derivative of $F(x)$, then we say that $F(x)$ is the anti-derivative of $f(x)$. In the following list, we review several important integration techniques, each of which will be leveraged throughout the book.

- **Integration by Parts**: Integration by parts is a useful technique when we are integrating a complex function where the we break the integral into more tractable parts, with a known antiderivartives. Integration by parts is defined by the following equation, which can be motivated by integrating both sides of the previously specified product rule and rearranging terms:

$$\int udv = uv - \int vdu \tag{2.10}$$

- **Change of Variables**: Another commonly employed integration technique is to change variables, which can be done via the following identity:

$$\int f(g(x))g'(x)dx = \int f(u)du \tag{2.11}$$

  This technique is often employed to simplify the integrands that we are working with, and will be a technique that is employed throughout the text. The reader should recall that a change of variables, or substitution in an integral requires us to changes the limits of integration when we are working with a definite integral.

- **Fundamental Theorem of Calculus**: The fundamental theorem of calculus tells us that a definite integral can be computed according to the following equation:

$$\int_a^b f(x)dx = F(b) - F(a) \tag{2.12}$$

  where $F(a)$ and $F(b)$ are the antiderivative of $f(x)$ evaluated as a and b respectively. Therefore, computing a definite integral will require us to first find the antiderivative of the function in question, and then evaluated it at the integral

limits. Importantly, it should be emphasized from this theorem that the result of a definite integral is a single number. This means that, if we were then to differentiate the result of the integral, we would obtain zero as it is a constant. In the next section we highlight a particular case of differentiating definite integral, notably, when the quantity we are differentiating with respect to is found in the limits of integration.

- **Differentiating Definite Integrals**: As previously stated, the result of a definite integral is a constant, or a number, making its derivative zero. However, if we are differentiating with respect to a function that is contained in the upper or lower integral limit, then the derivative is not zero, but can instead be defined as:

$$F'(x) = \frac{d}{dx}\int_a^x f(t)dt = f(x) \tag{2.13}$$

This is a direct result of the fundamental theorem of calculus and will be leveraged throughout the text.

2.1.4 Taylor Series

Taylor series expansions are another critical piece of mathematical machinery that will be leveraged throughout the text. Taylor series provides a method for approximating a function as the sum of a polynomial of its derivatives [3].

$$\begin{aligned} f(x+h) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!}h^n & (2.14)\\ &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + \ldots & (2.15)\end{aligned}$$

where $f^{(n)}$ is the n^{th} derivative of the function $f(x)$. Notice that the sum begins at $n = 0$, which makes this term simply the value of the function of $f(x)$ evaluated at x [1].

As we can see from this representation, the Taylor series expansion is itself an infinite sum of increasing order derivatives. However, the terms decrease in magnitude as the order of the derivative increases, meaning that in practice we can choose a cutoff k and truncate the series after k derivatives. For example, in the case where $k = 2$, this leads to the following, Taylor Series approximation:

$$f(x+h) \approx f(x) + f'(x)(h) + \frac{1}{2}f''(x)h^2 \tag{2.16}$$

Taylor Series is a concept that we will return to in later chapter in multiple

[1] As 0! equals 1

contexts. For example, later in this chapter, we will leverage a Taylor series expansion to better understand calculus in a stochastic setting. Later, in chapter 9, we will use a Taylor series expansion in the context of optimization techniques. Further, in chapter 10, we will use taylor series expansions to build finite difference estimates.

2.2 REVIEW OF PROBABILITY CONCEPTS

2.2.1 Discrete and Continuous Probability Distributions

Probability distributions are classes of functions that have special properties. In a probability distribution, the function provides the probability of different events, or values occurring. There are certain restrictions to functions that are probability distributions. For example, we cannot have an event with a negative probability, and therefore, probability distributions must be defined by non-negative probabilities. They also must sum to 1. That is, we know with certainty that one of the exhaustive set of events will occur. Probability distributions are broken into discrete and continuous distributions, with discrete distributions allowing a finite set of values and continuous distributions allow the distribution to take on any value. The properties of each type of distribution is summarized in the following list:

- **Discrete Probability Distribution**: A discrete probability distribution is one is which a finite set of values can occur each with a corresponding probability. In order for a discrete function to be a probability distribution certain requirements must be met. Notably, a discrete probability distribution must be defined by non-negative probability for all outcomes, and must have probabilities that sum to 1:

$$\sum_{i=1}^{N} p_i = 1 \tag{2.17}$$

$$p_i \geq 0 \quad \forall i \tag{2.18}$$

  In the next section we discuss some of the most common examples of discrete distributions, such as Bernoulli and Binomial distributions.

- **Continuous Probability Distribution**: A continuous probability distribution is one in which any value can occur. Because any value can occur, continuous probability distributions are defined by an infinite set of possible values. This means that the probably of any exact value is zero, but the probability of being within a given range will generally speaking not be. In order for a continuous function to be a probability distribution the following conditions must hold:

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{2.19}$$

$$f(x) \geq 0 \quad \forall x \tag{2.20}$$

In the next section, we detail many of the most common continuous probability distributions. Of particular note is the normal distribution, which is perhaps the most commonly applied distribution in practical applications.

2.2.2 Probability Density and Cumulative Density Functions

Discrete and continuous probability distributions are defined by their probability density and cumulative distribution functions, each of which are briefly reviewed in the following list:

- **Probability Density Function (PDF)**: A probability density function, loosely speaking measures the probability that a random variable X takes a specific value x. In the case of a discrete distribution, the meaning of a PDF, $f(x)$ is clear and can be written as:

$$\begin{aligned} f(x) &= P(X = x) && (2.21) \\ &= \sum_{i=1}^{N} \mathbb{1}_{x_i = x} p_{x_i} && (2.22) \end{aligned}$$

Here we calculate the PDF of a discrete distribution by summing over all probabilities where the outcome matches the specified value x. It should be emphasized that the PDF measures the probability of an exact value occurring. That is, it is the probability at a specific point. While this makes sense in the context of a discrete probability distribution, it is less clear in the context of a continuous probability distribution, where the probability of any specific value is zero because there are infinite values. In this case, we instead interpret the PDF as the probability of a random variable X being within an interval $[x, x + \Delta_x]$ for infinitesimally small Δ_x. For a continuous distribution, we can express the probability of obtaining a value for a random variable X that is between a and b as:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{2.23}$$

To obtain the probability density function for a continuous probability distribution, we can then take the limit as a and b converge. That is, the PDF of a continuous probability density function at x can be expressed as:

$$f(x) = P(x \leq X \leq x + \Delta_x) \tag{2.24}$$

where Δ_x is understood to be very small as we are taking the limit of (2.23).

- **Cumulative Distribution Function (CDF)**: While the PDF measures the probability at a given point, the CDF measures the probability that a random variable X is at or below a given value x. We can write the equation for the CDF as:

$$\begin{aligned} F(x) &= P(X \leq x) && (2.25) \\ &= \sum_{i=1}^{N} \mathbb{1}_{x_i \leq x} p_{x_i} && (2.26) \end{aligned}$$

Just as before, we calculate the CDF by summing the probability over the set of possible outcomes conditional on the value being below our specified threshold x. Similarly, for a continuous distribution, the CDF can be computed as an integral of the PDF:

$$\begin{aligned} F(x) &= P(X \leq x) && (2.27) \\ &= \int_{-\infty}^{x} f(u)du && (2.28) \end{aligned}$$

Importantly, this means that we can view the PDF of a continuous distribution as the derivative of the PDF as follows:

$$f(x) = \frac{dF(x)}{dx} \quad (2.29)$$

2.2.3 Expected Value, Variance and Correlation

In this section we review some of the core concepts of working with probability distributions that will be central to our modeling efforts as quants. At the heart of being a quant is being able to understand different probability distributions. This entails knowing the mean, or expected value of a given distribution, and also the uncertainty, which we might summarize by the variance, in a distribution. Additionally, when solving derivatives pricing problems, we are tasked with calculating a different expectation, that is, an expectation of some payoff function. A brief summary for calculating each of these quantities is provided in the following list. This is meant to solely be a brief review of these concepts. Readers looking for a refresher on basic probability concepts should consult [1].

- **Mean**: The mean, or expected value of a distribution is the value that we would expect to get, on average, after a large number of trials. For a discrete probability distribution, its mean can be computed using the following equation:

$$\mathbb{E}[X] = \sum_{i=1}^{N} x_i p_i \quad (2.30)$$

In the case of a continuous probability distribution, the discrete sum in (2.30) would be replaced with a continuous integral, and we would have the following expression for the mean or expected value:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) dx \tag{2.31}$$

An important property of the expected value of a distribution is that, generally speaking the following holds:

$$\mathbb{E}\left[\sum_{i=1}^{N} X_i\right] = \sum_{i=1}^{N} \mathbb{E}\left[X_i\right] \tag{2.32}$$

Mathematically, this formula can be motivated by employing Fubini's theorem to switch the order of the sum and expectation which holds under fairly wide circumstances. This is an important point as it shows we can compute the expectation of a sum of random variables by modeling the variables individually. That is, we don't need to understand their correlation structure in order to compute (2.32). This will not be the case when we compute non-linear functions, such as variance and contingent payoffs.

- **Variance**: Variance is a measure of uncertainty in the distribution, or width of the distribution around the mean. Mathematically, we define the variance as the expected squared distance between a value from the probability distribution, and its mean:

$$\begin{aligned} \text{var}(X) &= \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 && (2.33) \\ &= \left(\int_{-\infty}^{\infty} x_i^2 p_i dx_i\right) - \mathbb{E}[X]^2 && (2.34) \end{aligned}$$

We also often use the square root of this variance, which is known as the standard deviation of X:

$$\sigma_X = \sqrt{\text{var}(X)} \tag{2.35}$$

Variance is characterized by a few important properties. First, the variance of a constant is zero. Secondly, if a random variable is scaled by a constant, then its variance is scaled by the square of that constant. That leads to the following condition:

$$\text{var}(a + bX) = b^2 \text{var}(X) \tag{2.36}$$

Additionally, we can write the variance of the sum of two random variables as the following equation:

$$\text{var}(X + Y) = \text{var}(X) + \text{var}(Y) + 2\text{cov}(X, Y) \tag{2.37}$$

Notice that, unlike in the case of expected value, the variance of the sum is dependent on the covariance, or correlation between the two random variables. Notably, if the random variables are independent, then this term goes away and simplifies to the sum of the individual variances. More generally, however, this is not the case, and we can see that the variance of the sum will be smallest when the covariance term is as small as possible. This is a fundamental concept that we will use when learning about portfolio optimization techniques, and the benefits of diversification .

- **Covariance**: The previous metrics summarized the distribution of a single random variable. In quant finance applications, we often work with multiple variables. For example, we might want to understand the behavior of multiple asset classes, and better understand the dependence between them. Covariance and correlation can help us in this regard, as both are metrics of the co-movement of two random variables. Mathematically, the covariance of two random variables, X and Y can be written as:

$$\begin{aligned} cov(X,Y) &= \mathbb{E}[(X-\mu_x)(Y-\mu_y)] && (2.38)\\ &= \mathbb{E}[XY]-\mu_x\mu_y && (2.39) \end{aligned}$$

  As we can see, calculating the covariance of two random variables involves calculating the expectation of their products, which involves computing an integral against a two-dimensional distribution, and subtracting their means.

- **Correlation**: The correlation coefficient is a normalized measure of co-movement of two random variables, where we normalize by the standard deviation of the random variables:

$$\rho = \frac{cov(X,Y)}{\sqrt{\sigma_X \sigma_Y}} \tag{2.40}$$

  where σ_X and σ_Y are the standard deviation of X and Y, respectively.

  ρ is known as the correlation between X and Y and is strictly speaking $-1 \leq \rho \leq 1$. This normalization makes correlation coefficients easier to compare across different random variables than covariances.

  In practice, when we are working with large systems defined by large asset universes, it is useful to aggregate the correlation and covariance of the assets into a single matrix, which we refer to as the correlation matrix, or covariance matrix.

- **Expectation of Payoff**: More generally, for a discrete distribution, we can calculate the expectation of a given payoff function $V(X)$ via the following sum:

$$\mathbb{E}\left[V(X)\right] = \sum_{i=1}^{N} V(x_i)p_i \tag{2.41}$$

Similarly, for a continuous distribution, calculating the expectation of a payoff function amounts to computing a single, one-dimensional integral, as shown in the following equation:

$$\mathbb{E}[V(X)] = \int_{-\infty}^{\infty} V(x)f(x)dx \tag{2.42}$$

As we will soon see, derivatives pricing and options modeling largely comes down to the calculation of these types of integrals, or sums against different payoff functions that are determined by the options structures. In the case of European options, such as the call and put options introduced in chapter 1, the structures are only a function of the asset's terminal payoff, meaning they can be formulated as the type of integral in (2.42).

2.2.4 Common Probability Distributions

- **Bernoulli Distribution**: A Bernoulli distribution is a discrete probability distribution where one of two events takes place. We can think of these events as a binary variable, or success or failure. The Bernoulli distribution is defined by a single parameter, p, which specifies the probability of a "success" event. The PDF for a Bernoulli distribution can be written as:

$$f(k;p) = \begin{cases} p & k = 1 \\ 1-p & k = 0 \end{cases} \tag{2.43}$$

Using the definition of expectation and variance above, it is easy to see that the Bernoulli distribution has mean p and variance $p(1-p)$.

- **Binomial Distribution**: The Binomial distribution, like the Bernoulli distribution, is a discrete probability distribution . Whereas the Bernoulli distribution measures the probability distribution of a single binary event, such as success or failure, a binomial distribution measures the number of successes in a specified number of trials. The Binomial distribution is defined by two model parameters, p, the probability of a "success" event, and n, the number of independent trials that we are modeling. The PDF for a Binomial distribution can be written as:

$$f(x;p) = \binom{n}{k} p^k \left(1-p\right)^{n-k} \tag{2.44}$$

Once again, we can use the definitions above to obtain the mean and variance of the binomial distribution which are known to be np and $np(1-p)$, respectively.

- **Standard Normal Distribution**: Normal distributions play a fundamental role in probability and statistics, and are a core building block of quant finance applications. They are continuous probability distributions and are sometimes referred to as a bell-curve. Normal distributions occur frequently because of the Central Limit Theorem, which states that, often, when independent events are added, the distribution converges to a normal distribution as the number of trials increases. A standard normal distribution is a special type of normal distribution with a specified mean and variance. The PDF for this standard normal distribution can be written as:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{2.45}$$

- **Normal Distribution**: The normal distribution is a generalization of the standard normal distribution where the mean and variance, μ and σ^2 are parameters. The PDF for a normal distribution can be expressed as:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2.46}$$

  The reader should notice that if $\mu = 0$ and $\sigma = 1$ then we recover the PDF for the standard normal distribution presented above. Importantly, we can easily convert from a standard normal distribution to a normal distribution with an arbitrary mean and variance using the following well-known identity:

$$X = \mu + \sigma Z \tag{2.47}$$

  An important feature of a normal distribution is that it is fully defined by its mean and variance, meaning its skewness and excess kurtosis are equal to zero regardless of the parameters chosen.

- **Poisson Process**: A Poisson process is a discrete probability distribution that defines the number of independent events in a given, fixed interval. It is defined by a single parameter, λ, which controls the frequency of the events occurrence. It is often referred to as a rate or intensity. The PDF for a Poisson process, or distribution can be written as:

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2.48}$$

- **Exponential Distribution**: Where a Poisson Process tells us about the distribution of the number of events, an exponential distribution tells us about

the distribution of the time between events. In fact, the exponential distribution measures the distribution of the time between independent events in a Poisson process. Like a Poisson process, an exponential distribution is defined by a single parameter, λ, which again measures the rate or intensity of event occurrences. Larger values for the paramter λ lead to more frequent events and smaller times between events, on average. The PDF of an exponential distribution can be written as:

$$f(x;\lambda) = \lambda e^{-\lambda x} \tag{2.49}$$

where $x \geq 0$, otherwise $f(x;\lambda) = 0$. It can be shown that the mean of an exponential distribution is $\frac{1}{\lambda}$ and its variance is $\frac{1}{\lambda^2}$. The CDF of an exponential distribution can also be computed by applying standard integration techniques to (2.49), as we explore in the end of chapter exercises.

- **Gamma Distribution**: The Gamma distribution is a continuous probability distribution that is a generalization of the exponential distribution. It is defined by two parameters, κ and θ, which measure the shape and scale respectively. The quantity $\frac{1}{\theta}$ can then be interpreted as a rate parameter that is analogous to the λ parameter in the exponential distribution. The PDF for a Gamma distribution with parameters κ and θ can be written as:

$$f(x;\kappa,\theta) = \frac{x^{\kappa-1}e^{-\frac{x}{\theta}}}{\theta^{\kappa}\Gamma(\kappa)} \tag{2.50}$$

- **Student T Distribution**: Finally, the Student-T distribution is a continuous probability distribution that often arises in the context of statistics and econometrics. For example, as we will see in the next chapter, a student-t distribution is used to compute t-statistics in our regression models, which is used to judge statistical significance. The student-t distribution is defined by a single parameter, ν, which defines the degrees of freedom. Its PDF can be written as:

$$f(t;v) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} \tag{2.51}$$

Each of the distributions introduced above has applications in quant finance, and each will be referred to in later chapters of the text. For example, the binomial distribution will arise when we construct a binomial tree in the next section. Normal distributions will similarly arise anytime we use Brownian Motion as a building block, which will be the case in the vast majority of applications. Exponential distributions and poisson processes will turn out to be significant when we model processes with jumps . Additionally, exponential distributions will be central to our modeling efforts when tackling default risk.

2.2.5 Coding Example: Call Option Price via Binomial Tree

In this coding example we detail how a recombining binomial tree can be implemented in Python using the formulas detailed in the previous section:

```
def binomialTreeCallOptionPrice(N, r, sigma, S_0, T, K):
    deltaT = T/N
    u = np.exp(sigma * np.sqrt(deltaT))
    d = 1/u
    p = (np.exp(r*deltaT) - d)/(u-d)

    priceTree = [n*[None] for n in range(1,N+2)]

    for i in range(N+1): #each time period
        for j in range(i+1): #each possible price within each time
    period
            priceTree[i][j] = S_0 *  (u**(j)) * (d**(i-j))

    callPriceTree = [n*[None] for n in range(1,N+2)]
    #terminal payoffs
    callPriceTree[-1] = [max(px-K,0) for px in priceTree[-1]]

    for i in range(N-1,-1,-1): #each time period starting from the
    end
        for j in range(i+1): #each price within each time period
            callPriceTree[i][j] = np.exp(-r*deltaT)*((1-p)*
    callPriceTree[i+1][j] + p*callPriceTree[i+1][j+1])

    return callPriceTree[0][0]
```

The reader should note that in this example a European call option is priced via the binomial tree, however, is encouraged to try to modify the function to handle additional payoff structures.

EXERCISES

2.1 Consider the binomial distribution as defined in (2.44). Derive the mean and variance for this distribution as a function of the model parameter p.

2.2 Consider the standard normal distribution as described in (2.45). Derive the estimates of the mean and variance.

2.3 Consider a standard normally distributed random variable Z as well as the random variable X which is defined as:

$$X = \mu + \sigma Z \tag{P2.1}$$

where μ and σ are known constants. Show that X is normally distributed and derive its mean and variance.

2.4 Consider the PDF of the exponential distribution as defined in (2.49). Calculate the cumulative distribution function for this distribution.

2.5 **Binomial Tree Option Pricing**:

a. Using a binomial tree calculate the price of a three-month call option with strike 105 on an asset that currently trades at 100 for different values of sigma. Assume the asset pays no dividends and that interest rates are equal to zero.

b. Suppose you buy one unit of this call option using the price obtained when $\sigma = 0.1$ and the market rallies leading to a new price of the underlying asset of 110. What is the value of this option now, and how much money have you made or lost?

c. Now suppose the market instead sold off, leading to a new price for the underlying asset of 90. How much is your option position worth now, and what is your profit? Comment on the difference between the amount you made when the asset price increased, and the amount you lost when the asset price declined.

d. Finally, assume the volatility in the market increases, leading to a new σ for the option of 0.2. What is the value of the option now, and what is our profit or loss? Comment on the implications of this for options holders.

2.6 **Digital Option Pricing via Binomial Tree**:

a. Using a binomial tree calculate the price of a one-year digital call option which pays \$1 if the asset finishes above 100 in one-year and zero otherwise. Assume that interest rates and dividends are equal to zero and that the asset currently trades at 100. Further, assume that the volatility parameter, σ, is known to be 0.25.

b. Comment on the difference between the price of this digital call option and the internal Binomial tree parameter u. Vary σ and discuss whether this relationship changes. Why does it (or doesn't it) change?

2.7 Consider the following stochastic differential equation which is known as the CEV model:

$$dS_t = rS_t dt + \sigma S_t^{\beta} dW_t \tag{P2.2}$$

Derive the PDE that corresponds to this SDE using the Feynman-Kac formula or the replication argument presented in the chapter.

2.8 Compare the properties of the Bachelier and Black-Scholes models introduced in the chapter. Describe the differences in model parameters and the resulting strengths and weaknesses. Which model would you select (and why) for modeling the following asset classes:

- Equities

- Interest Rates
- Foreign Exchange

CHAPTER 3

Theoretical Underpinnings of Quant Modeling: Modeling the Physical Measure

3.0.1 Coding Example: Using OLS to Find Regression Betas

In this coding example we show how the OLS regression formula for the coefficient estimates detailed in (??) can be implemented in Python to obtain the coefficients in a multe-variate linear regression model:

```
1 import numpy as np
2
3 Xs = np.concatenate((np.full((100, 1), 1), np.random.rand(100, 10)),
      axis=1)
4 Ys = np.random.rand(100, 1)
5
6 # OLS Betas
7 betas = np.linalg.inv(Xs.T @ Xs) @ Xs.T @ Ys
8 print(betas)
```

The reader should notice that the first column in the explanatory variables has a constant value of 1 to represent an intercept . In chapter 5 we will show to use an internal Python package to implement regression models in Python, and the reader is encouraged to compare the coefficients obtained using this method to the built-in Python package.

3.0.2 Coding Example: Portfolio Return and Volatility

In this coding example we detail how to leverage the formulas in the previous section to calculate the return and volatility of a portfolio in Python:

```
1 def portfolioStats(expectedReturnVector, weightsVector,
      covarianceMatrix):
2     """
3     expectedReturnVector & weightsVector: np arrays of vector columns
4     covarianceMatrix: 2D symmetric np array
5     """
```

```
    portfolioExpRet = expectedReturnVector.T@weightsVector
    portfolioVol = np.sqrt(weightsVector.T @ covarianceMatrix @
    weightsVector)
    return portfolioExpRet[0,0], portfolioVol[0,0]
```

3.0.3 Coding Example: Bootstrapping Synthetic Returns

In the following coding example, we detail how to implement a simple bootstrapping algorithm in Python:

```
from random import choices

def bootstrapSyntheticReturns(historicalReturns, numberOfPaths,
    pathLength):
    nAssets = 1
    if (len(historicalReturns.shape) > 1):
        nAssets = historicalReturns.shape[1]

    paths = np.zeros((pathLength, nAssets,numberOfPaths))
    for path in range(numberOfPaths):
        dateIdcs = choices(range(1, len(historicalReturns)), k =
    pathLength) #with replacement
        paths[:, :, path] = historicalReturns[dateIdcs, :]

    return paths
```

EXERCISES

3.1 Describe the form of market efficiency that persistent excess returns from the following strategies would violate:

- Technical Analysis
- Value Strategy Based on Price to Earnings Ratio
- Strategy that buys stocks in advance of January and sells them immediately following month-end
- Machine Learning NLP model based on parsing economic news
- Strategy based on the current slope of the yield curve
- ARMA mean-reversion model

3.2 Derive the OLS estimator for the following modified univariate regression equation with no intercept:

$$Y = \beta X + \epsilon \tag{P3.1}$$

3.3 Consider an investor who has access to the following investments with the specified investment properties: Calculate the expected return and volatility of the following portfolios:

Investment	Expected Return	Volatility
A	5%	10%
B	7%	10%
C	10%	15%

- Fully invested portfolio in investments A, B & C
- Equally Weighted portfolio
- Portfolio whose weights are inversely weighted to their volatility

Under the assumptions of:

- Perfect Positive Correlation
- Uncorrelated Investments
- Perfect Negative Correlation

Comment on the properties of the different portfolio weighting schemes and their sensitivity to correlation. Also discuss how any diversification benefit changes as we modify the correlation assumptions.

3.4 **Regression in Practice**:

a. Download data for your favorite stock ticker and clean the data as appropriate.

b. Download data for the Fama-French factors from Ken French's website.

c. Plot the contemporaneous returns of the stock against each Fama-French factor, with the stock returns on the y-axis and the factor returns on the x-axis. Are there strong relationships between the variables? Do the relationships appear to be linear?

d. Find the OLS estimator of the coefficients for each Fama-French factor with the stock returns as the dependent variable. Comment on the significance of each factor and what that implies about the properties of your favorite stock.

e. Calculate the residuals, ϵ in your regression model with the β's fixed and comment on whether there properties match the underlying assumptions. Are they i.i.d.? Do they contain autocorrelation?

Practical regression question (include question about the assumptions of linear regression)

3.5 **Time Series Analysis in Practice**:

a. Download data for the ticker SPY and clean as needed.

b. Perform a stationarity test on the SPY price series. Are the prices stationary or non-stationary?

c. Difference the data by computing SPY log-returns. Estimate an AR(1) model on this differenced series. Is the coefficient you obtain what you would expect? What does it imply about market efficiency, and mean-reversion in the SPY index?

3.6 Comment on whether the following statements are true or false. Please justify each answer.

a. Prices implied from the options market are a forecast of what the asset price will be at expiry.

b. Drift in the risk-neutral measure incorporates risk preferences and is therefore usually higher than drift in the physical measure.

c. Hedging and replication arguments for options structures enable us to work in the risk-neutral measure. In the absence of these replication arguments, we must estimate the drift in a world that incorporates investors risk-aversion.

d. Modeling the risk-neutral and physical measure rely on completely different skills that have no overlap.

3.7 **Bootstrapping in Practice:**

a. Download data for the following set of ETFs:

Ticker	Description
SPY	US Equity Index
EFA	Developed Market Equity Index
EEM	Emerging Market Equity Index
VXX	VIX Futures ETN

b. Calculate the statistical properties of each ETF such as its historical return and volatility.

c. Consider an equally weighted portfolio of the four assets. Calculate the comparable historical risk and return statistics and comment on any differences in performance.

d. Using bootstrapping, simulate a set of synthetic data that last one year. Generate 10000 such synthetic data series and calculate the statistical properties of the assets and the portfolios.

e. Plot a histogram of the distribution of returns over the bootstrapped paths. Do they look reasonable? What are the best and worst case scenarios for the equally weighted portfolio? How does that compare to history?

f. Compare the expected return, volatility and correlation between the assets in the bootstrapped paths, by averaging over all paths, to the properties of the underlying historical data. Have the returns, volatilities and correlations converged? Why or why not?

CHAPTER 4

Python Programming Environment

4.0.1 Plotting / Visualizations

The following coding example shows how to leverage the matplotlib library to create each type of plot:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  retsX = np.random.normal(0.0, 1.0, size=1000)
5  retsY = np.random.normal(0.0, 1.0, size=1000)
6
7  #time series plot
8  plt.plot(retsX)
9  plt.show()
10
11 plt.plot(retsY)
12 plt.show()
13
14 #histogram
15 plt.hist(retsX)
16 plt.show()
17
18 plt.hist(retsY)
19 plt.show()
20
21 #scatter plot
22 plt.scatter(retsX, retsY)
```

4.1 WORKING IN A MULTI-PROGRAMMER ENVIRONMENT

When coding scalable models in Python quants are often working in teams. When doing so, quants need a means of coordinating their changes to the code and ensuring that the changes that they make don't negatively impact other's work. Many programs exist to facilitate this task, most notably git/github [4] .

The motivation for using git, or another similar tool, is partly to help coders

coordinate and merge their changes to a shared codebase. Additionally, these tools provide a robust means of version control. That is, git enables us to keep tract of all historical changes made to our codebase in an organized way. It even gives us a way to revert back to a previous version if necessary. Quant teams at both buy-side and sell-side institutions generally have strict release processes for their models. Using git enables firm's to keep historical code for all previous production versions of the code and enables a quant to roll back if necessary.

Because of this, git has benefits for both large teams and quants working individually. Most notably, quants working individually will benefit from the version control aspect of git, as will larger organizations with formal release processes. Teams using git will also benefit from easier code coordination and merging of new changes to the codebase.

Git consists of the following three layers of repositories :

- Local Working Copy
- Local Repository
- Remote Repository

The first layer is a copy of the code files on each user's machine, which the user will continually update as they program. This will not be viewable by any other users. The next layer is also stored on the user's machine, and is called the local repository. The local repository is also unaccessible to other users, and is where a user might store code that has been tested and seems safe but is not necessarily production ready [1]. Moving files from a local working copy to the local repository is called a commit, and is done using the git commit command. The final layer in a git infrastructure is the remote repository, which is accessible to all users. The remote repository is meant to only contain changes to the code base that have been finalized and tested. Moving files from a local repository to a remote repository is called a push, and is done via the git push command.

Some of the most common tasks in git are to add new files to a codebase, remove files from a codebase, to update existing files in the local or remote repository, or to get the latest changes from the remote repository.

The most common commands used in git are:

- **git status**: lists all changes you have made in your local files and local repository.
- **git add**: adds a new local file to your local repository.
- **git commit**: takes changes to all local files and commits them to your local repository (where they are still only visible to you).
- **git push**: pushes all committed changes from the local repository to the remote repository.

[1] Or ready for others to view

- **git pull**: retrieves changes from the remote repository.
- **git merge**: merges your changes with changes from a remote repository.

The merge command in particular is how a user would synchronize changes that they've made to their code if that file has been changed by another user as well. In some cases, git may be able to complete this merge automatically. In other more complex cases, git may not know how to conduct the merge, and some manual intervention may be required [2].

EXERCISES

4.1 Consider the following code:

```
1     str1 = "advanced"
2     print(str1)
3
4     str2 = "programming"
5     print(str2)
6
7     str = str1 + str2
8     print(str)
9
10    test = "gram" in str
11    print(test)
12    print(str[3])
13
14    str[1] = "p"
15    print(str)
16
```

What will be the result of the code? What will be printed to the console?

4.2 What will be result of the following Python code?

```
1     print (14 / 4, 14 % 4)
2
```

a. 3.5 3.5

b. 3.5 2

c. 3 3

d. 2 2

e. 3 2

f. The program will produce an exception

[2] In these situations, the git stash command is of great use.

4.3 Consider the following code that implements the Black-Scholes formula for a European call:

```
def callPx(s_0 , k, r, sigma , tau ):
sigmaRtT = ( sigma * math . sqrt ( tau ))
rSigTerm = (r + sigma * sigma / 2.0) * tau
d1 = ( math.log (s_0 / k) + rSigTerm ) / sigmaRtT
d2 = d1 - sigmaRtT
term1 = s_0 * norm.cdf(d1)
term2 = k * math.exp (-r * tau ) * norm.cdf (d2)
return term1 - term2

```

What happens when a negative value is passed for s_0? How about a negative sigma? Add the proper exception handling to this function to handle these parameter values.

4.4 What will be the output of the following Python program?

```
greeting = "Hello, world"
greeting[0] = "X"
print(greeting)

```

a. Xello, world

b. Hello, world

c. The program will produce an exception

4.5 Consider the following code containing price data for a few ETFs:

```
tuple1 = ("SPY", "S&P Index", 290.31)
tuple2 = ("XLF", "Financials", 28.33)
tuple3 = ("XLK", "Technology", 75.60)
tuples = [tuple1, tuple2, tuple3]
spyClose = tuples[x][y]
print(spyClose)

```

What values of x & y enable us to extract the closing value of the S&P (290.31) from the variable tuples?

a. $x = 2, y = 0$

b. $x = 1, y = 1$

c. $x = 0, y = 2$

d. None of the above

4.6 A Fibonacci sequence is a sequence that starts with 0 and 1, and then each number equals the sum of the previous two numbers. That is:

$$\text{Fib}_n = \text{Fib}_{n-1} + \text{Fib}_{n-2}$$

Write a Python function that implements a Fibonacci sequence using a recursive function.

4.7 Explain the difference between a Local Working Copy, a Local Repository and a Remote Repository in git. What commands would you use to check in your code to the Local and Remote Repositories respectively? Why might you check in your code to the Local Repository but not the Remote Repository?

CHAPTER 5

Programming Concepts in Python

5.0.1 Coding Example: Covariance Matrix Calculations

In this coding example, we show how to compute the covariance and correlation matrices of stock returns:

```
1  import numpy as np
2
3  ret_aapl = np.random.random(size=200)
4  ret_spy = np.random.random(size=200)
5
6  # calculate the covariance matrix using np.cov()
7  print(np.cov(ret_aapl, ret_spy))
8
9  # calculate the correlation matrix using np.corrcoef()
10 print(np.corrcoef(ret_aapl, ret_spy))
```

Note that covariance and correlation matrices are symmetric, and positive semi-definite. Further, we know that the diagonals in a covariance matrix consist of the variances of each variable, while the other entries represent the covariance between each pair of variables. Symmetry of a covariance matrix follows directly from the properties of covariance, that is: COV(A,B) = COV(B,A).

Likewise, the diagonals of the correlation matrix consist of 1's, which represents the correlation of each variable with itself. All the other entries are the correlation between each variable pair. It's easy to show that CORR(A,B) = CORR(B,A) resulting in a symmetric matrix.

In practice, the magnitude of variance and covariance terms depends on the magnitude of the underlying data. Series with large values are likely to lead to larger variances and covariances, all else equal, and vice versa. On the other hand, the correlation coefficient is normalized and is always between -1 and 1. Therefore, it is more appropriate to compare correlations over a cross-section of variables than covariances. In these cases, it is best practice to base our analysis on a correlation matrix.

5.0.2 Coding Example: Working with Time Series Data

In the following coding examples we detail many of the convenient statistical functions embedded within a series object and show how easy it is to leverage this functionality on a set of time series data:

```
df_rets = df_px.pct_change() # simple returns
df_log_rets = np.log(df_px / df_px.shift(1)) #log returns

df_rets.count()              # count
df_rets.mean()               # mean
df_rets.rolling(10).mean()   # rolling mean

df_rets.median()             # median
df_rets.quantile(q=0.75)     # quantile

df_rets.std()                # standard deviation
df_rets.rolling(10).std()    # rolling standard deviation
```

The reader should note that in this example we assume that we begin with a Dataframe named df_px containing price data.

5.0.3 Coding Example: Working with Panel Data

In this coding example, we show how data can be pivoted when working with a data frame:

```
df_panel = pd.DataFrame(data={'date': ['2021-2-20']*2 + ['2021-2-21']*2 + ['2021-2-22']*2 + ['2021-2-23']*2,
  'ticker': ['SPY','AAPL']*4,
  'return': np.random.random(size=8)})

# make a pivot table of adjusted close prices out of the panel data
df_pivot = df_panel.pivot(index='date', columns='ticker', values='return')

# transform the pivot table back to the panel data layout
df_panel_new = df_pivot.reset_index().melt(id_vars=['date'], value_vars=['SPY','AAPL'], value_name='return').sort_values('date', ignore_index=True)
```

This ends up being a very useful feature when working with panel data, as there may be times when we want all elements of the panel to present as different rows, and other times when we prefer each element in a panel to be displayed in a separate set of columns.

5.0.4 Coding Example: Testing for Autocorrelation in Stock Returns

In this coding example we detail how to perform an autocorrelation test on stock data using Yahoo Finance's API:

```
prices = data.get_data_yahoo('SPY', start='2000-03-01', end='2021-02-28').loc[:, 'Adj Close']
rets = prices.pct_change(1)
```

```
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(prices
    ) #check the price series for stationarity
arma = ARIMA(rets.to_numpy(), order=(1, 0, 0)).fit() #check whether
    the returns follow an AR(1) process
arma.summary()
```

The reader should notice that we first conduct an augmented dickey fuller test on the price series. Next, we calibrate the differenced, return series to an ARMA model with 1 AR lag and 0 MA lags.

5.0.5 Coding Example: Using Inheritance to Create a Generic Stochastic Process Class

In this coding example we show how to use inheritance to create a generic stochastic process class that easily supports switching between models / SDEs:

```
import numpy as np
import pandas as pd
import math
from scipy.stats import norm

#Define the base class
class StochasticProcess:
  def __init__(self, tau=1.0, S0=100.0, strike=100.0, rf=0.0, div
    =0.0):
    self.T = tau
    self.S = S0
    self.K = strike
    self.r = rf
    self.q = div

  def price(self):
    print("method not defined for base class")

#Define the derived class
class BlackScholesProcess(StochasticProcess):
  def __init__(self, sig=0.1, tau=1.0, S0=100.0, strike=100.0, rf
    =0.0, div=0.0):
    self.sigma = sig
    StochasticProcess.__init__(self, tau, S0, strike, rf, div)

  def price(self):
    sigmaRtT = (self.sigma * math.sqrt(self.T))
    rSigTerm = (self.r + self.sigma * self.sigma/2.0) * self.T
    d1 = (math.log(self.S/self.K) + rSigTerm) / sigmaRtT
    d2 = d1 - sigmaRtT
    term1 = self.S * norm.cdf(d1)
    term2 = self.K * math.exp(-self.r * self.T) * norm.cdf(d2)
    return  term1 - term2

bsProcess = BlackScholesProcess(0.1)
px = bsProcess.price()
```

5.0.6 Abstract Base Classes

```
import abc

class AbstractBaseClass(ABC):

    def __init__(self, r: float, q: float, tau: float):
        self.r = r
        self.q = q
        self.tau = tau

    @abc.abstractmethod
    def abstract_method(self) -> float:
        pass

class DerivedClass(AbstractBaseClass):

    def __init__(self, r: float, q: float, tau: float, sigma: float):
        super().__init__(r, q, tau)
        self.sigma = sigma

    def abstract_method(self) -> float:
        print("must define abstract_method in the base class")
        return 0.0
```

5.0.7 Factory Pattern

```
class BaseStochProc:
    def __init__(self, s0: float) -> None:
        pass

class BlackScholesStochProc(BaseStochProc):
    pass

class BachelierStochProc(BaseStochProc):
    pass

class StochProc(Enum):
    BASE_STOCH_PROC = 1
    BS_STOCH_PROC = 2
    BACHELIER_STOC_PROC = 3

class StochProcFactory:
    @staticmethod
    def create_stoch_proc(sp: StochProc, s0: float) -> BaseStochProc:
        if sp == StochProc.BASE_STOCH_PROC:
            stoch_proc = BaseStochProc(s0)
        elif sp == StochProc.BS_STOCH_PROC:
            stoch_proc = BlackScholesStochProc(s0)
        elif sp == StochProc.BACHELIER_STOC_PROC:
            stoch_proc = BachelierStochProc(s0)
        return stoch_proc
```

5.0.8 Singleton Pattern

```
class DBConnection:

    db_conn = None

    @classmethod
    def get_instance(cls):
        if cls.db_conn is None:
            cls.db_conn = DBConnection()
        return cls.db_conn
```

Note that the member variable db_conn is defined outside the constructor. This makes the variable global rather than an attribute of a specific instance of the class. This variable must be global so that it is accessible in the global getInstance function that creates the DBConnection object.

5.0.9 Template Method

The following coding sample shows how we might use this approach to implement a generic simulation algorithm via the template method:

```
import abc

class Simulate(ABC):

    def __init__(self, r: float, q: float, tau: float):
        self.r = r
        self.q = q
        self.tau = tau

    def run_simulation(self, s0: float, draws: int):
        timesteps = self.tau * 252
        vec_ST = []
        for i in range(draws):
            shat = s0
            for j in range(timesteps):
                next_dwt = self.get_next_step()
                shat += next_dwt
            vec_ST.append()(shat)
        return vec_ST

    @abc.abstractmethod
    def get_next_step(self) -> float:
        pass


class BSSimulate(Simulate):

    def __init__(self, r: float, q: float, tau: float, sigma: float):
        super().__init__(r, q, tau)
        self.sigma = sigma

    def get_next_step(self) -> float:
```

```
33          print("generate next BS step")
34          return 0.0
```

5.0.10 Binary Search Algorithm

A generic implementation of a binary search algorithm in Python can be found in the following code:

```
1  def binary_search(a, x):
2      first = 0
3      last = len(a) - 1
4      while first <= last:
5          mid = (first + last) // 2
6          if a[mid] < x:
7              first = mid + 1
8          elif a[mid] > x:
9              last = mid - 1
10         else:
11             return mid
12     return -1
```

5.0.11 Selection Sort

A selection sort algorithm is very easy to implement via a nested for loop, as shown below:

```
1  def selection_sort(arr):
2      n = len(arr)
3      for i in range(n):
4          min_loc, min_val = i, arr[i]
5          for j in range(i + 1, n):
6              if arr[j] < min_val:
7                  min_loc, min_val = j, arr[j]
8          arr[i], arr[min_loc] = arr[min_loc], arr[i]
9
10     return arr
```

5.0.12 Insertion Sort

The following code shows how one might implement an insertion sort algorithm:

```
1  def insertion_sort(arr):
2      n = len(arr)
3      for i in range(n):
4          val = arr[i]
5          j = i - 1
6          while j >= 0 and arr[j] > val:
7              arr[j + 1] = arr[j]
8              j -= 1
9          arr[j + 1] = val
10
11     return arr
```

5.0.13 Bubble Sort

A bubble sort can be implemented using the code below:

```
def bubble_sort(arr):
    n = len(arr)

    has_swap = False
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                has_swap = True
        if not has_swap:
            break

    return arr
```

5.0.14 Merge Sort

The following code shows how a merge sort can be implemented in Python:

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L, R = [0] * n1, [0] * n2
    for i in range(n1):
        L[i] = arr[l + i]
    for j in range(n2):
        R[j] = arr[m + 1 + j]
    i, j, k = 0, 0, l
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1


def merge_sort(arr, l, r):
    if l < r:
        m = (l + r) // 2
        merge_sort(arr, l, m)
        merge_sort(arr, m + 1, r)
        merge(arr, l, m ,r)
```

```
34      return arr
35
36  rand_arr = np.random.rand(10, 1)
37  arr = merge_sort(rand_arr, 0, len(rand_arr)-1)
```

EXERCISES

5.1 You are given a DataFrame with historical closing price data for an equity index. Write a Python function that takes this DataFrame as an input and computes rolling overlapping returns for the equity index. This function should take the period length of the return as an input.

5.2 You are given two DataFrames which contain historical price data for two different ETF's. Unfortunately, the dates in the two data frames don't match exactly. Write the Python code necessary to merge these two data frames, returning only dates where there are prices for both ETF's.

5.3 Consider the following python code:

```
1       class Position:
2       def __init__(self, shrs, px):
3       self.shares = shrs
4       self.market_price = px
5
6       p1 = Position(100, 1)
7       p2 = Position(100, 1)
8       test = p1 is p2
9       test2 = (p1.shares == p2.shares)
10
```

a. What will be the value of **test** after the program runs?
   i. True
   ii. False

b. What will be the value of **test2** after the program runs?
   i. True
   ii. False

5.4 What will be the output of the following code?

```
1       x = 100
2       pos = Position(100, 1)
3
4       class Position:
5       def __init__(self, shrs, px):
6       self.shares = shrs
7       self.market_price = px
8
9       def multiplyNumByFive(xx):
10      xx = 5.0*xx
```

```
11
12      def multiplyObjByFive(obj):
13      obj.shares = 5.0*obj.shares
14
15      multiplyNumByFive(x)
16      multiplyObjByFive(pos)
17
18      print(x, pos.shares)
19
```

a. 100 500

b. 100 100

c. 500 500

d. 500 100

e. The program will produce an exception

5.5 Consider the following Python code:

```
1       import numpy as np
2
3       tuple1 = (1,2,3)
4       tuple2 = (4,5,6)
5       tple = tuple1 + tuple2
6       print(tple)
7
8       list1 = [1,2,3]
9       list2 = [4,5,6]
10      lst = list1 + list2
11      print(lst)
12
13      arr1 = np.array(list1)
14      arr2 = np.array(list2)
15      arr = arr1 + arr2
16      print(arr)
17
```

What will be printed to the console when this application runs? What will happen if the following code is appended to the end?

```
1       tple[0] = 7
2       lst[0] = 7
3       arr[0] = 7
4
5       print(tple)
6       print(lst)
7       print(arr)
8
```

a. Consider the following Python class definition:

```
class BasePosition:
def __init__(self, shrs, long):
self.shares = shrs
self.isLong = long

def printPos(self):
print("calling printPos from Base")
print(self.shares)
print(self.isLong)

```

What will be the result of the following code?

```
basePos1 = BasePosition(100, 1)
basePos2 = BasePosition(100, 1)

eql1 = (basePos1 == basePos2)
eql2 = (basePos1 is basePos2)

print(eql1)
print(eql2)

```

b. Write a function that would modify the behavior of == comparisons. What technique are you using?

c. Suppose the parameter self was removed from the printPos function. What impact would this have on the printPos function? Would the code still run? If not, write an updated printPos function that would run and print "calling printPos from Base without self".

d. Write the code for a class that inherits from BasePosition. The derived class should have an additional attribute named *multiplier*. Implement the constructor as well as the printPos function. The constructor should call the constructor of the base class.

5.6 a. What is the factory pattern, and in what context should it be used?

b. Using pseudocode, write a sketch of the code needed to implement the factory pattern.

5.7 Using Python or C++, implement an insertion sort that uses a binary search. Discuss the strengths and weaknesses of this algorithm.

5.8 What is the template method? What object-oriented programming concepts does it leverage? Using the template method, write a class structure that efficiently handles pricing Lookback and European options via simulation from the Black-Scholes model.

5.9 a. You are given two pandas Series with historical closing price data for two equity indices. The equity indices are for different geographical regions and

therefore may have different holidays. Write a Python function that merges the two data frames without losing data for any dates.

b. Write a python function that calculates rolling overlapping annualized returns and rolling overlapping annualized volatilities for the merged data frame that you constructed above. Join the index rolling return and volatility series to the price DataFrame that you created above. This function should take the period length of the return and volatility calculations as an input.

c. You are given an additional pandas Series identifying the market regime. Write a Python function that joins this column to the dataframe created above and calculates the conditional return and volatility by market regime. Make sure that your code is generic enough to handle different numbers of market regimes.

5.10 a. Using Python create an abstract base class named OptionsPricer. The abstract base class should contain a constructor, a member function named price, as well as any attributes you believe are appropriate.

b. Create two derived classes that inherit from your OptionsPricer class, EuropeanOptionPricer and LookbackOptionPricer. These classes should assume that a simulated path for the underlying asset is provided to them.

c. Create an analogous class structure for a stochastic process consisting of a base class and Bachelier and Black-Scholes derived classes. Each class in the hierarchy should have a simulation method that implements the appropriate SDE and returns a simulated path that can be read by the OptionsPricer class hierarchy.
In addition, implement the appropriate operator to compare whether the simulated paths in each instance of a class are the same, greater than, or less than.

d. Write a generic global function that takes an option type and stochastic process, as well as any additional parameters you need (i.e. strike, expiry) and dynamically returns the price of the correct option using the desired model.

CHAPTER 6

Working with Financial Datasets

6.0.1 Coding Example: Downloading Data from Yahoo Finance

In this coding example, we show how to download Yahoo Finance price data for a given ticker via the Python API:

```
1 from pandas_datareader import data
2 df_aapl = data.get_data_yahoo('AAPL', start='2020-03-01', end='
    2021-02-28')
```

6.1 BASICS OF SQL QUERYING

Once we have connected to a given database, the natural next step is to be begin to execute a series of SQL statements against it. SQL is a powerful language that enables us to retrieve and aggregate data from multiple tables. SQL can also be used to modify data in a database by removing data, adding data, or updating existing data. In this book we cover only the preliminary topics related to the SQL language. For more information, interested readers should consult [5].

The SQL language has four main types of commands that help us read and modify data, which are briefly summarized here:

- **SELECT**: retrieve rows from a set of tables.
- **UPDATE**: modify rows in a table.
- **INSERT**: add rows to a table.
- **DELETE**: remove rows from a table.

In general, select queries will be the most common types of queries that we write, as they enable us to access data from the database and feed it into our models. An important part of writing SQL statements is being able to retrieve data from the database in a manner that is accessible to our model or application. Generally, once the select statement is run, the results are stored in a Python data frame, as

in the example above, and then used in broader calculations. The other three types of statements, INSERT, UPDATE and DELETE statements, enable us to modify our database rather than retrieve data. These statements may be useful in our data collection process if we choose to aggregate data from external sources and store it in a single centralized database, as is best practice in most production environments.

As a note to the reader, the SQL programming language varies slightly between database engines, such as SQL Server, Oracle and PostreSQL. In this text we focus on the syntax employed by SQL Server. Those who use other databases may need to tweak the syntax of their queries from those presented in the text, however, these differences are almost always minor. Readers are encouraged to consult the documentation for the specific database engine that they are working with for more information.

6.1.1 Modifying Data via Insert, Update and Delete Statements

In the following example, we show examples of insert, update and delete statements:

```
1  INSERT INTO prices (ticker, close_price)
2  VALUES ('MSFT', 100.0);
3
4  UPDATE prices SET close_price = 105.0
5  WHERE ticker = 'MSFT';
6
7  DELETE FROM prices
8  WHERE ticker = 'MSFT';
```

In the code above, the first SQL statement inserts a row into a table named prices, the second modifies all prices for a given ticker, again in the prices table, and the final statement deletes all rows for a given ticker. It should be noted that while insert statements are generally seen as safe operations, because they can easily be undone if anything unexpected occurs, this is not true for UPDATE and DELETE statements. Once we have deleted all rows from a table, or set all prices to the same value in a given table, the process of restoring the original data will be much more involved. Because of this, UPDATE and DELETE commands are considered risky and should be executed carefully. It is best practice to always run the UPDATE or DELETE query initially as a SELECT statement to ensure that the number of rows and content of the rows is what we expect. This should help catch situations where we would have accidentally updated or deleted an entire table, instead of a specific set of rows.

6.1.2 Retrieving Data via Select Statements

As previously mentioned, we can also use SQL queries to retrieve the data that we need from a set of underlying tables. This will be a critical tool in our toolkit when it comes to data collection. There are a handful of core components in a select statement which help us identify which tables we want to include in our query, which columns we want to return, and what conditions we want on the query. The *SELECT* clause defines the columns from our underlying tables that we would like returned in our

query. Next, the *FROM* clause defines the table that we are referencing. In future examples we will show how to include multiple tables in a single query using a *JOIN*. Finally, a *WHERE* clause restricts the rows that are selected from the table or tables based on one or more conditions.

To see a select statement at work, let's start with the following query that contains only a SELECT and FROM clause:

```
1  SELECT ticker, price
2  FROM Equity_Prices
```

This query would return the ticker and price columns for **ALL** rows in the Equity_Prices table. We may instead want to restrict to only rows from the current day, for example, which we could do using the following WHERE clause:

```
1  SELECT ticker, price
2  FROM Equity_Prices
3  WHERE quote_date = 'today'
```

WHERE clauses such as the one above can include simple equality comparisons, but can also be arbitrarily complex. For example, we could use the results from another query in a WHERE clause, in which case that query would be referred to as a subquery.

The next relevant clause in a select statement is an *ORDER BY* clause, which can be used to sort the results of a query by one or more columns.

```
1  SELECT ticker, shares, mkt_value
2  FROM positions
3  WHERE quote_date = 'today'
4  ORDER BY mkt_value DESC
```

Order by clauses can reference the column name, or can reference the order it appears in the SELECT statement. [1] Specifically, this query will show all positions for a given quote_date, and will display them in order from highest market value to lowest. The default sort order is ascending. The **DESC** keyword will sort the results in descending order.

6.1.3 Retrieving Data from Multiple Tables via JOINs

In most cases, we will need to retrieve data from more than one table in our queries. In order to accomplish this, we can utilize the *JOIN* keyword with the *FROM* clause. JOIN's are a powerful component of SQL programming and enable us to define how we merge the tables in a select statement. As an example, if we have two tables, one for positions and one for prices, then it will be a very common task to want to select data from both table simultaneously. When doing so, we need to know which positions rows match with which rows in the prices table. In this example, we might define the matching criteria to include the date and the ticker that the position and price correspond to. The JOIN clause is where we are able to specify this relationship, as shown in the example query below:

[1] Along these lines, we could replace mkt_value in the ORDER BY clause with a 3 and the result would be identical.

```
1  SELECT px.ticker, price, shares
2  FROM prices px
3  JOIN positions pos ON px.ticker = pos.ticker
4  AND px.date = pos.date
```

In this query we can see that the JOIN keyword is used to specify how the prices and positions tables are merged in the query and in particular the content after the ON clause defines the JOIN columns. In this example we are specifying that rows with matching tickers and dates in the positions and prices tables will be matched.

Also note that in this query an **alias** is used for the table names in the FROM clause. The aliases are defined as px and pos respectively, and can generally be specified immediately following the table name in the FROM clause. This alias can be created as it is done in the above query, or can be specified with an optional AS keyword. When an alias is specified, that alias can then be used in other parts of the query, such as in the query above where the alias is used in the SELECT clause and to define the JOIN columns. If we had not specified an alias for the first column in the select statement, ticker, then the query would no longer work as there are multiple columns named ticker in the query and without that prefix the interpreter would not know which column to select.

The JOIN used in the above example is referred to as an inner join, meaning that this query will only return rows where the ticker and date exists in both the prices and positions tables. This means that if a row is only present in the prices table, but has no corresponding positions row, then it will not be included in the query. This is often desired but in some cases we may want to return all rows in the a given table, such as the prices table, even if there is no row in the second table, in this case positions. In SQL we can accomplish this by specifying a different JOIN type, as we explore next.

SQL supports the following main types of JOIN's [2]:

- INNER: Return only rows that exist in both tables.
- LEFT: Return all rows in the left table regardless of whether they exist in the right table.
- RIGHT: Return all rows in the right table regardless of whether they exist in the left table.
- FULL: Return all rows in if they exist in either the left or right tables.

The default JOIN type in SQL is an INNER JOIN, meaning in cases where there is no keyword preceding the JOIN keyword, such as in the example above, an INNER JOIN is performed. If, in the previous example we wanted to return all prices rows whether or not a corresponding positions row existed, we could specify the JOIN as a LEFT JOIN rather than an INNER JOIN. If we were to use a RIGHT JOIN, then all rows in the positions table would be returned, regardless of whether a corresponding prices row exists.

[2]Note that the four types of JOIN's listed above are the most commonly used ones. There are also other types of JOIN's, such as CROSS JOIN and SELF JOIN.

6.1.4 Aggregating Data via a Group By Clause

Lastly, we explore how we can use SQL queries to aggregate data. When doing this, we can utilize the *GROUP BY* clause in a select statement. SQL has a set of aggregate functions that we can perform in our GROUP BY clauses, such as SUM, COUNT, MAX and MIN. The list of columns in the **GROUP BY** clause define how the results will be aggregated. We can choose to group by one or more columns, depending on the specific query we are creating.

To see how aggregate functions in SQL work, consider the following simple example of an aggregate function without a corresponding GROUP BY clause:

```
1  SELECT SUM(shares)
2  FROM positions
```

In this case we are able to aggregate the shares column, however, it aggregates the shares for all rows in the table. In many cases, we won't want a signal aggregate value but instead will want an aggregate value for a subset of groups, for example by ticker, by date, or by another column. The group by clause enables us to embed this logic in our queries. In particular, if we were to add a GROUP BY clause to the above query, it would then return the sum of the shares column for each group. An example of this can be found in the next query:

```
1  SELECT ticker, SUM(shares)
2  FROM positions
3  GROUP BY ticker
```

In this case we are grouping by ticker, meaning that the query will return a row for each ticker in the positions table. The query will return two columns, the first of which will contain the ticker, and the second of which will contain our aggregate, that is, the sum of the shares for that specific ticker.

If we wants to restrict that groups that are returned in a query that uses a GROUP BY clause, we can use the *HAVING* clause. A HAVING clause restricts the groups returned in a query. This contrasts with a WHERE clause which restricts the rows that are used in the underlying calculation, rather than the groups that are displayed after the aggregation is performed.

The following example sample SQL statement shows how we can utilize a HAVING clause to modify the groups that are returned:

```
1  SELECT ticker, SUM(shares) as shrs
2  FROM positions
3  GROUP BY ticker
4  HAVING shrs > 0
```

In particular, in this example we used the HAVING clause to only show long-only positions, that is, cases where the total positions is greater than zero.

6.1.5 Coding Example: Filling Missing Data with Python Data Frames

The following example shows how we can use the functionality embedded in Python's Data Frames to fill missing data via interpolation or filling forward:

```
df = pd.DataFrame(data={'date': pd.date_range(start='2021-2-19', end=
    '2021-2-26'),
  'price': [78, 73, np.nan, 75, 79, 83, np.nan, 78]})
df.loc[:, 'price'] = df.loc[:, 'price'].interpolate(method='linear')
      # interpolation
df.fillna(method='ffill') # forward fill
```

6.1.6 Coding Example: Filling Missing ETF Data via Regression

In this coding example, we show how regression techniques can be used to fill in (hypothetically) missing data for the XLK technology ETF based on market returns in the S&P 500 ETF, SPY. In this exercise we consider the case of having historical data from Jan 1, 2016 to February 28, 2021, but also suppose that all entries for XLK in January 2021 are missing. As we detailed in the prior section, we fit a linear regression model to estimate the coefficient, or beta of XLK to the S&P 500. It should again be emphasized that this approach assumes a significant amout of co-movement, or correlation, in the returns in the two ETFs to be successful. Finally, once we have calculate the parameter, β, then we used this beta to infer the returns of XLK for all days in January 2021. This is detailed in the code below:

```
from pandas_datareader import data
import statsmodels.api as sm

df = data.get_data_yahoo(['XLK', 'SPY'], start='2016-01-01', end='
    2021-02-28')['Adj Close']
df = df.pct_change().dropna()
df_missing = df['2021-01-01':'2021-01-31']

# For backtesting or prediction, only historical data is used
df_boot1 = df[df.index.min():'2020-12-31']
linear_reg1 = sm.OLS(df_boot1['XLK'], df_boot1['SPY']).fit()
beta_hat1 = float(linear_reg1.params)
xlk_boot1 = df_missing['SPY'] * beta_hat1

# For risk management purposes, the whole time period can be used (
    except the missing period)
df_boot2 = pd.concat([df_boot1, df['2021-02-01':'2021-02-28']])
linear_reg2 = sm.OLS(df_boot2['XLK'], df_boot2['SPY']).fit()
beta_hat2 = float(linear_reg2.params)
xlk_boot2 = df_missing['SPY'] * beta_hat2
```

The reader should notice that for different applications of this technique, different time periods may be used to calculate the β. When doing backtesting or making predictions, it is important to use historical data only because we do not want to include any information in the future. That is, we want to ensure that we are not looking ahead. For risk management purposes, however, it may be better to use all available data. This is because risk management calculations are less sensitive to look-ahead bias, and the additional data can help us recover a more stable estimate of the coefficient β.

6.1.7 Coding Example: Using Bootstrapping to Create Synthetic History for a set of Indices

In this coding example, we use the historical data of SPY, AAPL, GOOG to show how to generate a paths of synthetic returns via bootstrapping techniques:

```
from pandas_datareader import data
import random

df_price = data.get_data_yahoo(['SPY', 'AAPL', 'GOOG'], start='
    2020-03-01', end='2021-02-28')['Adj Close']
df_ret = df_price.pct_change().dropna()
N = df_ret.shape[0]
m = 10    # length of the missing data period
boot_index = random.sample(range(N), m)
df_boot = df_ret.iloc[boot_index].set_index(pd.date_range(start='
    2021-03-01', end='2021-03-12', freq='B'))
```

As a note, it is theoretically possible that the number of missing datapoints, or the length of our bootstrapped paths could exceed the length of the historical dataset. These situations may lead to a great deal of sampling however, as the lack of data makes it hard to infer the empirical distribution from the observations. As a result, bootstrapping may not work well in these situations. Instead, it will be much more effective when the historical dataset is large, increasing our confidence in our estimates of the empirical distribution.

6.1.8 Coding Example: Outlier Detection

In this coding example, we show how we can implement the outlier detection techniques discussed in this section in Python:

```
# plotting
df['ticker'].plot()

# standard deviation
mean = df['ticker'].mean()
std = df['ticker'].std()
suspected_outliers = df[(df['ticker']<mean-3*std) | (df['ticker']>
    mean+3*std)]

# density analysis
import seaborn as sns
import statsmodels.api as sm
sns.displot(df['ticker'], kde=True)  # histogram & density
sm.qqplot(df['ticker'], line ='q')  # QQ plot against the standard
    normal distribution
df['ticker'].skew()  # skewness
df['ticker'].kurtosis()  # kurtosis

# KNN
from sklearn.neighbors import NearestNeighbors
model = NearestNeighbors(n_neighbors = 3)  # n_neighbors is subject
    to change
model.fit(df['ticker'].values)
```

```
distances, indexes = model.kneighbors(df['ticker'].values)
distances = distances.mean(axis=1)
```

EXERCISES

6.1 Consider the following set of ETF's:

Ticker	Description
SPY	US Equity
TLT	20+ Year Treasuries
GOVT	All Maturity Treasuries
DBC	Commodities
HYG	High Yield
EEM	Emerging Market Equity
EAFE	Europe and East Asia Equity
AGG	US AGG
IAGG	International AGG
EMB	EM Debt
ACWI	All Country Equities
IWM	US Small Cap Equities

a. Download data from yahoo finance for the sector ETFs above.

b. Download data for the Fama-French factors from Ken-French's website.

c. Check the yahoo finance data for splits and other anomalies.

d. Analyze both dataset for outliers using the methods discussed in this chapter. Identify any data points that you believe might be suspicious.

e. Regress the ETF returns against the Fama-French Factors. Which sectors have the highest exposure to the Fama-French factors?

f. Using the regression above, plot the intercepts in the regression. Which intercept coefficients are biggest (smallest)? What conclusions can draw based on the intercept?

6.2 You are given a data set of sector ETF data which is missing multiple year-long periods for several ETFs. Describe the best approach to handle this missing data and explain what impact filling the missing data has on the distribution of the data as well as the assumptions implicit in your method. Write a piece of code that will find all such gaps in the data and fill them using the method described.

6.3 Write a function that will check the following options data for arbitrage:

How would you propose any arbitrage opportunities be handled?

6.4 Download data for a set of 20 tickers for 20 years. Delete one year's worth of data for each ticker (but not the same year). Use interpolation, regression, and k-means clustering to fill in the missing data. Calculate the distribution of the returns before and after the missing data was removed and replaced. Comment on how each method impacts the empirical distribution.

6.5 Describe the option portfolio that you would buy or sell if you observe that options prices are concave with respect to strike at strike K. Show why this portfolio leads to an arbitrage opportunity.

6.6 Download data for the following ETFs using the data source of your choice:

- UVXY
- VXX
- SVXY

Write an algorithm for finding outliers for each of the three ETFs and comment on whether any outliers you find are legitimate data points.

CHAPTER 7

Model Validation

Here is an example showing the implementation of a unit test by constructing a class instance using inheritance of the **unittest.TestCase** class.

```
import unittest
import numpy as np

class TestRoot(unittest.TestCase):
    def test_root(self):
        # np.roots([a,b,c]) solves the function ax^2+bx+c=0
        roots = list(np.roots([1,1,-2]))
        my_answer = [-2.0,1.0]
        self.assertEqual(my_answer, roots)

unittest.main()
```

As a note to the reader, in order for this unittest.main function call to run, the above code needs to be run as a Python file or from the Python console. Alternatively, if the reader is using jupyter Notebook then they can simply add the following arguments to the unittest.main call:

```
  unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

7.1 UNIT TESTS IN PRACTICE: VALIDATING A SIMULATION ALGORITHM

In this section we provide a coding example that might be used as part of a model validation process of a simple simulation algorithm:

```
import unittest
import numpy as np

def stock_price_simu(S0, r, sigma, t, N, n):
    '''
    dSt = r * dt + sigma * dWt
    N: # simulations, n: # periods
    St_list: terminal value of each path
    dt: time interval
    '''
    St_list = np.empty(N)
    dt = t/n
```

```
13      for i in range(N):
14          dwt = np.random.normal(loc=0,scale=np.math.sqrt(dt),size=n)
15          dSt = r*dt + sigma*dwt
16          St = S0 + sum(dSt)
17          St_list[i] = St
18      stock_mean = np.mean(St_list)
19      stock_var = np.var(St_list)
20      return St_list, stock_mean, stock_var
21
22  class TestSimulation(unittest.TestCase):
23
24      # the setUp function prepares the framework for the test
25      def setUp(self):
26          self.S0 = 20
27          self.r = 0.05
28          self.sigma = 0.5
29          self.t = 1
30          self.N = 1000
31          self.n = 100
32          self.St_list, self.stock_mean, self.stock_var =
        stock_price_simu(self.S0, self.r, self.sigma, self.t, self.N, self
        .n)
33
34      def test_valid_t(self):
35          self.assertTrue(self.t>0)
36          self.assertFalse(self.t<=0)
37
38      def test_valid_sigma(self):
39          self.assertTrue(self.sigma>0)
40          self.assertFalse(self.sigma<=0)
41
42      def test_valid_N(self):
43          self.assertTrue(self.N>0)
44          self.assertFalse(self.N<=0)
45
46      def test_valid_n(self):
47          self.assertTrue(self.n>0)
48          self.assertFalse(self.n<=0)
49
50  unittest.main()
```

Where the reader is encouraged to refer back to section 7 and recall the appropriate method(s) for calling the unittest.main function.

EXERCISES

7.1 List five assumptions of the Black-Scholes model

7.2 List three unit tests that you would add to a piece of code that downloads, parses and cleans options data.

7.3 List three ways to simplify a problem to be more tractable so that we would know the solution.

7.4 Consider the following implemented model. Conduct an independent code review and describe how you would find all bugs, identify all model assumptions and determine if the model is properly implemented.

7.5 Using the unittest module add unit tests to the building blocks of a model. Describe your criteria for where to add unit tests.

II

Options Modeling

CHAPTER 8

Stochastic Models

8.0.1 Coding Example: Black-Scholes Formula

In this coding example we implement the Black-Scholes formula in Python for a European call option:

```
import numpy as np
from scipy.stats import norm

def bs_call(S_0,K,T,sigma,r):
    d_1 = (np.log(S_0/K) + T*(r + (sigma**2)/2))/(sigma*np.sqrt(T))
    d_2 = d_1 - sigma*np.sqrt(T)
    return S_0*norm.cdf(d_1) - K*np.exp(-r*T)*norm.cdf(d_2)

print(bs_call(S_0 = 100, K = 105, T = 0.5, sigma = 0.3, r = 0.02))
#6.779490734346545
```

8.0.2 Coding Example: Bachelier Pricing Formula

In this coding example we show how to price call options under the Bachelier model in Python:

```
import numpy as np
from scipy.stats import norm

def bachelier_call(S_0,K,T,sigma,r):
    d_plus = (S_0*np.exp(r*T) - K)/(sigma*np.sqrt(T))
    return np.exp(-r*T)*sigma*np.sqrt(T)*(d_plus*norm.cdf(d_plus) +
    norm.pdf(d_plus))

print(bachelier_call(S_0 = 100, K = 105, T = 0.5, sigma = 30, r =
    0.02))
#6.549163351317259
```

8.0.3 Coding Example: Option Pricing under the CEV Model

In this coding example we show how the CEV model can be used to price a call option in Python:

```
from scipy.stats import ncx2

def cev_call(S_0,K,T,sigma,beta,r):
    v = 1/(2*(1-beta))
    x_1 = 4*(v**2)*(K**(1/v))/((sigma**2) * T)
    x_2 = 4*(v**2)*((S_0*np.exp(r*T))**(1/v))/((sigma**2) * T)
    kappa_1 = 2*v + 2
    kappa_2 = 2*v
    lambda_1 = x_2
    lambda_2 = x_1
    return np.exp(-r*T)*((S_0*np.exp(r*T)*(1-ncx2.cdf(x_1,kappa_1,
    lambda_1))) - K*ncx2.cdf(x_2,kappa_2,lambda_2))

print(cev_call(S_0 = 100, K = 100, T = 0.5, sigma =4, r = 0.02, beta
     = 0.5))
#11.676110446148732
```

8.0.4 Coding Example: Option Prices in the SABR Model

In the following coding example we detail how to calculate an approximate normal volatility using the asymptotic formula, given a set of SABR parameters:

```
def sabr_normal_vol(S_0,K,T,sigma_0,alpha,beta,rho):
    c = lambda x: x**beta
    c_prime = lambda x: beta*(x**(beta-1))
    c_prime_prime = lambda x: beta*(beta-1)*(x**(beta-2))
    S_mid = (S_0 + K)/2
    gamma_1 = c_prime(S_mid)/c(S_mid)
    gamma_2 = c_prime_prime(S_mid)/c(S_mid)
    zeta = alpha*(S_0**(1-beta) - K**(1-beta))/(sigma_0 * (1-beta))
    epsilon = T*(alpha**2)
    delta = np.log( (np.sqrt(1 - 2*rho*zeta + zeta**2) + zeta - rho)
    /(1-rho) )

    factor = alpha*(S_0 - K)/(delta)
    term_1 = ((2*gamma_2 - gamma_1**2)/24)* (sigma_0*c(S_mid)  /
    alpha)**2
    term_2 = rho*gamma_1*sigma_0*c(S_mid)/(4*alpha)
    term_3 = (2-3*(rho**2))/24
    return factor*(1 + epsilon*(term_1 + term_2 + term_3))

def sabr_call(S_0,K,T,sigma_0,r,alpha,beta,rho):
    assert(S_0 != K)
    vol = sabr_normal_vol(S_0,K,T,sigma_0,alpha,beta,rho)
    return bachelier_call(S_0,K,T,vol,r)
```

8.0.5 Coding Example: Local Volatility Model

In this coding example, we show how the local volatility model developed by Dupire can be implemented in Python given a surface of available option prices and strikes:

```
#given price_surface where columns are different strikes and rows are
     different expiries
```

```
2
3 # expiry_derivatives
4 expiry_derivatives = ((price_surface[2:,:] - price_surface[:-2,:]) /
      (expiries[2:] - expiries[:-2]).reshape(-1,1))[:,:-2]
5
6 # strike second derivative
7 strike_first_derivatives = (price_surface[:,1:] - price_surface
      [:,:-1]) / (strikes[1:]-strikes[:-1])
8 strike_second_derivatives = ((strike_first_derivatives[:,1:] -
      strike_first_derivatives[:,:-1]) / (strikes[1:-1]-strikes[:-2]))
      [1:-1,]
9
10 variance_surface = expiry_derivatives/ (0.5* (strikes[:-2]**2)*
      strike_second_derivatives)
11 vol_surface = np.sqrt(variance_surface)
```

It should be emphasized that this code assumes unique option prices are available at all strikes, and that a correspondingly fine grid of prices and strikes are provided to the function. The reader is encouraged to try this methodology in different market contexts, as well as with data of different quality in order to analyze how the solution is impacted.

EXERCISES

8.1 Derive the Black-Scholes formula for a European put option

8.2 Derive the Bachelier formula for a European call option

8.3 Use L'Hopital's rule to estimate the normal volatility in the SABR model for an at-the-money option for a given set of parameters.

8.4 Find a parameterization of the Heston, SABR and Variance Gamma models that enables us to recover the Black-Scholes model.

8.5 Download implied volatility data for VIX and calculate the local volatility function, $\sigma(K, T)$. Discuss how you used interpolation to compute the relevant partial derivatives and comment on the challenges of applying the local volatility model in practice.

8.6 Consider the following set of SABR parameters:

- $\alpha = 0.25$
- $\beta = 0.5$
- $\rho = 0.75$
- $\sigma_0 = 0.075$

a. Apply the SABR asymptotic formula to obtain a price for a three-month call option of an asset whose price is 100 and whose strike price is 105. Assume that the risk-free rate is 1.25% and the asset pays no dividens.

b. Holding all other parameters constant, re-price the option at two-months to expiry, one-month to expiry and at expiry. Use these calculations to comment on this options carry profile.

c. Now re-price the option after a shift up and down in the asset price to 99 and 101 respectively. Consider the three-month, two-month and one-month cases above. At which point does this shift make the most money? At which point does it lose the least? Why?

8.7 Consider the Variance Gamma model:

a. List the model parameters and describe what they do.

b. Under these dynamics, how do you shift the model parameters such that:
   - Shift the volatility surface up in parallel.
   - Add more negative skewness to the distribution of S_T.
   - Increase the kurtosis of the distribution of S_T.

   The scenarios should specify which parameter(s) you want to shift as well as the direction of the shift.

8.8 Consider the Heston model:

a. List the model parameters and describe what they do.

b. Under these dynamics, how do you shift the model parameters such that:
   - Shift the volatility surface up in parallel.
   - Steepen the volatility term structure (i.e., increase the price of longer expiries relative to shorter ones.)
   - Increase the volatility skew for out-of-the-money puts (i.e. make low strike puts more expensive).
   - Increase the kurtosis of the distribution of S_T.

   The scenarios should specify which parameter(s) you want to shift as well as the direction of the shift.

8.9 Consider the CEV model with $\sigma = 0.2$ and $\beta = 0.5$. Assume that the underlying asset is currently trading at 150, the risk-free rate is 5% and the asset pays 2.8% in dividends.

a. Calculate the price and implied volatilities of three-month options at the following strikes:
   - 140
   - 145
   - 150
   - 155
   - 160

b. Compare these prices to those under the Black-Scholes model with $\sigma = 0.2$. Which one do you expect to be higher?

c. Explain how you would find the equivalent Black-Scholes implied volatility corresponding to the CEV model defined above. Using this $\tilde{\sigma}$ compute prices under the Black-Scholes model. Which model leads to higher prices now?

d. The CEV model above defines the price for any European option for all strikes and all expiries. Use this function to calculate the local volatility function, $\sigma(K, T)$ under the assumption that these dynamics hold.

CHAPTER 9

Options Pricing Techniques for European Options

9.0.1 Coding Example: Pricing a Digital Option via Quadrature Methods

In this coding example we show how we could use quadrature to price a digital option with a specified risk-neutral density:

```
def price_digital_call_quad(S_0,K,r,T,density_func, b, N):
    """ Using left riemann sum"""
    width = (b-K)/N
    nodes = np.arange(K,b,width)
    areas = [width * density_func(node) for node in nodes]
    px = np.exp(-r*T)*sum(areas)
    return px
```

9.0.2 Coding Example: FFT Option Pricing in the Heston Model

In the following coding example we detail how the FFT techniques detailed in this chapter can be used to value European options using the Heston stochastic volatility that was introduced in chapter 8:

```
def heston_characteristic_eqn(u, sigma, k,p,s_0,r,t,theta, v_0):
    lambd = np.sqrt((sigma**2)*((u**2)+1j*u) + (k - 1j*p*sigma*u)**2)
    omega_numerator = np.exp(1j*u*np.log(s_0)+1j*u*(r)*t+(1/(sigma
    **2))*k*theta*t*(k - 1j*p*sigma*u))
    omega_denominator = (np.cosh(0.5*lambd*t) + (1/lambd)*(k - 1j*p*
    sigma*u)*np.sinh(0.5*lambd*t))**((2*k*theta)/(sigma**2))
    phi = (omega_numerator/omega_denominator) * np.exp(-((u**2 + 1j*u
    )*v_0)/(lambd*(1/np.tanh(0.5*lambd*t)) + (k - 1j*p*sigma*u)))
    return phi

def calc_fft_heston_call_prices(alpha, N, delta_v, sigma, k, p, s_0,
    r, t, theta, v_0, K = None):
    #delta is the indicator function
    delta = np.zeros(N)
    delta[0] = 1
    delta_k = (2*np.pi)/(N*delta_v)
    if K == None:
```

```
14          #middle strike is at the money
15          beta = np.log(s_0) - delta_k*N*0.5
16      else:
17          #middle strike is K
18          beta = np.log(K) - delta_k*N*0.5
19      k_list = np.array([(beta +(i-1)*delta_k) for i in range(1,N+1) ])
20      v_list = np.arange(N) * delta_v
21      #building fft input vector
22      x_numerator = np.array( [((2-delta[i])*delta_v)*np.exp(-r*t)  for
         i in range(N)] )
23      x_denominator = np.array( [2 * (alpha + 1j*i) * (alpha + 1j*i +
        1) for i in v_list] )
24      x_exp = np.array( [np.exp(-1j*(beta)*i) for i in v_list] )
25      x_list = (x_numerator/x_denominator)*x_exp* np.array([
        heston_characteristic_eqn(i - 1j*(alpha+1),sigma, k,p,s_0,r,t,
        theta, v_0) for i in v_list])
26      #fft output
27      y_list = np.fft.fft(x_list)
28      #recovering prices
29      prices = np.array( [(1/np.pi) * np.exp(-alpha*(beta +(i-1)*
        delta_k)) * np.real(y_list[i-1]) for i in range(1,N+1)] )
30      return prices, np.exp(k_list)
```

9.0.3 Coding Example: Extracting Implied Volatility using Root Finding

In the following coding example, we show to calculate an implied volatility for a given option in the Black-Scholes model using a one-dimensional root finding algorithm:

```
1  from scipy.optimize import root
2
3  def bs_call(S_0,K,T,sigma,r):
4      d_1 = (np.log(S_0/K) + T*(r + (sigma**2)/2))/(sigma*np.sqrt(T))
5      d_2 = d_1 - sigma*np.sqrt(T)
6      return S_0*norm.cdf(d_1) - K*np.exp(-r*T)*norm.cdf(d_2)
7
8  def get_implied_vol_bs(S_0,K,T,r,observed_px, initial_guess):
9      root_fn = lambda x: bs_call(S_0,K,T,x,r) - observed_px
10     return root(root_fn,initial_guess)['x'][0]
```

9.0.4 Coding Example: Finding the Minimum Variance Portfolio of Two Assets

In this coding example, we show how to use the above equations to find a minumum variance portfolio for two assets. To verify our result, we also show how this same problem can be computed using Python's built-in optimization libraries:

```
1  def min_var_portfolio(sigma_1,sigma_2,rho):
2      numerator = sigma_2**2 - sigma_1*sigma_2*rho
3      denominator = sigma_1**2 + sigma_2**2 - 2*sigma_1*sigma_2*rho
4      w_1 = numerator/denominator
5      return (w_1,1-w_1)
6
7  min_var_portfolio(0.25,0.3,-0.4)
8
```

```
#(0.5647058823529412, 0.43529411764705883)
```

```
from scipy.optimize import minimize

#the function we are minimizing
def portfolio_variance(weights,cov_mat):
    return weights.T@cov_mat@weights

def min_var_portfolio_opt(cov_mat):
    guess = np.array([1/len(cov_mat)]*len(cov_mat)).reshape(-1,1) #
    equal weights guess
    cons = {'type': 'eq', 'fun': lambda x:  np.sum(x)-1} #sum of
    weights equal to 1
    return minimize(portfolio_variance,x0=guess,args=(cov_mat),
    constraints=cons,tol=1e-8, method = 'SLSQP')['x']

correlation = -0.4
sigma_1 = 0.25
sigma_2 = 0.3
cov_mat = np.array([[sigma_1**2,correlation*sigma_1*sigma_2],[
    correlation*sigma_1*sigma_2,sigma_2**2]])

min_var_portfolio_opt(cov_mat)

#array([0.56470588, 0.43529412])
```

Readers should obtain the same results regardless of which methodology they choose.

9.0.5 Coding Example: Calibrating a Volatility Surface

In this coding example we detail how a volatility surface can be calibrated to the CEV model using the formulas presented in chapter 8:

```
from scipy.optimize import minimize

def sum_squares_cev(beta,S_0,sigma,r,call_prices, strikes, expiries):
    sum = 0
    for price, strike, expiry in zip(call_prices, strikes, expiries):
      sum += (price - cev_call(S_0,strike,expiry,sigma,r,beta))**2
    return sum

def find_beta_sigma_cev(S_0,r,call_prices, strikes, expiries, guess =
     [0.9,0.4],bounds = ((0.001,None),(0.001,None)),tol=1e-10):
    """
    call_prices, strikes, and expiries are arrays of equal length
    with each index corresponding to one option
    For example, if the first elements of each array are 10, 100, and
     1 respectively, this corresponds to an option
    with price 10, strike 100, and 1 year to expiry
    """
    #first element is beta, second element is sigma
    unique_expiries = np.unique(expiries)
    calibrated_beta_sigma = {} #this is where the results will go for
     each expiry
```

```
18      for expiry in unique_expiries:
19        #how many times this expiry appears
20        expiries_list = expiry*np.ones(np.sum(expiries == expiry))
21        #where it appears
22        indices = np.where(expiries == expiry)[0]
23        #what are the strikes for this expiry
24        strikes_for_expiry = np.array(strikes)[indices]
25        #what are the market prices for this expiry
26        prices_for_expiry = np.array(call_prices)[indices]
27        #the function to be minimized
28        opt_func = lambda x: sum_squares_cev(x[0],S_0,x[1],r,
      prices_for_expiry, strikes_for_expiry, expiries_list)
29        #calibrated values
30        beta, sigma = minimize(opt_func,guess, bounds = bounds ,method
      = 'SLSQP', tol=tol)['x']
31        #saving the results
32        calibrated_beta_sigma[expiry] = [beta, sigma]
33      return calibrated_beta_sigma
```

The reader should notice that a separate set of parameters are calibrated per expiry in the code. This is because the model parameterization of the CEV mdoel is unable to fit the term structure of volatility but is able to fit many observed volatility skews.

EXERCISES

9.1 **Option Pricing via FFT Techniques** The Heston Model is defined by the following system of stochastic differential equations:

$$\begin{aligned} dS_t &= rS_t\,dt + \sqrt{\nu_t}S_t\,dW_t^1 \\ d\nu_t &= \kappa(\theta - \nu_t)\,dt + \sigma\sqrt{\nu_t}\,dW_t^2 \\ Cov(dW_t^1, dW_t^2) &= \rho\,dt \end{aligned}$$

Assume the risk-free rate is 2%, the initial asset price is 250 and that the asset pays no dividends.

a. **Exploring FFT Technique Parameters** Consider a European Call Option with strike 250 expiring in six months.
Additionally, assume you know that the parameters of the Heston Model are:

$$\begin{aligned} \sigma &= 0.2 \\ \nu_0 &= 0.08 \\ \kappa &= 0.7 \\ \rho &= -0.4 \\ \theta &= 0.1 \end{aligned}$$

i. Calculate the price of the European Call option with many values for the damping factor, α. What values of α seem to lead to the most stable price?

ii. Using the results above, choose a reasonable value of α and calculate the price of the same European Call with various values of N and Δk (or equivalently N and B). Comment on what values seem to lead to the most accurate prices, and the efficiency of each parameterization.

iii. Calculate the price of a European Call with strike 260 using various values of N and Δk (or N and B). Do the same sets of values for N, B and Δk produce the best results? Comment on any differences that arise.

b. **Exploring Heston Parameters** Assume the risk-free rate is 2.5%, the initial asset price is 150 and that the asset pays no dividends.

$$\begin{aligned} \sigma &= 0.4 \\ \nu_0 &= 0.09 \\ \kappa &= 0.5 \\ \rho &= 0.25 \\ \theta &= 0.12 \end{aligned}$$

i. Using these parameters, calculate Heston Model prices for three-month options at a range of strikes and extract the implied volatilities for each strike. Plot the implied volatility $\sigma(K)$ as a function of strike.

ii. Use the FFT pricing technique to obtain prices of 150 strike calls at many expiries. Extract the implied volatility for each and plot the term structure of volatility by plotting time to expiry on the x-axis and implied volatility on the y-axis.

iii. Holding all other parameters constant, vary each of the model parameters and plot the updated volatility skews and term structures. Comment on the impact that each parameter has on the skew and term structure.

9.2 Consider a one-year European call option with strike 125 and starting asset price of 100. Assume that the risk-free rate is 1.3% and the asset pays dividends at a rate of 2.75%.

a. Value this option using FFT pricing techniques under Merton's jump diffusion model with the following parameters:

- $\sigma = 0.25$
- $\lambda = 0.05$
- $\alpha = -0.025$
- $\gamma = 0.1$

Try many values of α as well as N and B. Comment on what the optimal values for these parameters, as well as Δk and $\Delta\nu$ appear to be.

b. Comment on the impact of the jump component. Do they raise or lower the price of the option compared to the equivalent process without jumps? Why?

c. Find a parameterization of Merton's Jump Diffusion model that simplifies to a tractable model with a closed form solution. What model is it? Can you use this to validate your model implementation? Why or why not?

d. Calculate the sensitivity of this option price to each model parameter: $\{\sigma, \lambda, \alpha, \gamma\}$ by shifting each parameter and re-valuing the option. Explain why you think those relationships exist. Would these relationships hold for put options as well?

9.3 Recall that the Heston Model is defined by the following system of SDE's:

$$\begin{aligned} dS_t &= rS_t\,dt + \sqrt{\nu_t}S_t\,dW_t^1 \\ d\nu_t &= \kappa(\theta - \nu_t)\,dt + \sigma\sqrt{\nu_t}\,dW_t^2 \\ Cov(dW_t^1, dW_t^2) &= \rho\,dt \end{aligned} \tag{P9.1}$$

See the spreadsheet SPY_data.csv for options data. $r = 1.5\%$, $q = 1.77\%$, $S_0 = 267.15$.

Consider the given market prices and the following equal weight least squares minimization function:

$$\vec{p}_{\min} = \min_{\vec{p}} \left\{ \sum_{\tau,K} \left(\tilde{c}(\tau, K, \vec{p}) - c_{\tau,K}\right)^2 \right\} \tag{P9.2}$$

where $\tilde{c}(\tau, K, \vec{p})$ is the FFT based model price of a call option with expiry τ and strike K.

a. Check the option prices for arbitrage. Are there arbitrage opportunities at the mid? How about after accounting for the bid-ask spread? Remove any arbitrage violations from the data.

b. Using the FFT Pricing code from your last homework, find the values of κ, θ, σ, ρ and ν_0 that minimize the equal weight least squared pricing error. You may choose the starting point and upper and lower bounds of the optimization. You may also choose whether to calibrate to calls, puts, or some combination of the two.
Note also that you are given data for multiple expiries, each of which should use the same parameter set, but will require a separate call to the FFT algorithm.

c. Try several starting points and several values for the upper and lower bounds of your parameters. Does the optimal set of parameters change? If so, what does this tell you about the stability of your calibration algorithm?

d. Instead of applying an equal weight to each option consider the following function which makes the weights inversely proportional to the quoted bid-

ask spread:

$$\omega_{\tau,K} = \frac{1}{c_{\tau,K,\text{ask}} - c_{\tau,K,,\text{bid}}} \tag{P9.3}$$

$$\vec{p}_{\min} = \min_{\vec{p}} \left\{ \sum_{\tau,K} \omega_{\tau,K} \left(\tilde{c}(\tau, K, \vec{p}) - c_{\tau,K} \right)^2 \right\} \tag{P9.4}$$

where as before $\tilde{c}(\tau, K, \vec{p})$ is the FFT based model price of a call option with expiry τ, strike K. Repeat the calibration with this objective function and comment on how this weighting affects the optimal parameters.

9.4 Consider the process of calibrating a Variance Gamma model to market data using FFT:

a. Download options data for JC Penney (ticker: JCP) at 1m, 3m, 6m and 1y expiries. Check the data for arbitrage and resolve any arbitrage opportunities using the technique of your choice.

b. Using the data above, extract and plot an implied volatility smile for each expiry.

c. For each expiry, calibrate the optimal set of Variance Gamma parameters $\{\theta, \sigma, \nu\}$. Comment on how you formulated your objective function and how you chose the weights to apply to each option.

d. Try several values for the technique parameters as well as variations of the optimization formulation (i.e. modify the weighting scheme, starting guess and upper and lower bounds for each parameter). What do you think is the optimal setup?

e. Using the calibrated parameters, calculate the price of at-the-money digital put options at 1m, 3m, 6m and 1y expiries. Comment on the difference in pricing of the digitals as expiry increases. How does it relate to the shape of the volatility smiles you plotted above?

f. Re-price all calibrated instruments after the following shifts to the Variance Gamma model parameters:
   - $\hat{\sigma} = \sigma + 0.05$
   - $\hat{\theta} = \theta - 0.25$
   - $\hat{\nu} = \nu + 0.025$

   Plot the updated implied volatility surfaces for 1,3,6 and 12 months. Comment on the impact of this shift to the volatility smile.

9.5 Consider the following risk-neutral pricing formula for a European Digital Call:

$$c_0 = \tilde{\mathbb{E}} \left[e^{-rT} 1_{\{S_T > K\}} \right]$$

a. Derive the pricing formula that can be used to price European Digital Call options using the characteristic function of a stochastic process.

b. Describe two approaches for calculating the final integral and the trade-offs between them.

c. Discuss how you would use this pricing formula to create an approximation for an American Digital put option. The payoff for an American Digital put option can be written as:

$$p_0 = \tilde{\mathbb{E}}\left[e^{-rT} 1_{\{M_T < K\}}\right]$$

where M_T is the minimum value for the asset over the observation period. Comment on what factors would go into the accuracy of your approximation and whether the estimate would be reliable.

d. Consider the following strike spacing grid used in the FFT method:

$$k_m = \beta + (m-1)\Delta k \quad \text{for} \quad m = 1, \ldots, N = 2^n,$$

where β is defined as:

$$\beta = \log S_0$$

Is this a valid strike spacing grid for the FFT technique? Will the at-the-money strike, S_0 fall on the grid? If so, at which value of m will the at-the-money-strike be?

9.6 When using the FFT method, we define the log-strikes as

$$k_m = \beta + (m-1)\Delta k \quad \text{for} \quad m = 1, \ldots, N = 2^n.$$

We have previously defined β as follows:

$$\beta = \log S_0 - \frac{N\Delta k}{2}$$

a. Prove that the at-the-money strike will fall exactly on our strike grid. Find the index m that will correspond to the at-the-money strike.

b. If we are calculating the price of an out-of-the-money option using the β above, do we know whether the strike be on the grid? If not, how will we infer the price from the set of strikes that we have?

c. Find another β that ensures that a given out-of-the-money strike, K^*, will appear exactly on the grid. Show the index that will correspond to your desired strike.

d. Under which of these two choices of β in (9.6b.) and (9.6c.) would you use a higher N? Why?

e. What β can we use so that our desired strike appears last in the array? What do you think of this as a strike grid?

CHAPTER 10

Options Pricing Techniques for Exotic Options

10.0.1 Coding Example: Simulation from the Heston Model

In this example, we show how to simulate the price and payoff of a European call option using the Heston model:

```
def heston_simu(S0, K, r, T, v0, sig, k, theta, rho, n, N):

    dt = T/N
    mean = (0,0)
    cov = [[dt,rho*dt],[rho*dt,dt]]
    st_vec = np.empty(n)
    vt_vec = np.empty(n)
    payoff_vec = np.empty(n)
    price_vec = np.empty(n)

    for i in range(n):

        x = np.random.multivariate_normal(mean,cov,(N,1))
        st = S0
        vt = v0

        for j in range(N):
            dst = r*st*dt + np.sqrt(vt)*st*x[j][0][0]
            dvt = k*(theta-vt)*dt + sig*np.sqrt(vt)*x[j][0][1]
            st += dst
            vt += dvt
            if vt<0:
                vt = -vt

        payoff_vec[i] = max(st-K,0)
        price_vec[i] = payoff_vec[i]*np.exp(-r*T)
        st_vec[i] = st
        vt_vec[i] = vt

    plt.hist(st_vec,bins=100)
    plt.title("Terminal stock price distribution")
    plt.xlabel("stock price")
```

```
33      plt.ylabel("frequncy per " + str(n) + " trials")
34      plt.show()
35
36      return price_vec
```

Note that in this coding example we rely on the reflection approach to handling potential negative volatilities.

10.0.2 Coding Example: Simulation from the Variance Gamma

In this example, we show how to simulate the price of an asset using the Variance Gamma model:

```
1  def var_gamma(sigma, v, theta, S0, T, r, q, n, N):
2
3      dt = T/N
4      w = np.log(1-theta*v-0.5*v*sigma**2)/v
5      price_vec = np.empty(n)
6
7      for i in range(n):
8
9          lns0 = np.log(S0)
10         xt = 0
11
12         for j in range(N):
13
14             gamma = np.random.gamma(dt/v, v)
15             z = np.random.normal(0,1)
16             xt += theta*gamma + sigma*np.sqrt(gamma)*z
17
18             Tj = dt*(j+1)
19             lnst = lns0 + (r-q+w)*Tj + xt
20
21         price = np.exp(lnst)
22         price_vec[i] = price
23
24     return price_vec
```

10.0.3 Coding Example: American Options Pricing via the Black-Scholes PDE

In this example, we calculate the price of an American call by solving the Black-Scholes PDE using the explicit discretization scheme:

```
1  def solve_bs_pde(s0,smax,k,T,N,M,sig,r,p):
2
3      ht = T/N
4      hs = smax/M
5      t = np.arange(0, T+ht, ht)
6      s = np.arange(0, smax+hs, hs)
7
8      d = 1-(sig**2)*(s**2)*ht/(hs**2)-r*ht
9      l = 0.5*(sig**2)*(s**2)*ht/(hs**2)-r*s*ht/(2*hs)
10     u = 0.5*(sig**2)*(s**2)*ht/(hs**2)+r*s*ht/(2*hs)
11
```

```
    A = np.matrix(np.zeros((M-1,M-1)))
    diag = d[1:]
    upperDiag = u[1:M-1]
    lowerDiag = l[2:M]
    for i in range(len(upperDiag)):
        A[i,i+1] = upperDiag[i]
        A[i+1,i] = lowerDiag[i]
    for i in range(M-1):
        A[i,i] = diag[i]
    vec_eigenvalue = np.linalg.eigvals(A)

    b = u[M-1]*(s[M]-k*np.exp(-r*(T-t)))
    #ba = u[M-1]*(s[M]-k)

    diff = s-k
    diff[diff<0]=0
    ter_c = np.matrix(diff[1:M]).T
    cont_val = ter_c

    for i in range(N, 1, -1):
        bb = np.append(np.zeros(M-2),b[i]).reshape(M-1,1)
        exercise_val =  np.maximum(s[1:-1]-k, np.zeros(M-1)).reshape(
    M-1,1)
        cont_val = A @ cont_val + bb
        # exercise if exercise value exceeds continuation value
        vec_c = np.maximum(cont_val, exercise_val)

    return vec_c
```

The **solve_bs_pde** function returns the update matrix A with its eigenvalues, and the price vector. The price vector denotes the option price at time 0 with different underlying asset prices, so we can plug in the current value of the asset price to obtain our estimated model price via a PDE approximation.

EXERCISES

10.1 Match the market instrument on the right with the most appropriate pricing technique on the left.

Quadrature	European Call with Heston dynamics
FFT	American Option with Heston dynamics
PDE	Knock-out options with Variance Gamma dynamics
Simulation	Call option contingent on European trigger with Black-Scholes dynamics

NOTE: Please use each item only once.

10.2 Consider the Black-Scholes SDE:

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

You are asked to use this SDE to write a simulation algorithm to price an

American upside one touch option. Recall that the payoff of an American one touch is defined as:

$$c_0 = \tilde{\mathbb{E}}\left[e^{-rT} 1_{M_T > K}\right]$$

where M_T is the maximum value of the asset over the period.

a. (2.5 points) List the set of parameters that you will have in your algorithm and describe their meaning.

b. (7 Points) Write a piece of pseudo-code that you can use to simulate from the given stochastic process.

c. (7 Points) Write a piece of pseudo-code that will define the payoff function for your exotic option.

d. (3.5 points) Describe three unit tests that you would create to ensure that your model prices are correct.

10.3 Consider the SDE for an OU process:

$$dS_t = \kappa(\theta - S_t)dt + \sigma dW_t$$

You are asked to use this SDE to write a simulation algorithm to price an American up-and-out call option. Recall that the price of an American up-and-out call option is defined as:

$$c_0 = \tilde{\mathbb{E}}\left[e^{-rT}(S_T - K)^+ 1_{M_T < B}\right]$$

Where M_T is the maximum value of the asset over the period and B is the barrier level.

a. (2 points) List the set of model parameters and describe their meaning. Explain how to simplify this model to the Bachelier model.

b. (2 points) Comment on whether you think this would be a good model for an equity index? How about a rates or volatility model? Why or why not?

c. (3.5 Points) Write a piece of code in Python or C++ that you can use to simulate a path from the given stochastic process.

d. (3.5 Points) Write a piece of code in Python or C++ that will define the price function of your exotic option, given a simulated path.

e. (2 points) Name the simulation parameters that you must choose and describe how you would set them optimally.

f. (2 points) Describe three unit tests that you would create to ensure that your model prices are correct.

10.4 Let S_t follow the Black-Scholes process:

$$dS_t = (r - q)S_t dt + \sigma S_t dW_t \tag{P10.1}$$

where r is the risk-free rate and q is the dividend rate. Then the Black-Scholes PDE for a derivative security c is:

$$\frac{\partial c}{\partial t} + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 c}{\partial s^2} + (r - q)s\frac{\partial c}{\partial s} - rc = 0 \tag{P10.2}$$

a. Write a discretization of this equation which would lead to an explicit scheme.

b. Write the boundary conditions for an American Digital Put which pays \$1 if the asset touches a downside barrier K, at any time.

c. Discuss how you select the time and space grid.

d. If you know the price of the European Digital Put how could you create a rough estimate of the American Digital Put price? What assumptions impact the accuracy of this approximation?

10.5 Consider a generic function of two variables $f(x, y)$. We would like to use finite differences to approximate the derivatives of $f(x, y)$.

a. Write the Taylor Series expansion of $f(x, y)$ near (x_0, y_0).

b. Compute an approximation to the cross partial derivative $\frac{\partial^2 f}{\partial x \partial y}$

c. What is the error of this approximation?

10.6 For $\lambda_n, \lambda_p > 0$ with $\lambda_n \neq \lambda_p$, consider the following probability density function for a random variable X:

$$f(x) = \begin{cases} \frac{1}{2}\lambda_n e^{\lambda_n x}, & x < 0 \\ \frac{1}{2}\lambda_p e^{-\lambda_p x}, & x \geq 0 \end{cases}$$

a. Write pseudo-code that implements the inverse transform method to generate a sample of X.

b. Write pseudo-code that uses one sample of a Bernoulli random variable and one sample of an exponential random variable to generate a sample of X.

c. Which method is faster for generating a sample of X? Why?

10.7 Consider an Asian option whose risk-neutral pricing formula can be written as:

$$c_0 = \tilde{\mathbb{E}}\left[e^{-rT}(\bar{S} - K)^+\right]$$

where $\bar{S}$ is the arithmetic average value of S over the observation period, r is equal to 0 and the asset pays no dividends. Additionally, assume that the

current value of the asset is 100. Consider a three-month at-the-money spot Asian option ($K = 100$) on this asset.

Also assume that the market follows the dynamics in the Heston model which are defined as:

$$\begin{aligned} dS_t &= rS_t\,dt + \sigma_t S_t\,dW_t^1 \\ d\sigma_t^2 &= \kappa(\theta - \sigma_t^2)\,dt + \xi\sigma_t\,dW_t^2 \\ Cov(dW_t^1, dW_t^2) &= \rho\,dt \end{aligned}$$

a. (3 points) List all parameters in the Heston model and describe intuitively what they represent.

b. (6 points) Using the Heston model and the Euler discretization scheme for Monte-Carlo, compute an approximation for the Asian option assuming the following parameters:

$$\begin{aligned} \sigma_0 &= 0.05 \\ \kappa &= 1 \\ \theta &= 0.1 \\ \rho &= -0.5 \\ \xi &= 0.25 \\ \Delta t &= \frac{1}{252} \end{aligned}$$

c. (3 points) Run your approximation with many different numbers of simlated paths. Estimate the number of draws that are needed for the price of the option to converge. Comment on the difference between this and the theoretical convergence rate.

d. (5 points) Consider Z and $-\frac{1}{2}Z$ as antithetic variables, where Z is a standard normal random variable. Calculate the mean and variance of an estimator using these pairs of variables. Is this approach unbiased? Does it reduce the variance? By how much?

e. (3 points) Describe what you think would be an ideal control variate for the Asian option priced above. Justify your answer with either a theoretical or empirical argument.

f. (5 points) Reprice the Asian option above using the control variate of your choice. Does the simulated price converge faster? Why or why not?

10.8 The SDE for the CEV model for $\beta > 0$ is:

$$dS_t = r\,S_t dt + \sigma S_t^{\beta}\,dW_t$$

a. (3 Points) Write the PDE that would result from using the CEV model.

b. (3 Points) What are the parameters in the model and what is their interpretation?

c. (2 Points) What parameters would lead this model to match the dynamics of the Black-Scholes model?

d. (6 Points) Write a discretization of this equation which would lead to an implicit scheme.

e. (6 Points) Discuss how you select the time and space grid.

10.9 Suppose that the underlying security SPY evolves according to the Heston model. That is, we know its dynamics are defined by the following system of SDEs:

$$\begin{aligned} dS_t &= (r-q)S_t\,dt + \sqrt{\nu_t}S_t\,dW_t^1 \\ d\nu_t &= \kappa(\theta-\nu_t)\,dt + \sigma\sqrt{\nu_t}\,dW_t^2 \\ Cov(dW_t^1, dW_t^2) &= \rho\,dt \end{aligned} \tag{P10.3}$$

You know that the last closing price for SPY was 282. You also know that the dividend yield for SPY is 1.77% and the corresponding risk-free rate is 1.5%.

Using this information, you want to build a simulation algorithm to price a knock-out option on SPY, where the payoff is a European call option contingent on the option not being knocked out, and the knock-out is an upside barrier that is continuously monitored. We will refer to this as an **up-and-out call**.

This payoff can be written as:

$$c_0 = \mathbb{E}\left[(S_T - K_1)^+ 1_{\{M_T<K_2\}}\right] \tag{P10.4}$$

where M_T is the maximum value of S over the observation period, and $K_1 < K_2$ are the strikes of the European call and the knock-out trigger respectively.

a. Find a set of Heston parameters that you believe govern the dynamics of SPY. You may use results from a previous Homework, do this via a new calibration, or some other empirical process. Explain how you got these and why you think they are reasonable.

b. Choose a discretization for the Heston SDE. In particular, choose the time spacing, ΔT as well as the number of simulated paths, N. Explain why you think these choices will lead to an accurate result.

c. Write a simulation algorithm to price a European call with strike $K = 285$ and time to expiry $T = 1$. Calculate the price of this European call using FFT and comment on the difference in price.

d. Update your simulation algorithm to price an up-and-out call with $T = 1$, $K_1 = 285$ and $K_2 = 315$. Try this for several values of N. How many do you need to get an accurate price?

e. Re-price the up-and-out call using the European call as a control variate. Try this for several values of N. Does this converge faster than before?

f. Calculate the delta, vega and gamma of both a European call and our up-and-out option. What variance reduction techniques can we apply to this calculation? Comment on the difference in the Greeks between the European call and the up-and-out option. Do they make sense to you?

10.10 Suppose that the underlying security SPY evolves according to standard geometric brownian motion.

Then its derivatives obey the Black-Scholes equation:

$$\frac{\partial c}{\partial t} + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 c}{\partial s^2} + rs\frac{\partial c}{\partial s} - rc = 0 \tag{P10.5}$$

Use SPY's closing price of 380.

We are going to find the price of an American call spread with the right to exercise early. Suppose the two strikes of the call spread are $K_1 = 385$ and $K_2 = 390$ and that the risk-free rate is 1.5%. Consider a three-month American call spread.

a. Explain why this instrument is not the same as being long an American call with strike 385 and short an American call with strike 390, both with expiry in three-months.

b. Let's assume that we are not able to find σ by calibrating to the European call spread price and must find it by other means. Find a way to pick the σ, explain why you chose this method, and then find the σ.

c. Set up an explicit Euler discretization of (P10.5). You will need to make decisions about the choice of $s_{\max}$, h_s, h_t, etc. Please explain how you arrived at each of these choices.

d. Let A be the update matrix that you created in the previous step. Find out its eigenvalues and check their absolute values.

e. Apply your discretization scheme to find today's price of the call spread *without* the right of early exercise. The scheme will produce a whole vector of prices at time 0. Explain how you chose the one for today's price.

f. Modify your code in the previous step to calculate the price of the call spread *with* the right of early exercise. What is the price?

g. Calculate the early exercise premium as the difference between the American and European call spreads. Is it reasonable?

10.11 Consider a one-year **fixed strike lookback option** which enables the buyer to choose the point of exercise for the option at its expiry.

Recall that the dynamics of the Black-Scholes model can be written as:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t \tag{P10.6}$$

And that each Brownian motion increment dW_t is normally distributed with mean zero and variance Δt. Assume that $r = 0$, $S_0 = 100$ and $\sigma = 0.25$.

a. Generate a series of normally distributed random numbers and use these to generate simulated paths for the underlying asset. What is the mean and variance of the terminal value of these paths? Does it appear to be consistent with the underlying dynamics?

b. Calculate the payoff of a European put option with strike 100 along all simulated paths. Make a histogram of the payoffs for the European option. What is the mean and standard deviation of the payoffs?

c. Calculate a simulation approximation to the price of a European put option by taking the average discounted payoff across all paths.

d. Compare the price of the European put option obtained via simulation to the price you obtain using the Black-Scholes formula. Comment on any difference between the two prices.

e. Calculate the payoff of a fixed strike lookback put option with stike 100 along all simulated path. (HINT: The option holder should exercise at the minimum price along each simulated path). Calculate the simulation price for the lookback option by averaging the discounted payoffs.

f. Calculate the premium that the buyer is charged for the extra optionality embedded in the lookback. When would this premium be highest? Lowest? Can it ever be negative?

g. Try a few different values of σ and comment on what happens to the price of the European, the Lookback option and the spread/premium between the two.

10.12 Consider again a one-year **fixed strike lookback option** which enables the buyer to choose the point of exercise for the option at its expiry. The **Bachelier model** can be written as:

$$dS_t = r\,dt + \sigma\,dW_t \tag{P10.7}$$

Assume that $r = 0$, $S_0 = 100$ and $\sigma = 10.0$.

a. Generate a series of normally distributed random numbers and use these to generate simulated paths for the underlying asset.

b. Plot a histogram of the ending values of the asset price along the simulated paths. Are the ending values of your simulated paths normally distributed? Check using your favorite normality test.

c. Calculate a simulation approximation to the price of a Lookback put option with strike 100 under the Bachelier model. Compare the price of the lookback option in the Bachelier model to the Black-Scholes model price obtained in HW1.

d. Calculate the delta of the lookback option using finite differences as discussed in class. That is:

$$\Delta \approx \frac{c_0(S_0 + \epsilon) - c_0(S_0 - \epsilon)}{2\epsilon} \tag{P10.8}$$

Try for several values of ϵ and plot the calculated Δ against the choice of ϵ. Comment on what you think is the optimal value of ϵ and what values lead to the largest amounts of error.

CHAPTER 11

Greeks and Options Trading

11.0.1 Coding Example: Calculating Delta and Gamma

This coding example shows how to calculate the delta and gamma of European call and put options, in the Black-Scholes model, in Python:

```
s0, K, r, T, sigma = 100, 100, 0.05, 1, 0.2
d1 = (np.log(s0/K) + (r + 0.5*sigma**2) * T) / (sigma * np.sqrt(T))
delta_c = norm.cdf(d1)  # 0.6368306511756191
delta_p = -norm.cdf(-d1)  # -0.3631693488243809
```

11.0.2 Coding Example: Black-Scholes Theta

In this coding example, we detail how to calculate the theta of a European call and put option using the equations in (**??**):

```
s0, K, r, T, sigma = 100, 100, 0.05, 1, 0.2
d1 = (np.log(s0/K) + (r + 0.5*sigma**2) * T) / (sigma * np.sqrt(T))
d2 = d1 - sigma * np.sqrt(T)
theta_c = -s0*norm.pdf(d1)*sigma / (2*np.sqrt(T)) - r*K*np.exp(-r*T)*
    norm.cdf(d2)  # -6.414027546438197
theta_p = -s0*norm.pdf(d1)*sigma / (2*np.sqrt(T)) + r*K*np.exp(-r*T)*
    norm.cdf(-d2)  # -1.657880423934626
```

11.0.3 Coding Example: Estimation of Lookback Option Greeks via Finite Differences

In this coding example we show how finite difference can be used compute first-order Greeks for a lookback option, where the lookback option itself is valued using the simulation techniques discussed in chapter 10:

```
def bs_simulate(s0, t, r, sigma, n, N):
    dt = 1 / n
    dwt = np.random.normal(0, np.sqrt(dt), (N, int(n * t)))
    s = np.zeros((N, dwt.shape[1] + 1))
    s[:, 0] = s0
    for i in range(1, s.shape[1]):
        s[:, i] = s[:, i-1] + s[:, i-1] * r * \
            dt + s[:, i-1] * sigma * dwt[:, i-1]
    return s
```

```
10
11 def lookback_price(k, t, is_call, paths):
12     s_m = np.max(paths, axis=1) if is_call else np.min(paths, axis=1)
13     return np.fmax(s_m - k, 0) if is_call else np.fmax(k - s_m, 0)
14
15 def c(s0=s0, sigma=sigma, t=t, r=r, k=k, n=n, N=N):
16     np.random.seed(0)  # to get the same random numbers every time
17     paths = bs_simulate(s0, t, r, sigma, n, N)
18     look_prc = np.exp(-r * t) * np.mean(lookback_price(k, t, True,
       paths))
19     return look_prc
20
21 epsilon = 1e-6  # different epsilon gets different gamma
22 delta = (c(s0+epsilon) - c(s0-epsilon)) / (2*epsilon)  #
       1.1608790693173887
23 vega = (c(s0, sigma+epsilon) - c(s0, sigma-epsilon)) / (2*epsilon)  #
        106.27836798526857
```

11.0.4 Coding Example: Smile Adjusted Greeks

In this coding example we calculate a smile adjusted delta under the Heston Model via finite differences. To accomplish this, we shift the underlying asset as is standard in delta calculations. However, we also shift the initial level of volatility, σ_0, as the processes are correlated and therefore a change in the asset implies a certain expected change to the volatility or variance process. The following code details our implementation of smile-adjusted deltas in the Heston model:

```
1  def heston_simulate(s0, t, r, nu0, kappa, theta, xi, rho, n, N):
2      dt = 1 / n
3      dwt = np.random.multivariate_normal(
4          mean=(0, 0),
5          cov=[[dt, rho*dt], [rho*dt, dt]],
6          size=(N, int(n * t)))
7      s, nu = np.zeros((N, dwt.shape[1] + 1)), np.zeros((N, dwt.shape
       [1] + 1))
8      s[:, 0], nu[:, 0] = s0, nu0
9      for i in range(1, s.shape[1]):
10         s[:, i] = s[:, i-1] + s[:, i-1] * r * dt \
11             + s[:, i-1] * nu[:, i-1]**0.5 * dwt[:, i-1, 0]
12         nu[:, i] = nu[:, i-1] + kappa*(theta-nu[:, i-1]) \
13             + xi*nu[:, i-1]**0.5*dwt[:, i-1, 1]
14     s = s
15     nu = nu
16     return s, nu
17
18 def euro_payoffs(k, t, is_call, paths):
19     st = paths[:, -1]
20     return np.fmax(st - k, 0) if is_call else np.fmax(k - st, 0)
21
22 def c(s0=s0, nu0=nu0, theta=theta, t=t, r=r, kappa=kappa, xi=xi, rho=
       rho, n=n, N=N, k=k):
23     np.random.seed(0)  # to get the same random numbers every time
```

```
    pathsS0, pathsNu = heston_simulate(s0, t, r, nu0, kappa, theta,
    xi, rho, n, N)
    euro_prc = np.exp(-r * t) * np.mean(euro_payoffs(k, t, True,
    pathsS0))
    return euro_prc

dnu0_delta = (xi * rho * epsilon) / s0

epsilon = 1e-6
smile_delta = (c(s0=s0+epsilon, nu0=nu0+dnu0_delta) - c(s0=s0-epsilon
    , nu0=nu0-dnu0_delta)) / (2*epsilon)  # 0.6303207156221902
```

EXERCISES

11.1 Derive the formula for the delta of a call option under the Bachelier model using the closed form pricing formula in (**??**).

11.2 Derive the formula for theta under the Black-Scholes model as defined in (**??**).

11.3 Option traders often say that when buying options we get gamma at the expense of theta. Explain what they mean.

11.4 Describe the options structure that you would trade on the S&P 500 if you had the following objectives:

a. Provide insurance in the event of a large sell-off

b. Harvest a premium between implied and realized volatility

c. Sell volatility in a defined downside manner

d. Isolate over-priced downside skew

e. Take advantage of a mis-priced CDF

11.5 Consider the following table of options:

Description
Long at-the-money one-year call option
Short 25 delta three-month put option
Long one-month at-the-money straddle
Six-month risk reversal
Long three-month at-the-money to 25 delta call spread

Assume $S_0 = 100.0$, $\sigma = 0.2$ and $r = q = 0.0$.

Calculate a standard set of Black-Scholes Greeks for each row in the table, including delta, gamma, theta and vega and comment on the relative properties of the options.

11.6 **Hedging Under Heston Model:** Consider a 3 month European call with strike 275 on the same underlying asset.

a. Calculate the price of this 3-month 110 strike call using your calibrated Heston parameters and your FFT pricing algorithm.

b. Extract the Black-Scholes implied volatility for this option.

c. Calculate this option's Heston delta using finite differences. That is, calculate a first order central difference by shifting the asset price, leaving all other parameters constant and re-calculating the FFT based Heston model price at each value of S_0.
   i. Compare this delta to the delta for this option in the Black-Scholes model. Are they different, and if so why? If they are different, which do you think is better and why? Which would you use for hedging?
   ii. How many shares of the asset do you need to ensure that a portfolio that is long one unit of the call and short x units of the underlying is delta neutral?

d. Calculate the vega of this option numerically via the following steps:
   i. Calculate the Heston vega using finite differences. To do this, shift θ and ν_0 by the same amount and calculate a first order central difference leaving all other parameters constant and re-calculating the FFT based Heston model price at each value of θ and ν_0.
   ii. Compare this vega to the vega for this option in the Black-Scholes model. Are they different, and if so why?
   iii. Calculate the Heston vega of an at-the-money straddle (long 1 unit of the 267.15 strike call + long 1 unit of the 267.15 strike put) using finite differences.
   iv. How many units of the straddle do you need in order to ensure that a portfolio that is long one unit of the call and short x units of the straddle is vega neutral?

11.7 **Volatility Experiments:**

a. Download historical data for the S&P using the SPY ETF and for the VIX index, which we will use as a proxy for volatility.

b. Examine both the S&P and the VIX index data for autocorrelation. You may do this using the regression approach you used in the first two homeworks, or via stationarity tests and ARMA models. Do you find evidence of autocorrelation? Which series has more evidence of autocorrelation? Where would you expect more autocorrelation?

c. Calculate the correlation of the S&P and its implied volatility (using VIX as a proxy) on a daily and monthly basis. Is the correlation significant? What implications does this have for an options pricing model, such as the Black-Scholes model?

d. Calculate rolling 90-day correlatons of the S&P and its implied volatility as well. When does the correlation deviate the most from its long-run average?

e. Calculate rolling 90-day realized volatilities in the S&P and compare them to the implied volatility (again using VIX as a proxy). Plot the premium of implied vol. over realized vol. Is the premium generally positive or negative? When is the premium highest? Lowest?

f. Construct a portfolio that buys a 1M at-the-money straddle (long an at-the-money call and long an at-the-money put) every day in your historical period. Use the Black-Scholes model to compute the option prices and use the level of VIX as the implied vol input into the BS formula.

g. Calculate the payoffs of these 1M straddles at expiry (assuming they were held to expiry without any rebalances) by looking at the historical 1M changes in the S&P. Calculate and plot the P&L as well. What is the average P&L?

h. Make a scatter plot of this P&L against the premium between implied and realized volatility. Is there a strong relationship? Would you expect there to be a strong relationship? Why or why not?

11.8 **Implied vs. Realized Volatility:**

a. Download historical data for the following set of ETFs:

ETF	Description
SPY	S&P 500
HYG	High-Yield
TLT	Long Maturity Treasuries
USO	Oil

Calculate the rolling three-month realized volatility for each ETF.

b. Download historical options data for the set of ETFs. If necessary, calculate the implied volatility for each day for an at-the-money, three-month option.

c. Plot this three-month, at-the-money implied volatility against the three-month future realized volatility. What does this imply about being long vs. short option positions?

d. Calculate the implied vs. realized premium and plot this quantity as well. Describe qualitatively the behavior of this premium over time. Is it persistent across all asset classes? Explain any differences in the premium between assets.

e. Comment on the times when realized volatility exceeds implied volatility. Does it seem to correspond to major market events? Does it happen at the same time for each asset? Why?

CHAPTER 12

Extraction of Risk Neutral Densities

12.0.1 Breeden-Litzenberger: Python Coding Example

The following piece of code can be used to implement the Breeden-Litzenberger technique in Python:

```
import math
from scipy.stats import norm
from scipy import interpolate

def breeden_litzenberger(strikes, vols, s_0, rf, tau, dk):
    #use linear interpolation of implied volatility
    vol_interp = interpolate.interp1d(strikes, vols, fill_value="
    extrapolate")

    #loop through arrays and use Breeden-Litzenberger to find Risk
    Neutral Density at each point
    phis = np.full(len(strikes), 0.00)
    for zz, strike_zz in enumerate(strikes):
        px_up = callPx(s_0, strike_zz + dk, rf, vol_interp(strike_zz
    + dk), tau)
        px = callPx(s_0, strike_zz, rf, vol_interp(strike_zz), tau)
        px_dn = callPx(s_0, strike_zz - dk, rf, vol_interp(strike_zz
    - dk), tau)

        numer = (px_up - 2 * px + px_dn)
        denom = (dk * dk)
        phis[zz] = numer / denom

    return phis


s0 = 100.0
r = 0.0
exp_t = 0.25
upper_k = 150.0
lower_k = 50.0
n_points = 10001
```

```
29
30 dk = (upper_k - lower_k) / n_points
31
32 ks = np.linspace(lower_k, upper_k, n_points)
33 ivs = np.full(n_points, 0.2)
34
35 rnd = breeden_litzenberger(ks, ivs, s0, r, exp_t, dk)
36 print(rnd)
```

Listing 12.1 Breeden Litzenberger Code Sample

Note that the callPx function is used in the above coding sample is taken from section **??**.

12.0.2 Coding Example: Pricing a Digital Option

The following code can be used to price a digital option given a risk neutral PDF:

```
1 def digi_call_px(k, rf, tau, nodes, probs, dk):
2     digi_px = 0.0
3     for zz, node_zz in enumerate(nodes):
4         if node_zz >= k:
5             digi_px += math.exp(-rf*tau) * probs[zz] * dk
6
7     return digi_px
```

Listing 12.2 Digital Call Option Price Code Sample

12.0.3 Weighted Monte Carlo: Coding Example

The following coding example shows a simple implementation of Weighted Monte Carlo where the optimization is done on a set of terminal probabilities and where no prior distribution is incorporated:

```
1 import numpy as np
2 import cvxpy as cp
3 import math
4
5 def weighted_mc(call_strikes, impl_vols, s_0, tau, rf, num_nodes,
      lower_k, upper_k):
6     nodes = np.linspace(lower_k, upper_k, num_nodes)
7
8     x = cp.Variable(shape=num_nodes)
9     obj = cp.Maximize(cp.sum(cp.entr(x)))
10
11    constraints = [0.0 <= x, cp.sum(x) == 1, cp.sum(x * nodes) == s0
      * math.exp(rf * tau)]
12
13    threshold = s_0 * 0.005
14    for zz, call_strike_zz in enumerate(call_strikes):
15        payoffs = np.exp(-rf*tau) * np.maximum(nodes - call_strike_zz
      , 0.0)
16        call_prices_zz = callPx(s_0, call_strike_zz, rf, impl_vols[zz
      ], tau)
```

```
17          constraints.append(cp.sum(x * payoffs) <= call_prices_zz +
        threshold)
18          constraints.append(cp.sum(x * payoffs) >= call_prices_zz -
        threshold)
19
20      prob = cp.Problem(obj, constraints)
21      prob.solve(verbose=True)
22
23      return x.value
24
25  s0 = 100.0
26  r = 0.0
27  exp_t = 0.25
28  upper_k = 150.0
29  lower_k = 50.0
30  n_points = 501
31
32  ks = np.linspace(lower_k, upper_k, n_points)
33  ivs = np.full(n_points, 0.2)
34
35  wmc_probs = weighted_mc(ks, ivs, s0, exp_t, r, n_points, lower_k,
        upper_k)
36  print(wmc_probs)
```

Listing 12.3 Weighted Monte Carlo Code Sample

Where again the callPx function referenced in the above code leverages the example in **??**

EXERCISES

12.1 Show how you would extract the risk-neutral density using the Breeden-Litzenberger methodology for a set of put options.

12.2 Derive the price of a digital put option and show its relationship to the CDF of a risk-neutral distribution.

12.3 Consider a standard European call:

$$\begin{aligned} c_0 &= \tilde{\mathbb{E}}\left[e^{-rT}\left(S_T - K\right)^+\right] \\ &= e^{-rT}\int_{-\infty}^{+\infty}\left(S_T - K\right)^+ \phi(S_T)\, dS_T \end{aligned}$$

a. Differentiate the European call payoff function with respect to strike.

b. Calculate $\frac{\partial^2 c_0}{\partial K^2}$ and discuss the relationship between this derivative and the PDF of the risk neutral density.

c. Using finite differences, show the relationship between the CDF of a risk-neutral density and the prices of call spreads with strikes K and $K + \epsilon$ (for sufficiently small ϵ).

d. Describe a set of arbitrage conditions that a set of call spreads with different K's (but the same ϵ) should obey.

12.4 **Implementation of Breeden-Litzenberger:** You are given the following volatility data:

Expiry / Strike	1M	3M
10DP	32.25%	28.36%
25DP	24.73%	21.78%
40DP	20.21%	18.18%
50D	18.24%	16.45%
40DC	15.74%	14.62%
25DC	13.70%	12.56%
10DC	11.48%	10.94%

You also know that the current stock price is 100, the risk-free rate is 0, and the asset pays no dividends.
NOTE: The table of strikes is quoted in terms of Deltas, where the DP rows indicate "X Delta Puts" and the DC rows indicate "X Delta Calls".

a. Using the table of quoted (Black-Scholes) Deltas and volatilities, extract a table of strikes corresponding to each option.

b. Choose an interpolation scheme that defines the volatility function for all strikes, $\sigma(K)$.

c. Extract the risk neutral density for 1 & 3 month options. Comment on the differences between the two distributions. Is it what you would expect?

d. Extract the risk neutral density for 1 & 3 month options using a constant volatility equal to the 50D volatility. Contrast these densities to the densities obtained above.

e. Price the following European Options using the densities you constructed in (12.4c.).
   i. 1M European Digital Put Option with Strike 110.
   ii. 3M European Digital Call Option with Strike 105.
   iii. 2M European Call Option with Strike 100.

12.5 Consider the following risk neutral pricing formula for a European put option:

$$p_0 = \tilde{\mathbb{E}}\left[(K - S)^+\right]$$

For purposes of this problem you may assume that interest rates are equal to zero and thus ignore discounting terms.

a. (5 Points) Construct a butterfly centered on some strike, K^*, using only put options. Please include the strike of each option as well as the units and whether you should be long or short each put.

b. (8 Points) Using the Breeden-Litzenberger technique, derive the relationship between the risk-neutral density of S and put options.

c. (6 points) Why must put option prices be convex with respect to strike in the absence of arbitrage? Suppose you noticed that this condition was violated. Construct a riskless portfolio that would take advantage of this.

d. (6 points) Describe the structure that you would trade if you believe that the market was mispricing the CDF of S. You should assume only European Calls and Puts are traded in your answer.

12.6 Density extraction via Weighted Monte Carlo

a. Download futures and options data for 1M, 3M and 6M Gold futures.

b. Using weighted monte carlo with the Black-Scholes model as a prior calibrate the risk-neutral density.

c. Plot the 1M, 3M and 6M risk-neutral densities and comment on the differences between them.

d. Using bootstrapping of historical returns create an estimate of the physical distribution of Gold. Compare this to the risk-neutral distribution and comment on any features present in the risk-neutral distribution but not in the physical distribution.

12.7 Compare the risk-neutral density of two models (i.e. SABR vs. Heston) with given parameters

12.8 Choose a model and compare the risk-neutral distribution as wel change the parameters.

III

Quant Modelling in Different Markets

CHAPTER 13

Interest Rate Markets

13.0.1 Coding Example: Bond Pricing, Duration & Convexity

In this coding example we detail how to leverage the formulas presented in this chapter, (??) and(??), to compute a bond price, as well as its duration and convexity, in Python code:

```
def bond_price_dur_conv(C,P,y,mat,coupon_frequency):
    B = 0
    duration_sum = 0
    convexity_sum = 0

    total_payments = coupon_frequency * mat
    for payment in range(1,total_payments+1):
        payment_time = payment/coupon_frequency

        B +=  C * math.exp(-y * payment_time)
        duration_sum += C * payment_time* math.exp(-y * payment_time)
        convexity_sum += C * payment_time**2 * math.exp(-y *
    payment_time)

    B += P * math.exp(-y* mat)
    duration_sum += P * mat * math.exp(-y * mat)
    convexity_sum += P * mat**2 * math.exp(-y * mat)

    ###convention is to report duration as a positive number
    duration = -(-1/B * duration_sum)
    convexity = 1/B * convexity_sum

    return B,duration,convexity

bond_price_dur_conv(0.05/2*10000,10000,0.04,5,2)

# (10430.589989789156, 4.498366646086723, 21.596380839001984)
```

13.0.2 Coding Example: Bootstrapping a Yield Curve

In this coding example we detail how a bootstrapping yield curve construction algorithm can be implemented in Python for a set of interest rate swaps:

```
def swaprate(t,smallf):
    s_num = 0
    s_denom = 0

    for i in range(1,t):

        #s_num += discount(0,t_i,smallf)*forward(t_i,t_i + 1,smallf)
        s_num += discount(0,i,smallf) * forward(i,i+1,smallf)
        s_denom += discount(0,i,smallf)

    return s_num/s_denom

def rootfinder(x,t,smallf,df,prev):

    for i in range(1+prev,t):
        smallf[i] = x

    swap = swaprate(t,smallf)
    return swap - df['Swap Rates'][t]/100

#df is a Pandas DataFrame with Swap Rates and corresponding Maturity
    as columns.
#maturity is the longest maturity
def boot_strap(df,maturity):

    smallf = np.zeros(maturity)
    prev = 0

    for k in df['Maturity']:
        result = scipy.optimize.root(rootfinder, 0.01,args = (k,
    smallf,df,prev))
        prev = k

    return smallf
```

The reader will notice that instruments other than swaps, such as Eurodollar futures and FRAs, are not included in this bootstrapping algorithm. This is instead left as an exercise to the reader to extend the coding example to accommodate these instruments.

13.0.3 Coding Example: Cap Pricing via Black's Model

In this coding example we detail how a cap or floor pricing model can be implemented in Pyhon, focusing on Black's model. As we can see in the following sample, the model works by pricing each caplet separately in a for loop which can be done because each caplet has an independent exercise decision:

```
def price_caplet(sigma,t,K,delta,rate):

    f_0 = forward(t,t+delta,rate)

    d1 = (math.log(f_0/K) + 0.5*sigma**2*t)/(sigma * math.sqrt(t))
    d2 = (math.log(f_0/K) - 0.5*sigma**2*t)/(sigma * math.sqrt(t))

```

```
    cap = delta * discount(0,t+delta, rate) * (f_0 * sp.norm.cdf(d1)
    - K*sp.norm.cdf(d2))

    return cap

def price_cap(sigma,delta,start,length,rate,K):

    cap = 0

    for i in range(length/delta):

        t = start+delta*i
        caplet = price_caplet(sigma,t,K,delta,rate)
        cap += caplet

    return cap
```

13.0.4 Coding Example: Calibrating Swaptions via the SABR Model

In this coding example, we show how a SABR model can be leveraged to calibrate a swaption volatility in interest rate markets. The reader should notice that a separate set of model parameters are calibrated for each expiry, and that, as is standard when calibrating a SABR model, the β parameter is fixed and the remaining parameters are optimized for:

```
def asymptotic_normal_vol(T,K,F0,sigma0,alpha,beta,rho):

    Fmid = (F0 + K)/2
    zeta = alpha/(sigma0 * (1-beta)) * (F0 ** (1-beta) - K **(1-beta)
    )
    eps = T*alpha**2
    delta = np.log((np.sqrt(1-2 * rho * zeta + zeta**2) + zeta - rho)
    /(1-rho))
    gamma1 = beta/Fmid
    gamma2 = beta * (beta -1)/Fmid**2

    parta = alpha * (F0-K)/delta
    partb1 = (2*gamma2 - gamma1**2)/24 * (sigma0 * Fmid ** beta /
    alpha)**2
    partb2 = rho * gamma1/4 * sigma0 * Fmid **beta/alpha
    partb3 = (2-3 *rho **2)/24
    partb = (1+(partb1 + partb2 + partb3) * eps)

    return parta*partb

def rootfind_helper(sabr_params,T,K_list,F0,sigma_list):

    sigma0,alpha,rho = sabr_params
    beta = 0.5
    MSE = 0

    for i in range(len(K_list)):

```

```
26          diff = asymptotic_normal_vol(T,K_list[i],F0,sigma0,alpha,beta
    ,rho) - sigma_list[i]
27
28          MSE += diff**2
29
30      return MSE
31
32  def calibrate_sabr(T_list,K_table,F0_list,sigma_table):
33
34      init_guess = [0.1,0.1,-0.1]
35
36      params = np.zeros((len(T_list,3)))
37
38      for T,i in enumerate(T_list):
39
40
41          opt = scipy.minimize(rootfind_helper,init_guess,args = (T,
    K_table[i], F0_list[i],sigma_table[i]), method = 'SLSQP', bounds =
     ((0.01,1.5),(0,1.5),(-1,1)))
42          params[i] = opt.x
43
44      return params
```

13.0.5 Coding Example: Simulation from the Vasicek Model

In this coding example we provide a function for simulating from the Vasicek short rate model:

```
1  def sim_Vasicek(r0, kappa, theta, sigma, T,nper):
2      rate_path = np.zeros(T*nper)
3      rate_path[0] = r0
4
5      dt = 1/nper
6      for i in range(1,T*nper):
7          dWt = np.random.normal(0,np.sqrt(1/nper))
8
9          rate_path[i] = rate_path[i-1] + kappa*(theta - rate_path[i
    -1]) * dt + sigma * dWt
10
11     return rate_path
```

EXERCISES

13.1 Consider the following table of bond yields by maturity:

Maturity(years)	Yield
1	0.025
2	0.026
3	0.027
5	0.03
10	0.035
30	0.04

Note: these are yield to maturities for each bond can be used for all its underlying cashflows. For example, for a 5-year coupon (or zero-coupon) bond, use a constant yield of 3% for all the bonds cashflows.

a. Calculate prices of a zero coupon bond that pays $100 at maturity for each maturity & yield combination. Which price is the highest? Is this reasonable?

b. Calculate the duration of each zero coupon bond, or sensitivity of the bond price to a change in bond yield. What is the relationship between bond prices and bond yields?

c. Calculate prices of coupon bonds that pay $100 at maturity at 3% annually until maturity. Which prices are below $100? Which prices are above? Why?

d. Calculate the duration of each coupon bond using finite differences. Do zero-coupon bonds or coupon bonds have higher duration? Why?

e. Calculate the second derivative of each bond price with respect to yield (commonly known as convexity). Are the second derivatives positive or negative?

f. Consider a portfolio that is long one unit of the 1 year zero-coupon bond, long one unit of the 3 year zero-coupon bond and short two units of the 2 year zero-coupon bond. Calculate the initial value of the portfolio.

g. Calculate the duration of this portfolio. Calculate the convexity of the portfolio as well. Which quantity is bigger?

h. Adjust the number of units of the short position in the two year zero-coupon bond so that the portfolio is duration neutral (leaving the units of the one and three year zero-coupon bonds unchnaged). How many units of the two year zero-coupon bond are required to do this?

i. Suppose you own this adjusted portfolio and rates sell off by 100 basis points (each yield rises by 1%). What happens to the value of your portfolio?

j. Now suppose you own this adjusted portfolio and rates rally by 100 basis points (each yield decreases by 1%). What happens to the value of your portfolio? Is this a portfolio you would want to own? What are the risks of owning this portfolio?

k. Print the cashflows of a 5-year amortizing bond that repays 20% of its principal annually and pays a 3% coupon (annually).

l. Calculate the price and duration of the amortizing bond using finite differences. Comment on the difference between this bond and its zero coupon and coupon equivalents.

13.2 Consider the following table of swap rates:

1Y	2.8438%
2Y	3.060%
3Y	3.126%
4Y	3.144%
5Y	3.150%
7Y	3.169%
10Y	3.210%
30Y	3.237%

a. Extract the constant forward rate for the first year that enables you to match the 1Y market swap rate.

b. Holding this first year forward rate fixed, find the forward rate from one year to two years that enables you to match the two year swap (while also matching the one year).

c. Continue this process and extract piecewise constant forward rates for the entire curve. Comment on the forward rates vs. the swap rates.

d. Compute the fair market, breakeven swap rate of a 15Y swap. That is, find the swap rate that equates the present values of the fixed and floating legs.

e. Compute discount factors. Compute zero rates by finding the constant rate that leads to the calibrated discount factors. Comment on the differences in the zero rates and swap rates.

f. Shift all forward rates up 100 basis points and re-calculate the breakeven swap rates for each benchmark point. Generate a table of new swap rates. Are these rates equivalent to having shifted the swap rates directly?

g. Consider a bearish steepener to the swap rates, that is perform the following shifts on each swap rate:

1Y	+0 bps
2Y	+0 bps
3Y	+0 bps
4Y	+5 bps
5Y	+10 bps
7Y	+15 bps
10Y	+25 bps
30Y	+50 bps

Print the new swap rates.

h. Re-run your bootstrapping procedure with this new curve. Comment on the changes to the forward rates.

i. Consider a bull steepener to the swap rates, that is perform the following shifts on each swap rate:

1Y	-50 bps
2Y	-25 bps
3Y	-15 bps
4Y	-10 bps
5Y	-5 bps
7Y	+0 bps
10Y	+0 bps
30Y	+0 bps

Print the new swap rates.

j. Re-run your bootstrapping procedure with this new curve. Comment on the changes to the forward rates.

13.3 Using the yield curve extracted in the previous problem:

a. Calculate the three-month carry of both payer and receiver swap at each benchmark by revaluing the swaps with expiry shortened by three-months. Assume that swaps rates roll-down the yield curve, meaning the nine-month swap rate in three-months is equal to today's nine-month swap rate..

b. Do payer or receiver swaptions seem to have positive carry? Why?

c. Construct a duration portfolio that is long the highest carry swap and short the lowest carry swap. Describe this portfolio. What are its main risks?

13.4 Consider the following table of normal swaption volatilities and corresponding par swap rates:

Expiry	Tenor	F_0	ATM - 50	ATM - 25	ATM - 5	ATM + 5	ATM + 25	ATM + 50
1Y	5Y	117.45	57.31	51.51	49.28	48.74	41.46	37.33
2Y	5Y	120.60	51.72	46.87	43.09	42.63	38.23	34.55
3Y	5Y	133.03	46.29	44.48	43.61	39.36	35.95	32.55
4Y	5Y	152.05	45.72	41.80	38.92	38.19	34.41	31.15
5Y	5Y	171.85	44.92	40.61	37.69	36.94	33.36	30.21

NOTE: All numbers in the table are reported in bps.

a. Calculate the constant instantaneous forward rate for each swap that will lead to the par swap rates listed. You may use a different instantaneous forward rate for each swap and are not required to go through an entire bootstrapping exercise.

b. Using the rates obtained above, calculate the current annuity value for each swap in the above table.

c. Calculate a table of premiums for each swaption in the table using the Bachelier pricing formula and the annuities computed above.

d. For each option expiry, find the set of SABR parameters that best matches the quoted normal volatilities. Utilize the asymptotic approximation formula to calculate the normal volatility for a given set of SABR parameters and look for a solution that minimizes the distance between market and model volatilities.

e. Comment on the relationship of the calibrated parameters as a function of expiry.

f. Using these calibrated SABR parameters, calculate the price and normal volatility of swaptions with strikes equal to ATM - 75 and ATM + 75.

g. Calculate the equivalent Black volatilities for each option in the table above.

13.5 Consider an investor who buys a 3-year no call 30 year Bermudan swaption and sells a 3y27 swaption at the same, at-the-money strike.

a. Without modeling the Berm, plot an estimated payoff diagram of the underlying structure and describe the trades properties.

b. What happens to the holder of this position if rates rally or sell-off by 100 basis points? Do they make or lose money?

c. Suppose Bermudan swaptions are not available to trade in this rates market. Propose an analogous structure of European swaptions that would mimic the payoff you made above.

13.6 Consider an investor who has a strong macroeconomic view that the yield curve is likely to steepen.

a. Describe the exotic option structure that will most precisely instrument that view.

b. Would higher or lower correlations lead to better pricing for this investor?

c. What is the correlation bet implicit in their view?

d. Suppose exotic options are illiquid and cannot be traded. Construct a portfolio of swaptions that best replicate this payoff. Highlight any key differences between the two structures.

13.7 The dynamics of the Hull-White mode are presented in (**??**). Consider this model with the following parameters:

$$\begin{aligned} \sigma &= 0.25 \\ \kappa &= 1 \\ r_0 &= 0.0118 \end{aligned}$$

a. Download yield curve data and extract a set of zero-coupon bond prices for various maturities.

b. Calibrate the Hull-White drift term, θ_t so that the model matches the current yield curve.

c. Derive the conditional distribution of the short rate, $r(t)$ at time t. Calculate its expected value and variance.

d. Using simulation, generate $N = 10000$ paths for the short rate that extend to time $t = 10$. Discuss the most efficient approach to conducting this simulation.

e. Calculate the value of a 10-year zero-coupon bond along each simulated path. Compare this to the known formula for Zero-Coupon bonds in the Hull-White model.

f. Calculate the value of a call option on a 10-year zero-coupon bond with strike $K = 99$.

g. Re-price this zero-coupon bond option with $\sigma = 0.4$. What happens to the price? Why?

h. Re-price this zero-coupon bond option with $\kappa = 5$. What happens to the price? Why?

13.8 Consider the following set of bonds:

- Five-Year Zero Coupon Bond
- Five-Year Coupon Paying Bond with fixed coupon C
- Five-Year Floating Rate Bond which pays Libor plus a constant spread, S
- Thirty-Year Zero Coupon Bond

a. Download data from treasury.gov or another reputable source and use bootstrapping to extract a piecewise constant instantaneous forward rate process.

b. Calculate the price of each bond listed above. Which is the highest? Lowest? Why?

c. Calculate the duration or sensitivity of each bond to a parallel shift of the instantaneous forward rates by one basis point. Which bonds have the highest and lowest duration? Explain why.

d. Calculate the convexity of each bond to a parallel shift of the instantaneous forward rates. Which bonds have the highest and lowest convexity? Explain why.

e. Which bond is best suited to serve as an inflation hedge (or more simply a hedge against rising rates)?

f. Construct a duration neutral portfolio that is long one unit of the five-year zero coupon bonds and short Δ units of the thirty-year zero coupon bond. Explain the remaining exposures for this trade. Does it benefit from the curve steepening or flattening?

13.9 Match each of the following exotic options to the most appropriate model below (use each model only once):

Bermudan Swaption
Vanilla Swaption
Cap
Eurodollar Future Convexity Correction

For each answer describe why this choice is optimal.

a. SABR/LMM

b. Hull-White Short Rate Model

c. Black's Model for the Underlying Rate

d. SABR Model for the Underlying Rate

13.10 **Yield Curve Construction**

a. (15 points) Write a series of functions that take a vector of prices of zero coupon bonds with distinct maturities and extracts the following rates:
   i. Zero Rates from today until time T
   ii. Spot Rates from today until time T
   iii. Instantaneous, Continuously Compounded Forward Rates from time t to T
   iv. Semi-Annually Compounded Forward Rates from time t to T
   v. Par Swap Rates from time t to T

   t and T should be inputs to the functions. You may write the functions in Python, or use pseudo-code.

b. (12 points) Consider a 5y fixed-for-floating swap that you enter paying fixed.
   i. What is the present value of the swap entered at the par swap rate?
   ii. Given that you are paying fixed on the swap, do you make or lose money if rates increase?
   iii. Write a function or pseudocode that computes the value of a contract struck at fixed coupon C after the instantaneous forwards in (13.10(a.)iii) increase by δ basis points in parallel. Assume that C was the par swap rate prior to the shift in the instantaneous forwards.

c. (8 points) Consider the following instruments:

- Long Position in Coupon Paying Bond with maturity 5-years
- Long Position in Zero-Coupon Bond with maturity 5-years
- Paying Fixed in 30y Swap

i. Rank the expected change in value of these positions from highest (most positive) to lowest (most negative) from a parallel shift of the curve up by 5 basis points.

ii. Rank the absolute expected change in value of these positions from highest (most positive) to lowest (most negative) from a bullish steepener where short end rates cause the curve to steepen. Assume that only rates in the first two years decline.

Explain why you chose the rankings that you did.

13.11 Suppose you have calibrated a set of caplet volatilities to a SABR model and obtained the following parameters:

Expiry	Alpha	Beta	Rho	Sigma0
3m	0.5	0.5	-0.3	0.1
6m	0.48	0.5	-0.25	0.125
9m	0.46	0.5	-0.2	0.15
1y	0.45	0.5	-0.15	0.1675
15m	0.43	0.5	-0.1	0.18
18m	0.4	0.5	-0.05	0.1875
21m	0.385	0.5	0	0.1925
2y	0.37	0.5	0.05	0.19

a. Using this table of parameters, calculate the price and constant Black volatility of spot starting 1y and 2y caps and floors that are at-the-money and 50 basis points above and below the at-the-money-strike.

b. Create a set of shifts to the volatility surface such that:
   - The volatility surface increases in parallel
   - The volatility surface steepens
   - The skew smiles
   - The tails become fatter

   Evaluate the table of options above under each scenario and comment on which options have the highest and lowest sensitivity to these movements.

c. Without modeling or calculating the value, discuss how you might use this table of SABR coefficients to value a swaption. What are the challenges? What approximations would you rely on?

CHAPTER 14

Credit Markets

14.0.1 Coding Example: Pricing a Risky Bond

In this coding example we implement a pricing function for a bond with embedded default risk. We assume constant interest rates, as well as a constan hazard rate, λ for the risky bond, however, the reader is encouraged to extend this to a more realistic assumptions of time-varying interest and default rates.

```
def price_bond(lamb,pay_times,rf,c,recovery):

    rf_bond = [math.exp(-x*rf) for x in pay_times]
    surv_prob = [math.exp(-lamb*x) for x in pay_times]
    default_prob = [1-x for x in surv_prob]

    default_B = 0
    nondefault_B= 0

    for i in range(len(pay_times)):
        default_B += c*rf_bond[i] * surv_prob[i] + recovery*lamb*
    rf_bond[i]*surv_prob[i]
        nondefault_B += c*rf_bond[i]


    default_B += rf_bond[len(pay_times)]*surv_prob[len(pay_times)]
    nondefault_B += rf_bond[len(pay_times)]

    return default_B,nondefault_B
```

14.0.2 Coding Example: Calibrating a CDS Curve via Optimization

In this coding example we detail how a simple optimization framework could be applied to the problem of calibrating a set of piecewise constant hazard rates to a set of traded credit spreads:

```
import numpy as np
import math
from scipy.optimize import minimize, Bounds, LinearConstraint

def getSurvProbAtT(T, cdsTenors, cdsLambdas, recov):
    survProb = 1.0
```

```
    prevT = 0.0
    for tenor_ii, lambda_ii in zip(cdsTenors, cdsLambdas):
        if (T > prevT):
            tau = min(tenor_ii, T) - prevT
            survProb *= math.exp(-tau * lambda_ii)
        prevT = tenor_ii

    return survProb

def pvCDS(cdsLambdas, cdsTenors, cdsSpread, cdsMat, coupon_frequency, rf, recov):
    total_payments = coupon_frequency * cdsMat
    pv_def_leg = 0.0
    pv_no_def_leg = 0.0

    for payment in range(1,total_payments+1):
        payment_time = payment/coupon_frequency

        pv_no_def_leg += cdsSpread * math.exp(-rf * payment_time) * getSurvProbAtT(payment_time, cdsTenors, cdsLambdas, recov)

        lambda_ord = np.where(cdsTenors == min(cdsTenors[cdsTenors >= payment_time]))[0][0]
        lambda_curr = cdsLambdas[lambda_ord]

        pv_def_leg +=  (1 - recov) * math.exp(-rf * payment_time) * getSurvProbAtT(payment_time, cdsTenors, cdsLambdas, recov) * lambda_curr

    return pv_def_leg, pv_no_def_leg

def cdsPricingErrorSquared(cdsLambdas, cdsTenors, cdsSpreads, coupon_frequency, rf, recov):
    sse = 0.0

    for tenor_ii, spread_ii in zip(cdsTenors, cdsSpreads):
        pv_def_leg_ii, pv_no_def_leg_ii = pvCDS(cdsLambdas, cdsTenors, spread_ii, tenor_ii, coupon_frequency, rf, recov)
        pv_cds_ii = (pv_def_leg_ii - pv_no_def_leg_ii)
        sse += pv_cds_ii**2

    return sse;

cdsSpreads = np.asarray([120, 135, 150, 160, 175]) / 10000.0
cdsTenors = np.asarray([1, 3, 5, 7, 10])
cdsLambdas = [0.05, 0.05, 0.05, 0.05, 0.05]
coupon_frequency = 2
rf = 0.005
recov = 0.4

pvCDS(cdsLambdas, cdsTenors, 0.05*(1-0.4), 5, 2, 0.0, 0.4)

res = minimize(cdsPricingErrorSquared,
               x0 = cdsLambdas,
```

```
                bounds  = ((0.0, None),(0.0, None), (0.0, None), (0.0,
     None), (0.0, None)),
              args = (cdsTenors, cdsSpreads, coupon_frequency, rf,
    recov),
              tol = 1e-12,
              method = 'SLSQP',
              options={'maxiter': 2500, 'ftol': 1e-14})

res
```

14.0.3 Coding Example: Pricing a Knock-out CDS Swaption

In this coding example we show how Black's model can be applied to the case of a knock-out swaption on a single name credit default swap:

```
from scipy.stats import norm

def payer_swaption(F,sigma,t,K, swapT, rf, const_lamb,
    coupon_frequency):
    d1 = (math.log(F/K) + 0.5*sigma**2*t)/(sigma * math.sqrt(t))
    d2 = (math.log(F/K) - 0.5*sigma**2*t)/(sigma * math.sqrt(t))

    ra = 0.0
    matT = swapT + t
    total_payments = int(coupon_frequency * (matT - t))
    for payment in range(1,total_payments+1):
        payment_time = payment/coupon_frequency
        ra += math.exp(-rf * payment_time) * math.exp(-const_lamb *
    payment_time)

    swn_px = ra * (F * norm.cdf(d1) - K * norm.cdf(d2))
    return swn_px

payer_swaption(500.0/10000.0, 0.2, 1.0, 500.0 / 10000.0, 5.0, 0.01,
    0.025, 2)
```

The reader is encouraged to notice the use of the risky annuity term as the numeraire when modeling this CDS swaption.

14.0.4 Coding Example: Building an Index Loss Distribution in the Homogeneous Model

In this coding sample we leverage the formulas presented in the chapter to detail how an index loss distribution can be modeled in Python under the assumption that all credits are homogeneous:

```
import numpy as np

spread = 500.0 / 10000.0
recov = 0.4
tau = 5.0
N = 100
M = 200
```

```
rho = 0.2

def index_loss_distr(N, M, rho, spread, recov, tau):
    implied_lambda = spread / (1 - recov)
    surv_prob = math.exp(-implied_lambda * tau)
    def_prob = 1 - surv_prob
    C = norm.ppf(def_prob)

    fLs = np.zeros(N+1)

    for ii in range(M+1):
        Zi = -5.0 + ii * (10 / M)
        phiZ = norm.pdf(Zi)
        dz = 10 / M

        for n in range(N+1):
            innerNumer = C - rho*Zi
            innerDenom = np.sqrt(1 - rho**2)
            innerTerm = innerNumer / innerDenom

            pLCondZ = (1 - recov) * norm.cdf(innerTerm)
            fLCondZ = (math.factorial(N) / (math.factorial(n) * math.factorial(N-n))) *pLCondZ**n * (1-pLCondZ)**(N-n)
            fLs[n] += fLCondZ * phiZ * dz

    return fLs

fLs = index_loss_distr(N, M, rho, spread, recov, tau)
print(fLs)
print(fLs.sum())
plt.plot(fLs)
```

14.0.5 Coding Example: Merton's Formula

In the following coding sample, we should how a Merton model can be implemented in Python, where we use an optimization framework to infer the asset volatility, σ_A and value of equity from the assets, liabilities and volatility of the equity, σ_E:

```
from scipy.stats import norm
from scipy.optimize import minimize, Bounds, LinearConstraint

def solveForSigmaAssetsAndValueOfEquity(params, A, L, sigmaEq, t, r):
        sigmaAssetsHat = params[0]
        EHat = params[1]

        d1 = (math.log(A/L) + r + 0.5*sigmaAssetsHat**2*t)/(sigmaAssetsHat * math.sqrt(t))
        d2 = (math.log(A/L) + r - 0.5*sigmaAssetsHat**2*t)/(sigmaAssetsHat * math.sqrt(t))

        sigmaAssetsError = (sigmaAssetsHat - sigmaEq * norm.cdf(d1) * (EHat / A))**2
        eError = (EHat - (A * norm.cdf(d1) - L * math.exp(-r*t) * norm.cdf(d2)))**2
```

```
13
14          return (sigmaAssetsError + eError)
15
16 A = 100.0
17 L = 90.0
18 sigmaEq = 0.5
19 t = 5.0
20 r = 0.0
21
22 bounds = Bounds((0,0), (None, None))
23 guess = (sigmaEq, A - L)
24 guess
25 res = minimize(solveForSigmaAssetsAndValueOfEquity,
26                x0 = guess,
27                bounds  = ((0.0, None),(0.0, None)),
28               args = (A, L, sigmaEq, t, r),
29               tol = 1e-10,
30               method = 'SLSQP',
31               options={'maxiter': 400, 'ftol': 1e-14})['x']
32
33 print(res)
34 print(solveForSigmaAssetsAndValueOfEquity(res, A, L, sigmaEq, t, r))
35
36 def merton_model(A, L, sigmaAssets, t, r):
37
38     d1 = (math.log(A/L) + r + 0.5*sigmaAssets**2*t)/(sigmaAssets *
      math.sqrt(t))
39     d2 = (math.log(A/L) + r - 0.5*sigmaAssets**2*t)/(sigmaAssets *
      math.sqrt(t))
40
41     E = A * norm.cdf(d1) - L * math.exp(-r*t) * norm.cdf(d2)
42
43     DD = (math.log(A/L) + (r - 0.5*sigmaAssets**2*t)) / (sigmaAssets
      * math.sqrt(t))
44     pdef = norm.cdf(DD)
45     return E, sigmaAssets, DD, pdef
46
47 print(merton_model(A, L, res[0], t, r))
```

EXERCISES

14.1 Consider the following table of par spreads for an IG credit index:

Maturity	Par Spread (bps)
1y	13.93
3y	27.50
5y	50.98
7y	71.13
10y	92.30

Assume that the recovery rate, R, is 40% and the discount curve is defined by flat instantaneous forward rates at 2%.

a. Write a bootstrapping algorithm that extracts a set of piecewise constant hazard rates that enables you to match all quoted par spread simultaneously.

b. Using the credit triangle formula, compute estimates of the hazard rates for each maturity. Comment on the difference between these hazard rates and the bootstrapped hazard rates obtained above.

c. Assume now that the convention is to trade IG indices with a fixed coupon of 100 basis points. Calculate the upfront value that would be required for the present value of this contract to be equal to zero.

d. Calculate the sensitivity of each contract to a parallel shift upwards of the par spread curve by 10 basis points.

e. Calculate the sensitivity of each contract to a flattening of the credit curve defined as follows:

Maturity	Par Spread (bps)
1y	20
3y	15
5y	10
7y	5
10y	0

Which contracts have the highest sensitivity to this flattening scenario? Why?

f. Calculate the present value of each CDS contract if the recovery amount R assumption is modified to 30%.

14.2 Suppose you are creating a bespoke Emerging market sovereign credit index product that is a function of the following underlying single-names:

Entity	Weight	5Y Par Spread (bps)
Brazil	0.25	223
Turkey	0.25	443
Russia	0.25	116
Chile	0.25	56

Assume that each entity has an assumed recovery of 40%.

a. Find the constant hazard rate that matches each par CDS spread and the survival probability of each entity five years in the future.

b. Calculate the fair spread of the bespoke index.

c. Suppose the dealer quotes you a spread of 200 basis points to enter into this contract. Would you enter into this contract? Would be long or short protection?

d. Is there a trade available with the single-names and the bespoke index that you would engage in? Describe carefully the units of each item you would buy or sell and the properties of the entire package.

14.3 Consider an investor who is looking to harvest carry in credit markets:

a. Download historical data for HYG, JNK and LQD and clean the data for stock splits and other anomalies.

b. Calculate the rolling 1y realized volatility of each ETF and comment on any patterns that you observe.

c. Estimate the carry in each ETF and compute the implied sharpe ratio due to carry as: $\frac{\text{Carry}}{\text{Realized Vol}}$. Where realized vol is measured over the entire period. Rank the ETFs by their relative carry and compare them to the analogous value for S&P 500 and comment on the difference.

d. Obtain at-the-money implied volatility data for each ETF and compare it to realized volatilty. Is it higher or lower? Why?

14.4 Consider a CDS whose 5 year par spread is 145 basis points.

a. Calculate the value of a knockout payer swaption with strike spread of 150 basis points expiring in three months. Assume $\sigma = 0.2$.

b. Calculate the value of the front-end protection on this CDS from today until the option expiry in three months.

c. Use the results above to value a non-knockout payer swaption with strike spread of 150 basis points expiring in three months.

d. Calculate delta, gamma, vega and theta of the knockout swaption. Describe the dominant Greeks for this option.

e. Describe how you would construct a delta-hedging scheme for this option and describe the major remaining risks once the delta has been neutralized.

f. Plot the value of the non-knockout option as time passes from three months until expiry and use this to comment on the theta profile of the option.

14.5 Consider the Hirsa-Madan approach to connecting equity and credit:

a. Download an options volatility surface for Xerox Corp (ticker XRX). Clean the data for arbitrage.

b. Calibrate a Variance Gamma model for each expiry with the additional default probability parameter calibrated as well. Comment on the default probabilities for each expiry.

c. Try multiple start guesses, upper and lower bounds and weighting schemes in your calibration. Are the parameters stable with respect to changes in the optimization structure?

d. Repeat this exercise with a CEV model replacing the Variance Gamma dynamics and comment on any differences in the extracted default probabilities.

14.6 Assume that the risk-free interest rate is 2% for all maturities and suppose that the CDS spreads for contracts that are starting today are given by the table below. Also, assume that the expected recovery $R = 40\%$.

Maturity	Par Spread (bps)
1y	100
2y	110
3y	120
5y	140

a. Use bootstrapping to extract a set of piecewise constant hazard rates that allow us to simultaneously match the par spreads in 14.6.

b. Using these calibrated hazard rates, calculate the pair spread of 4y CDS

c. Suppose you had bought a 5y CDS exactly one year ago with a par spread of 80 basis points. What is the present value of this CDS today?

d. Compute the sensitivity of each quoted CDS with respect to a parallel shift in the par CDS spread curve.

e. Compute sensitivity of each quoted CDS with respect to R.

14.7 Suppose a company HTZ issues a 5 year bond with a 4% coupon, paid semi-annualy. The current market price of the bond is \$10 (per \$100 face) and the risk-free interest rate is constant at 1%.

Suppose the expected recovery is zero, i.e., $R = 0\%$, and the spreads are

Maturity	spread (bps)
1y	8173
2y	6672
3y	6051
5y	5330
7y	4720
10y	4152

Assume that, as usual, the CDS payments are quarterly.

a. Compute the quarterly survival curve out to 5 years. Be explicit about the assumptions you are making.

b. Using survival probabilities from 14.7a., show how would you value a single payment at time t that is conditional on survival of HTZ until time t.

c. Consider the bond above that pays a 2% coupon twice a year. Show how you would value this coupon bond on HTZ? Provide the bond's present value.

d. Is the market price above or below the value? What you do if it is above or below?

e. Calculate CDS-bond basis using the following method: find the parallel shift of the CDS curve that would make the present value of the bond's cashflow equal to its market price.

f. Suppose that you own $\$10M$ notional (face value) of the bond. How much of the CDS should you buy to offset as much of the bond's risk as possible? Please specify the maturity, notional, and the spread of the CDS contract.

CHAPTER 15

Foreign Exchange Markets

15.0.1 Coding Example: Pricing FX Options via the Black-Scholes Model

In this coding example, we show how this modified version of the Black-Scholes model can be used to price vanilla call options on an exchange rate:

```
import math

def fxBsCall(S0, K ,sigma, r_d, r_f, T):
    F = S0 * exp ((r_d - r_f) * T)
    d1 = (1/(sigma*np.sqrt(T))) * ( np.log(F/K) + 0.5*(sigma**2)*T )
    d2 = d1 -sigma*np.sqrt(T)
    pxCall = np.exp(-r*T)*( F*norm.cdf(d1) - K*norm.cdf(d2) )
    return pxCall
```

The reader should note that the risk-free rate in this Black-Scholes pricing formula should correspond to the risk-free rate of the domestic currency .

15.0.2 Coding Example: Calibrating an FX Vol. Surface using SABR

In the following coding example we show how a SABR model can be calibrated to a quoted FX volatility surface:

```
from pyfinance.options import BSM

def bs_sabr_vol(F_0, K, T, sigma_0, beta, alpha, rho):
    normVol = sabr_normal_vol(F_0,strike,T,beta,sigma_0,alpha,rho)
    normPx = bachelier_call(F_0,strike,T,normVol,0.0)
    impVol = get_implied_vol_bs(F_0,strike,T,0.0, normPx,0.2)
    return impVol

def strikeGivenDelta(F_0, T, rDom, rFor, delta, sigma):
    K = F_0 * math.exp(-sigma*np.sqrt(T)*norm.ppf(delta) + 0.5*(sigma
    **2)*T);
    return K

def sabrDeltForStrike(F_0, T, K, sigma_0, alpha, beta, rho, isPut):
    impVol = bs_sabr_vol(F_0, T, K, sigma_0, beta, alpha, rho)
    d1 = (1/(impVol*np.sqrt(T))) * ( np.log(F_0/K) + 0.5*(impVol**2)*
    T )

```

```
17      if isPut > 0:
18          delta = -norm.cdf(-d1)
19      else:
20          delta = norm.cdf(d1)
21      return delta
22
23  def getStrikeGivenDeltaSABRSmile(F_0,T,sigma_0, alpha, beta, rho,
        isPut, deltaHat, initial_guess):
24      root_fn = lambda x: sabrDeltForStrike(F_0,T,x,sigma_0, alpha,
        beta, rho, isPut) - deltaHat
25      return root(root_fn,initial_guess)['x'][0]
26
27  def objectiveFunctionFX(params,beta,F_0,T, K_atm, K_c_st, K_p_st,
        rDom, rFor, sigma_atm, rr_vol, st_px):
28
29      #extract the 25D risk reversal strikes
30      K_c_rr = getStrikeGivenDeltaSABRSmile(F_0,T,params[0], params[1],
         beta, params[2], 0, 0.25, K_c_st)
31      K_p_rr = getStrikeGivenDeltaSABRSmile(F_0,T,params[0], params[1],
         beta, params[2], 1, -0.25, K_p_st)
32
33      #check the pricing error
34      sigma_atm_hat = bs_sabr_vol(F_0, K_atm, T, params[0], beta,
        params[1], params[2])
35      atm_error = (sigma_atm - sigma_atm_hat)**2
36
37      sigma_rr_c_hat = bs_sabr_vol(F_0, K_c_rr, T, params[0], beta,
        params[1], params[2])
38      sigma_rr_p_hat = bs_sabr_vol(F_0, K_p_rr, T, params[0], beta,
        params[1], params[2])
39      rr_vol_hat = sigma_rr_c_hat - sigma_rr_p_hat
40      rr_error = (rr_vol - rr_vol_hat)**2
41
42      sigma_st_c_hat = bs_sabr_vol(F_0, K_c_st, T, params[0], beta,
        params[1], params[2])
43      sigma_st_p_hat = bs_sabr_vol(F_0, K_p_st, T, params[0], beta,
        params[1], params[2])
44      px_st_c_hat = math.exp(-r_d*T)*BSM(kind='call', S0=F_0, K =
        K_st_c, T = T, r = 0.00, sigma = sigma_atm+strangle_offset).value
        ()
45      px_st_p_hat = math.exp(-r_d*T)*BSM(kind='put', S0=F_0, K = K_st_p
        , T = T, r = 0.00, sigma = sigma_atm+strangle_offset).value()
46      st_px_hat = (px_st_c_hat + px_st_p_hat)
47      st_error = (st_px - st_px_hat)**2
48
49      error = atm_error + rr_error + st_error
50      return error
51
52  #input data
53  F_0, r_d, r_f, T = 100.0, 0.015, 0.005, 0.5
54  sigma_atm, sigma_rr, strangle_offset = 0.2, 0.04, 0.025
55
56  #extract the ATM  strike
57  K_atm = strikeGivenDelta(F_0, T, r_d, r_f, 0.5, sigma_atm)
58
```

```
#extract the 25D strangle strikes
K_st_c = strikeGivenDelta(F_0, T, r_d, r_f, 0.25, sigma_atm+
    strangle_offset)
K_st_p = strikeGivenDelta(F_0, T, r_d, r_f, 0.75, sigma_atm+
    strangle_offset)

#calculate 25D strangle price
st_c_px = math.exp(-r_d*T)*BSM(kind='call', S0=F_0, K = K_st_c, T = T
    , r = 0.00, sigma = sigma_atm+strangle_offset).value()
st_p_px = math.exp(-r_d*T)*BSM(kind='put', S0=F_0, K = K_st_p, T = T,
     r = 0.00, sigma = sigma_atm+strangle_offset).value()
st_px = st_c_px + st_p_px

results = minimize(objectiveFunctionFX,
                   x0 = [0.15,0.5,0] ,
                bounds  = ((0.00001, F_0),(0.00001, 10),(-1,1)),
                args = (beta,F_0,T,K_atm, K_st_c, K_st_p, r_d, r_f,
    sigma_atm, sigma_rr, st_px),
                tol = 1e-14,
                method = 'SLSQP',
                options={'maxiter': 400, 'ftol': 1e-14})['x']

print(f'sigma_0 = {results[0]} \nalpha = {results[1]} \nrho = {
    results[2]}')
```

As is customary in a SABR model calibration, the parameter β is set to a constant, due to its correlation with ρ, and the remaining three parameters are calibrated.

15.0.3 Coding Example: Pricing Digis and One-Touches

In the following coding example, we detail how digital and one-touch prices can be computed given a set of simulated paths:

```
def priceDigiCall(paths, K, r):
    """paths: 2D np array, each subarray represents a period in time
    """
    terminalVals = paths[-1]
    payoffs = np.where(terminalVals > K,1,0)
    return np.exp(-r*T)*np.mean(payoffs)

def priceOneTouchCall(paths, K, r):
    """paths: 2D np array, each subarray represents a period in time
    """
    maxVals = np.max(paths, axis = 0)
    payoffs = np.where(maxVals > K,1,0)
    return np.exp(-r*T)*np.mean(payoffs)
```

For more details on how to build the code to generate the underlying simulated paths, the reader is encourage to refer to the coding examples in chapter 10.

EXERCISES

15.1 **Time Series Analysis of FX Carry**

a. Download historical data for the USDCAD exchange rate as well as one-month forwards.

b. Extract and plot a time series of the interest rate differential for the USD-CAD currency pair. Comment on how stable and / or volatile it is.

c. Compare the expected change in USDCAD over the interval implied by the forwards to the actual change in the rate. Has the forward realized?

d. Plot subsequent one-month returns of USDCAD as a function of the interest rate different. Do you observe a positive or negative relationship? Why?

15.2 **FX Volatility Surface Calibration**

a. Using Bloomberg or another data provider, download volatility surface data for EURUSD for one, three and six month expiries.

b. Extract the at-the-money strike corresponding to the quoted at-the-money volatility.

c. For each each, utilize the at-the-money, risk reversal and butterfly quotes to calibrate a set of SABR parameters to the volatility surface. Comment on the fit of the data and how the parameters evolve as expiry increases.

d. Using the calibrated SABR models calculate the price of digitals and one-touch options with 25-delta strikes. Is the ratio in line with the 2:1 rule of thumb? Why or why not?

e. Shift the SABR parameter ρ higher and lower by 0.1 and recalculate the price of the digital and one-touch option. What happens to the ratio of one-touch vs. digital prices as we shift ρ?

f. Shift each SABR parameter and identify which parameter most directly impacts each of the quoted points, at-the-money, risk reversals and butterflies.

15.3 **Barrier Option Pricing**:

a. Consider a Variance Gamma model with the following parameters:
   i. $S_0 = 100$
   ii. $\sigma = 0.15$
   iii. $\theta = -0.1$
   iv. $\nu = 0.25$
   v. $r = 0$

   Using FFT, obtain prices for three-month European call and put options for many strikes.

b. Convert each price to an implied volatility and plot the three-month volatility smile.

c. Consider a three-month 110 strike European call option on this underlying. Using simulation, calculate its price and compare to the analogous price you get using FFT. Comment on any difference in prices due to switching techniques.

d. Next, consider the same three-month 110 strike European call, this time with a European-style knock-out at 125. Use simulate to obtain an estimate for the price of this option under the Variance Gamma model.

e. Calculate the rebate that you get from adding the European style barrier. What does this imply about the distribution at expiry? Do you believe it makes the trade more or less attractive?

f. Finally, consider the same knock-out option this time with an American style barrier. That is, consider a 110 strike, three-month European call option that knocks-out if the value of the asset exceeds 125 at any point during the period. Estimate the price of this structure via simulation, and calcualte the rebate relative to the European option with no barrier. How would you decide if you thought this rebate was more (or less) attractive than that of the European-style barrier?

g. Shift each Variance Gamma parameter and comment on how they impact the following quantities computed above:
   - European Call
   - European Call with European Barrier
   - Rebate from European Barrier
   - European Call with American Barrier
   - Rebate from American Barrier

15.4 **Volatility vs. Variance Swaps**:

a. Consider a Heston model with the following parameters:
   i. $S_0 = 100$
   ii. $\kappa = 0.5$
   iii. $\theta = 0.15$
   iv. $\rho = -0.7$
   v. $\xi = 0.6$
   vi. $\sigma_0 = 0.08$
   vii. $r = 0$

   Given this set of Heston parameters, comment qualitatively on what you expect the smile and term structure of volatility to look like.

b. Using simulation, calculate the value of a one-year variance swap with $\sigma_k = 0.01$. If this contract could be executed with no upfront payment, would it be over or undervalued?

c. Again using simulation, calculate the value of a one-year volatility swap with $\sigma_k = 0.1$ Would this contract struck at 0 upfront cost be over or undervalued?

d. Construct a histogram of payoffs for the volatility and variance swap. Comment on when they are similar and when they diverge the most.

e. Construct a strategy that is long the volatility swap and short the variance. That is, consider a strategy that pays fixed and received floating on a volatility swap and receives fixed and pays floating on a variance swap. How would you think about sizing the two legs?

f. Make a histogram of payoffs of this strategy. How does it compare to the outright histograms for the volatility and variance swaps?

15.5 **Cross-Sectional Analysis of FX Carry Strategy**:

a. Download historical exchange rate and three-month forward data for the G10 major currencies vs. USD. Clean the data for outliers and anomalies.

b. Plot the interest rate differential for each of the 10 currency pairs over time. For which currency pairs does USD tend to have a higher (lower) yield?

c. For each date, sort the currency pairs by interest rate differential and comment on how this evolves over time.

d. Back-test a strategy that is long the three pairs with the highest carry and short the three pairs with the lowest carry. The strategy should take positions in the spot exchange rate.

15.6 **One-touch vs. Digital Option**:

a. Download current volatility surface data for USDZAR three-month options using Bloomberg or another source of your choice, and clean as needed.

b. Plot the volatility smile for USDZAR options and explain why you think it is shaped the way that it is.

c. Using the volatility surface calibration techniques discussed in the chapter, calibrate a SABR model to the data.

d. Extract the strikes that correspond to 10 and 25D calls and puts respectively. Calculate the price of out-of-the-money digis and one-touches at each of these strikes. What is the ratio of one-touch to digital prices? Is it similar for each strike? If not, where is it the highest (lowest)?

e. Repeat this exercise for the GBPUSD currency pair. Plot the volatility smile for GBPUSD and compare it to that of USDZAR.

f. As before, calibrate a SABR month to the set of three-month options and calculate digital and one-touch option prices for 10D and 25D strikes. Are they ratios of one-touch to digital higher or lower than for USDZAR? Why?

CHAPTER 16

Equity & Commodity Markets

EXERCISES

16.1 Consider a basket option on the following assets:

Ticker	Description
SPY	S&P 500 Index
DIA	Dow Jones Industrials Index
QQQ	NASDAQ
IWM	Russell 2000 Small Cap Index

a. Download historical data for the basket components and calculate the entire period realized volatility and correlation matrix.

b. Assume that each asset is governed by a Black-Scholes model with volatility equal to historical, realized volatility and that the correlation structure of the assets matches the realized correlation matrix. Using simulation, price a six-month basket call option with strike equal to 105% of the current basket value. Further assume that $r = 0$, and $q = 0$ for all assets, and that each asset is currently trading at 100.

c. Re-calculate the price of the basket option using the current asset prices for S_0 instead of 100. Comment on the impact this scaling has on your model.

d. Comment on whether using realized volatility as a proxy for implied volatility is a reasonable assumption and what bias it might introduce into your model price.

e. Build a portfolio of vanilla options on the underlying assets that will best replicate the basket options. Describe any remaining exposures and why they exist.

f. Calculate the sensitivity of the structure to an increase and decrease in

correlations of the assets. Explain why this relationship exists for a basket option.

g. Discuss the strengths and weakness of applying the Black-Scholes model for basket options pricing.

16.2 Consider a worst-of option on the following assets:

Ticker	Description
SPY	US Equities (S&P 500 Index)
EFA	Europe, East Asia and the Far East
EEM	Emerging Markets

a. Download current implied volatility data for the three underlying assets and calibrate individual Heston models for the assets.

b. Download historical price and return data and calculate a covariance matrix of asset returns. Be explicit about the assumptions you make in estimates the covariances, such as the lookback window and sampling frequency and why you believe they are appropriate.

c. Build a correlation matrix that can be used to perform a multi-dimensional simulation from the Heston model by assuming that the cross asset correlations are in line with the realized estimates that you obtained above, and that the cross asset volatility processes are uncorrelated.

d. Comment on whether you think these correlation assumptions across the assets and with respect to the volatility processes are appropriate, and what impact you think it might have on the results.

e. Consder a three-month, at-the-money, worst-of option on these assets. Using simulation, the individual calibrated Heston parameters and the augmented covariance matrix, estimate the price of this worst-of option. Assume that each asset has a starting value of 100, and that $r = 0$ and the assets pay no dividends ($q = 0$).

f. Create a table of sensitivities to each Heston parameter for each asset as well as the additional correlations. What parameter or parameters are most important for worst-of options? Why do you think this is?

16.3 Consider a 3 month fixed strike lookback put on the ETF SPY with its strike set to 90% of its current value.

a. Describe the conditions in which the lookback makes the most money.

b. Gather data on the current level of SPY, as well as the current level of VIX, the current three month T-Bill rate, and the dividend yield of the S&P.

c. Using the current value of VIX as a proxy for the implied volatility, use to Black-Scholes formula to estimate the price of a three month European put with its strike equal to 90% of its current value. Use the parameters you obtained above for S_0, r and q, respectively.Perform a simulation from the Black-Scholes model to verify that your simulation yields the same result as the closed form solution and comment on any divergence.

d. Using the same implied volatility, estimate the price of the lookback via simulation assuming it follows a Black-Scholes model. As before, use the parameters you obtained above for S_0, r and q, respectively.

e. Plot the payoffs of the European and Lookback across each path and calculate the correlation of their payoffs. Comment on when each payoff is a highest and lowest. What identifying traits about these paths can you identify? What about on the paths when the values diverge?

16.4 Consider an investor who buys a one-month fixed strike lookback structk at 95% of the value of the financials ETF, XLF and sells a European put option with matching expiry and strike.

a. Derive the payoff for the lookback vs. European structure as a function of the minimum value of XLF, its terminal value, and the strike, K.

b. Describe how you would expect this trade to behave as a function of the following:
   - Volatility
   - Upward Drift
   - Skew to the Put Side
   - Mean-Reversion in XLF

c. Download current options data for XLF and calibrate a Variance Gamma to the set of one-month options. Comment on what the parameters imply about the risk-neutral distribution.

d. Using the calibrated Variance Gamma model, use FFT techniques to price the European option that you have sold.

e. Using simulation and the same set of Variance Gamma parameters, obtain a comparable simulation based estiamte of the European option price. Does it match the FFT based value? Why or why not?

f. Use simulation to value both the lookback option in isolation and the value of the Lookback vs. European package. Comment on the difference in payoff profiles. What does the sold European option do to our profile?

16.5 Consider a VIX roll-down strategy that attempts to harvest risk premium in VIX futures:

a. Download data for all available VIX futures as well as the VIX index.

b. Extract the annualized implied dividend, q, for each future with respect to spot. Plot this data for each future historically.

c. Also plot the slope of the futures curve with respect to spot as defined by:

$$s = \log\left(\frac{f_i}{s_0}\right) \tag{P16.1}$$

where f_i is the value of future i, s_0 is the spot value of the VIX index and s is the slope of the curve. Annualize these slopes in your plot. Which parts of the curve have the most/least slope?

d. Consider a strategy that persistently sells the one-month future and rolls one-week prior to expiry. What is the payoff profile of this strategy?

e. Next, consider a strategy that persistently sells the three-month future and rolls every month (when it becomes a two-month future). How does this compare to selling the front month? How correlated are their payoffs? Do you see any pros or cons or using the front vs. third month to harvest a roll premium?

f. Finally, consider a dynamic strategy which creates a long-short portfolio that is long the flattest part of the curve and short the steepest. Comment on your approach to sizing the long and short legs.

16.6 Consider a set of options on oil futures:

a. Download the current futures and volatility surface data. Check the options data for arbitrage correcting any anomalies.

b. Plot the volatility smile for each option expiry and choose the stochastic model that you think is best suited to calibrate the features of the vol. surface. Explain why you think this is the case.

c. Using the calibrated volatility surface price the following set of exotic options:
   - An at-the-money basket put option with a one-year expiry.
   - A 5% out-of-the-money best-of call option with a one-year expiry.
   - A 5% out-of-the-money worst-of call option with a one-year expiry.

d. Calculate the ratio of the exotic options relative to each other. Which is the biggest? Smallest? Why?

e. Try several different parameter sets and comment on how these ratios evolve over time. Is the order preserved? Why or why not? Are the ratios contained with certain intervals?

16.7 Consider an investor looking to sell strangles on the NASDAQ ETF QQQ and delta-hedge:

a. Download historical price and options data for the QQQ ETF. Compute a time series of realized and implied volatilities and plot these time series as well as the spread between implied and realized.

b. Assuming a flat volatility surface with volatility equal to the at-the-money observed volatility, calculate the price of a 25-delta one-month strangle on QQQ.

c. Calculate the realized payoff of being short this 25-delta strangle without any hedge. What is the expected payout? Plot a histogram of payoffs as well.

d. Back-test a strategy that sells one-month 25-delta strangles on QQQ and then delta-hedges every week. A single, one-month short strangle position should be held each month until expiry, or another roll date of your choice. What is the risk / return profile of this strategy?

e. What are the main risks when running this type of volatility selling algorithm? How could the underlying structure be improved while still allowing us to harvest the well-documented volatility risk premium?

16.8 Consider the following universe of commodities:

Bloomberg Ticker	Commodity
NG	Natural Gas
CL	Oil
C	Corn
W	Wheat
GC	Gold
HG	Copper
CT	Cotton
SB	Sugar

a. Download historical data for the front two month futures for the above commodity asset universe.

b. Calculate the slope of each futures curve as:

$$s = \log\left(\frac{f_2}{f_1}\right) \tag{P16.2}$$

where f_2 is the second month future's value and f_1 is the value of the front month future.
Plot this slope, s for each commodity over time and comment on the various slopes across time and across the cross section of commodities.

c. Build an algorithm that sorts the commodities by curve slope at each historical date. Plot the relative rankings over time. Which commodities have the steepest / flattest curves consistently?

d. At each date, build an equally weighted portfolio that is long the second future for the two flattest curves and short the second future for the two steepest curves, re-balancing every day. What are the properties of this strategy? Does it have appealing risk/reward characteristics?

IV

Portfolio Construction & Risk Management

CHAPTER 17

Portfolio Construction & Optimization Techniques

17.0.1 Coding Example: Creating an Efficient Frontier

In this coding example we detail how a fully invested efficient frontier can be generated and visualized in Python:

```
def plot_efficient_frontier(ret, cov_mat, N=5000):

    assert len(ret)==len(cov_mat), 'Please make sure the returns
    matches the shape of the covariance matrix.'

    # compute coefficients
    n = len(ret)
    a = np.ones(n).T@np.linalg.inv(cov_mat)@ret
    b = ret.T@np.linalg.inv(cov_mat)@ret
    c = np.ones(n).T@np.linalg.inv(cov_mat)@np.ones(n)
    d = b*c-a**2

    # compute optimal portfolios
    ret_arr = np.linspace(0.05,0.2,N)
    vol_arr = np.zeros(N)
    weight_arr = np.zeros((N, len(ret)))

    for i in range(N):
        w = 1/d*(c*np.linalg.inv(cov_mat)@ret-a*np.linalg.inv(cov_mat
    )@np.ones(n))*ret_arr[i] + 1/d*(b*np.linalg.inv(cov_mat)@np.ones(n
    )-a*np.linalg.inv(cov_mat)@ret)
        vol_arr[i] = np.sqrt(w.T@cov_mat@w)
        weight_arr[i,:] = w

    # plot the efficient frontier
    plt.scatter(vol_arr, ret_arr)
    plt.xlabel('Volatility')
    plt.ylabel('Return')
    plt.show()

    return weight_arr, ret_arr, vol_arr
```

Given a vector of expected returns and a covariance matrix, we compute the optimal portfolios under different desired expected returns using the formula given in section **??**. These optimal portfolios together form the efficient frontier. It should be emphasized that in the formulation of our optimization we enforced that the weights summed to one, or that the portfolio was fully invested. However, we did not constrain the portfolio to be long-only. As such, in this coding example we allow positive and negative weights, and solutions that may contain leverage.

17.0.2 Coding Example: Safely Inverting a Covariance Matrix

In this coding example, we show how a covariance matrix can be inverted in Python. Importantly, in this coding example we make sure that the inverse operation is stable by checking for negative and small, statistically insignificant eigenvalues. These values are set to zero, and the remaining eigenvalues are rescaled.

We begin by checking the matrix for negative eigenvalues. A negative eigenvalue implies an invalid covariance matrix, and therefore should be handled prior to inverting the matrix. Next, we calculate the relative value of each eigenvalue, which also represents the explained variance by each eigenvector, to see if all eigenvalues are statistically different from zero. A threshold is then introduced to rule out the eigenvalues and eigenvectors that are most likely to be noise. Finally, we invert the noise-free diagonal matrix Λ, enabling us to get the inverse of the covariance matrix in a stable way [1]. The function below shows each step in order and provides visualizations for eigenvalues:

```
1  def inverse_cov_mat(cov_mat, eps=1e-2, is_plot=True):
2
3      w, v = np.linalg.eig(cov_mat)
4
5      # step 1. check if eigenvalues are non-negative
6      assert np.where(w>=0, True, False).sum()==len(w), 'Please ensure
       the covariance matrix is positive semi-definite.'
7
8      # step 2. calculate relative weights and drop small eigenvalues
9      weighted_w = w/np.sum(w)
10     if is_plot:
11         plt.plot(np.sort(w)[::-1], marker='x', label='eigenvalue')
12         plt.legend()
13         plt.show()
14         plt.bar(range(len(w)), np.sort(weighted_w)[::-1], width=0.3,
       label='relative weight')
15         plt.legend()
16         plt.show()
17
18     w_hat = np.where(weighted_w>=eps, w, 0)
19     noise_free_w = w_hat*(np.sum(w)/np.sum(w_hat))
20
21     # step 3. calculate inverse matrix
22     inv_mat = v@np.diag(np.where(noise_free_w!=0, 1/noise_free_w, 0))
       @v.T
```

[1] As is shown in (**??**)

```
23
24      return w, noise_free_w, inv_mat
```

17.0.3 Coding Example: Finding the Tangency Portfolio

In section 17.0.1, we show how to compute the optimal portfolio given an expected return. In order to find the tangency portfolio on the efficient frontier, we just need to go one step further. That is, compute the Sharpe Ratio for each optimal portfolio and pick the portfolio with highest Sharpe. We can build on top of the **plot_efficient_frontier** function in 17.0.1, and use the following snippet to locate the tangency portfolio:

```
1 sharpe_arr = ret_arr / vol_arr
2 tan_idx = sharpe_arr.argmax()  # sharpe_arr contains SR of all
      portfolios on the efficient frontier
3 tan_weight = weight_arr[tan_idx,:]  # weight_arr stores the weight
      vector for all portfolios on the efficient frontier
4 tan_ret = ret_arr[tan_idx]
5 tan_vol = vol_arr[tan_idx]
```

17.0.4 Coding Example: Resampling an Efficient Frontier

This coding example computes a resampled efficient frontier under the assumption that return are jointly normal. For each simulation of the multi-variate normal distribution, we formulate a joint path for all underlying assets using the given mean and covariance of the returns. This will imply a new mean return and a new covariance matrix specific to each simulated path. We can then make use of previous developed functions to compute the efficient frontier along each simulated path. After iterating through all simulated, an average weight vector is computed, which is our resampled efficient frontier:

```
1  def resampled_efficient_frontier(ret, cov_mat, size=252, N_path=1000,
       N_point=5000):
2
3      assert len(ret)==len(cov_mat), 'Please make sure the returns
       matches the shape of the covariance matrix.'
4      n = len(ret)
5      ret_arr = np.linspace(0,0.3,N_point)
6      vol_arr = np.zeros(N_point)
7      weight_arr_all = np.zeros((N_path,N_point,n))
8
9      for i in range(N_path):
10
11         # generate resampled paths
12         data = np.random.multivariate_normal(ret, cov_mat, size=size)
13         ret_i = data.mean(axis=0)
14         cov_mat_i = np.cov(data.T)
15
16         weight_arr = np.zeros((N_point,n))
17         a = np.ones(n).T@np.linalg.inv(cov_mat_i)@ret_i
18         b = ret_i.T@np.linalg.inv(cov_mat_i)@ret_i
```

```
        c = np.ones(n).T@np.linalg.inv(cov_mat_i)@np.ones(n)
        d = b*c-a**2

        # compute the efficient frontier
        for j in range(N_point):
            w = 1/d*(c*np.linalg.inv(cov_mat_i)@ret_i-a*np.linalg.inv
    (cov_mat_i)@np.ones(n))*ret_arr[j] + 1/d*(b*np.linalg.inv(
    cov_mat_i)@np.ones(n)-a*np.linalg.inv(cov_mat_i)@ret_i)
            weight_arr[j,:] = w

        weight_arr_all[i,:] = weight_arr

    # average the weights of multiple efficient frontiers
    avg_weight_arr = weight_arr_all.mean(axis=0)

    for k in range(N_point):
        w = avg_weight_arr[k,:]
        vol_arr[k] = np.sqrt(w.T@cov_mat@w)

    return ret_arr, vol_arr
```

17.0.5 Coding Example: Risk Parity Optimization

In this coding example, we detail how to find an equal risk contribution portfolio as defined in (**??**). In the function **rc_err**, we compute the sum of squared distance from equal risk contributions given a weight vector w and covariance matrix Σ. We then use scipy.optimize to minimize the distance from equal risk contributions. We further assume a long-only portfolio that does not contain leverage. To do this, we make sure the weights $0 \leq w_i \leq 1$ using **Bounds** and further that they sum to one using a **LinearConstraint**:

```
from scipy.optimize import minimize, Bounds, LinearConstraint

def rc_err(w):

    n_asset = len(cov_mat)
    denom = np.sqrt(w.T@cov_mat@w)
    numer = np.zeros(n_asset)
    rc = np.zeros(n_asset)

    for i in range(n_asset):
        numer[i] = w[i]*(cov_mat@w)[i]
        rc[i] = numer[i]/denom

    avg_rc = np.sum(rc)/n_asset
    err = rc - avg_rc
    squared_err = np.sum(err**2)

    return squared_err

bounds = Bounds([0.0]*len(cov_mat), [1.0]*len(cov_mat))
sum_constraint = LinearConstraint([1.0]*len(cov_mat), [1.0], [1.0])
w0 = np.array([1.0/len(cov_mat)]*len(cov_mat))
```

```
23 res = minimize(rc_err, w0, method='trust-constr', constraints=
       sum_constraint, bounds=bounds)
24 print(res.x)
```

It should be emphasized that this implement of a risk parity, or equal risk contribution portfolio does not include leverage. We could then add leverage to the portfolio by simply rescaling the weights, and adding a corresponding short position in cash, or the risk-free asset.

EXERCISES

17.1 Consider the following set of sector ETFs:

Ticker	Description
XLB	Materials
XLE	Energy
XLF	Financials
XLI	Industrials
XLK	Technology
XLP	Consumer Staples
XLU	Utilities
XLV	Healthcare
XLY	Consumer Discretionary

a. Download historical price data from January 1st 2010 until today for the sector ETFs in the table above. Clean the data for splits and any other anomalies.

b. Calculate the covariance matrix of daily returns for the sector ETFs.

c. Perform an eigenvalue decomposition of the covariance matrix. Plot the eigenvalues in order from largest to smallest. How many eigenvalues are positive? How many are negative? How many are zero? If any are negative, is this a problem? How many do you think are statistically significant?

d. Generate a random matrix of the same size as your covariance matrix, where each element has a standard normal distribution.

e. Perform an eigenvalue decomposition of this random matrix. Plot the eigenevalues in order from largest to smallest. Comment on the differences and similarities between the structure of the eigenvalues in this matrix and your historical covariance matrix.

17.2 Recall that an unconstrained optimal portfolio can be found using the following formula:

$$\max_{w} w^T R - a w^T C w \tag{P17.1}$$

The constant a indicates the amount of our risk aversion. Assume you are an investor with a risk aversion coefficient of 1.

a. Calculate the historical annualized returns for each sector ETFs in 17.1.

b. Calculate the covariance matrix of the returns of the sector ETFs. Comment on the rank of the matrix and the profile of the eigenvalues.

c. Using the annualized returns as expected returns and the covariance matrix obtained above, calculate the weights of the unconstrained mean-variance optimal portfolio.

d. Create a new set of expected returns equal to the historical annualized return (μ) plus a random component, that is:

$$\mathbb{E}[r] = \mu + \sigma * Z \tag{P17.2}$$

where σ is the volatility of the random component and Z is a standard normal random variable.
Using $\sigma = 0.005$, $\sigma = 0.01$, $\sigma = 0.05$ and $\sigma = 0.1$ calculate the unconstrained mean-variance optimal portfolio using the adjusted expected returns and the covariance matrix. Comment on the stability of your portfolio weights to changes in expected returns.

e. Consider a regularized covariance matrix that is a blend of the historical covariance matrix and a diagonal matrix with each asset's variance along the diagonals, that is:

$$\tilde{\Sigma} = \delta\Sigma_{\text{diag}} + (1-\delta)\Sigma_{\text{full}} \tag{P17.3}$$

where δ is the weight of the diagonal matrix and $(1-\delta)$ is the weight of the historical covariance matrix.

f. Set $\delta = 1$ and perform and eigenvalue decomposition of the regularized covariance matrix. What is the rank of the regularized covariance matrix now?

g. Try different values of δ between 0 and 1 and perform eigenvalue decomposition on the regularized covariance matrix. How many eigenvalues are positive? How many are zero?

h. Repeat the exercise in (17.2d.) with the regularized covariance matrix for a few values of δ. Compare the stability of your portfolio weights to what you got with the historical covariance matrix.

17.3 **Portfolio Construction on a Large Asset Universe**:

a. Download historical data from your favorite source for 5 years and at least 100 companies or ETFs. In this problem we will look at the covariance matrix for these assets and its properties.

   i. Clean the data so that your input pricing matrix is as full as possible. Fill in any gaps using a reasonable method of your choice. Explain why you chose that particular method.

ii. Generate a sequence of daily returns for each asset.

iii. Calculate the covariance matrix of daily returns and perform an eigenvalue decomposition on the matrix. How many positive eigenvalues are there? How many were negative? If any are negative, what happened?

iv. How many eigenvalues are required to account for 50% of the variance of the equally-weighted portfolio of the securities you picked? How about 90%? Does this make sense to you?

v. Create the daily residual returns after the principal components that correspond to the 90% eigenvalues have been removed from the equally-weighted portfolio. Plot this return stream and comment on its properties.

b. We defined a Lagrangian for portfolio with constraints in matrix form by

$$L(w, \lambda) = \langle R, w \rangle - a \langle w, Cw \rangle - \langle \lambda, Gw - c \rangle \tag{P17.4}$$

i. Form the matrix G by imposing the budget constraint, which is $\langle 1, w \rangle = 1$, and another constraint that allocates 10% of the portfolio to the first 17 securities (to simulate sector allocation). Using C from Problem 1, use your favorite method and the software package of your choice to invert $GC^{-1}G^T$ in a nice, stable way. (Hint: consider my favorite method).

ii. What does the resulting portfolio look like? Would it be acceptable to most mutual funds? If not, what would you to do to fix that?

17.4 Consider the following set of ETFs:

ETF Ticker	Description
SPY	S&P 500
VXX	Volatlity ETN
HYG	High Yield
DBC	Commodities
TLT	Long Maturity Treasuries

a. Calculate a time series of returns of each assets as well as an equally weighted portfolio of the five assets. Is this equally weighted portfolio appealing?

b. Construct an inversely weighted portfolio of the five assets and compare to the equally weighted portfolio. Which assets are weighted more or less than in the equally weighted case?

c. Calculate a covariance matrix of returns for the set of ETFs. Plot the rolling three-month pairwise correlations as well. Which assets have the highest and lowest correlation? What appears to be the best diversifier? Are the correlations stable in time?

d. Using this covariance matrix, build a risk parity portfolio using the same set of ETFs. Compare this to the inverse volatility weighted portfolio and comment on how correlation between the asset class impacts the weights.

e. Calculate the volatility of the risk parity portfolio and compare it to the volatility of the equally weighted portfolio. Explain how you would lever up or down the risk parity portfolio so that the volatility in the leveraged risk parity portfolio matches the equally weighted portfolio. What is the leverage required?

17.5 Consider the following set of country ETFs:

ETF Ticker	Country
SPY	US
EWA	Australia
EWC	Canada
EWD	Sweden
EWG	Germany
EWH	Hong Kong
EWI	Italy
EWJ	Japan
EWK	Belgium
EWL	Switzerland
EWN	Netherlands
EWO	Austria
EWP	Spain
EWQ	France
EWS	Singapore
EWU	United Kingdom
EWW	Mexico
EWZ	Brazil

a. Download data for the country ETFs listed above and compute a correlation matrix of daily and monthly returns.

b. Construct the fully-invested efficient frontier using these ETFs with no additional constraints.

c. Construct the efficient frontier again assuming that the portfolio must be long-only and the max position size is 10%.

d. Download data for the VIX index. Create a conditioning variable that defines the market regime as follows:
   - High Volatility Regime: VIX ≥ 20
   - Normal Volatility Regime: VIX < 20

   Calculate the correlation matrix of daily and monthly returns for both the high volatility and normal volatility regimes. Comment on the difference between the correlation structure as well as the deviation from the unconditional correlation matrix.

e. Using a subjective probability of being in a high regime of 25%, calculate a

modified correlation and covariance matrix. How does this compare to the unconditional?

f. Construct the fully-invested efficient frontier again using this regime weighted covariance matrix. Compare to the results obtained using the unconditional covariance matrix and comment on any notable differences.

17.6 **Black-Litterman in Practice**:

a. Download price and market capitilization data for the sector ETFs in the S&P 500 defined in 17.1. Calculate the market cap weights of the sectors.

b. Using the historical returns and covariance matrix, calculate the unconstrained mean-variance optimal portfolio for a given risk aversion parameter, λ. Explain how you chose a value for λ.

c. Shift the historical returns for each asset up and down by 1% respectively. Comment on the changes to the optimal portfolio weights.

d. Extract a set of equilibrium returns implied by the market cap weights.

e. Calculate the Q and P matrix respectively such that the view specified for each sector ETF is its average historical return over the period.

f. Within the Black-Litterman model, describe how you would determine the additional parameters, such as the risk aversion parameter, the uncertainty in each view and τ the scaling coefficient.

g. Using the market implied equilibrium returns, and the P and Q matrices, and the additional parameters estimated above, calculate a set of optimal weight that satisfy this Black-Litterman problem. Compare these

h. Shift all input returns in the Q matrix up and down by 1% respectively. Comment on the changes in the optimal portfolio weights relative to the changes in a mean-variance implementation with similar shifts.

CHAPTER 18

Modelling Expected Returns and Covariance Matrices

18.0.1 Coding Example: Regularization Models

In this coding example we show how Lasso and Ridge regression techniques can be applied in Python. The reader should notice that in this example the regularization parameter α is set to a fixed amount, however, in practice this regularization parameter may be tuned to optimize performance. Methods for performing this type of cross validation are explored further in chapter 21.

```
from sklearn.linear_model import Ridge, Lasso
import numpy as np

np.random.seed(0)
x = np.linspace(0, 10, 20)
epsilon = np.random.randn(20)
y = 3 * x + epsilon

ridge = Ridge(alpha=0.1)
ridge.fit(x[:, np.newaxis], y)
print(ridge.coef_, ridge.intercept_)  # [2.88471143]
    1.1457774515537391

lasso = Lasso(alpha=0.1)
lasso.fit(x[:, np.newaxis], y)
print(lasso.coef_, lasso.intercept_)  # [2.87542027]
    1.1922332348198434
```

18.0.2 Coding Example: Calculating Rolling IC

In this coding sample we show how to calculate a rolling information coefficient, by calculating the rolling correlation between our signal and future returns over a given horizon:

```
import matplotlib.pyplot as plt
from pandas_datareader import data

```

```
df_px = data.get_data_yahoo(('CAD=X'), start='2010-03-01', end='
    2021-02-28').loc[:, 'Adj Close'].to_frame()
df_rets = df_px.pct_change(1).dropna()
df_signal = df_rets.shift(10).dropna()

def calcRollingIC(df_rets, df_signal, roll_window_length):
    roll_ics = df_rets.rolling(roll_window_length).corr(df_signal)

    print(roll_ics.mean())

    plt.plot(roll_ics)
    plt.show()

    plt.plot(roll_ics.rolling(100).mean().dropna())
    plt.show()

    plt.plot(roll_ics.cumsum().dropna())
    plt.show()

    return roll_ics

calcRollingIC(df_rets, df_signal, 10)
```

18.0.3 Coding Example: Calibration of GARCH(1,1) Model

In this coding example we show how maximum likelihood can be used to estimate the parameters in a GARCH(1,1) model:

```
from pandas_datareader import data
from scipy.stats import norm
from scipy.optimize import minimize, Bounds, LinearConstraint

df_px = data.get_data_yahoo(('SPY'), start='2000-03-01', end='
    2021-02-28').loc[:, 'Adj Close']
df_rets = df_px.pct_change(1).dropna()
df_rets

alpha = 0.2
beta = 0.2
gamma = 0.25
sigma_0 = 0.1

#calculate GARCH coefficients using Maximum Likelihood
def garchLogLikelihood(params, df_rets):
    alpha = params[0]
    beta = params[1]
    gamma = params[2]
    sigma_0 = params[3]

    logLikeli = -len(df_rets)*np.log(math.sqrt(2*math.pi))
    sigma_i = sigma_0
    ret_i = df_rets[0]
    for ii in range(2, len(df_rets)):
        sigma_i = alpha + beta * ret_i + gamma * sigma_i
```

```
        ret_i = df_rets[ii]

        logLikeli -= (0.5 * np.log(sigma_i**2) + 0.5*(ret_i**2/
    sigma_i**2))

    return -logLikeli

res = minimize(garchLogLikelihood,
               x0 = [alpha, beta, gamma, sigma_0],
               bounds  = ((0.0, None),(0.0, None), (0.0, None), (0.0,
     None)),
             args = (df_rets),
             tol = 1e-12,
             method = 'SLSQP',
             options={'maxiter': 2500, 'ftol': 1e-14})

print(res.message)
print(res['x'])

#calculate GARCH coefficients using arch package
garch = arch_model(rets, vol='GARCH', p=1, q=1)
garch_fitted = garch.fit()
print(garch_fitted)
```

As the reader can see in the coding example, the initial volatility, σ_0, is treated as a model parameter in the GARCH calibration.

18.0.4 Coding Example: Shrinking a Covariance Matrix

In this coding example we show how shrinkage can be applied to a covariance matrix. We assume a target or normalizing matrix that has constant correlation between assets, and allow a parameter which varies the amount of shrinkage applied:

```
def covMatShrinkage(cov_mat, vols, const_corr, phi):
    N = cov_mat.shape[0]

    #make the constant correlation covariance matrix with the proper
    vols
    const_corr_mat = np.full((N, N), const_corr)
    np.fill_diagonal(const_corr_mat, 1.0)

    vols_mat = np.diag(vols)
    const_cov_mat = vols_mat.T @ const_corr_mat @ vols_mat

    #combine the empirical and constant correlation covariance matrix
    cov_mat_hat = (1 - phi)* cov_mat + phi * const_cov_mat

    return cov_mat_hat
```

EXERCISES

18.1 **Information Coefficient in Practice**:

a. Download data for the ETF GLD and clean as appropriate.

b. Build an expected return model for future one-month GLD returns that uses the prior one-month return and the prior twelve month to one-month return as signals. This signals mimic the postulated reversal and momentum definitions in [2].

c. Using the entire period of analysis, estimate the coefficients on the reversal and momentum signals. Are the coefficients in line with your expectations?

d. For each period, calculate the expected return of GLD and the ensuing, one-month forward GLD returns. Calculate the information coefficient for the signals by computing the entire period correlation between expected returns and forward returns. Using this correlation, comment on the attractiveness of the signal.

e. Compute the same information coefficient over all rolling one-year periods and plot the cumulative correlation. Does this indicate any particular periods where the strategy worked well or didn't work?

18.2 **Regularization models in Practice**:

a. Consider the following set of sector ETFs:

Ticker	Description
XLB	Materials
XLE	Energy
XLF	Financials
XLI	Industrials
XLK	Technology
XLP	Consumer Staples
XLU	Utilities
XLV	Healthcare
XLY	Consumer Discretionary

Download the appropriate data for these sector ETFs as well as the Fama-French Factors and clean as needed.

b. For each sector ETF, estimate the exposures to the Fama-French factors by running a contemporaneous regression of the sector ETF returns against the factor returns. Make a table of these exposures and comment on anything noteworthy.

c. Recalculate these exposures using Ridge regression with varying values for the parameter λ. How does this change the coefficients? Does it make them more or less consistent across sector?

d. Now use Lasso regression, again with different values for λ and discuss how the coefficients change. What factors seem to be removed first for each sector ETF as λ increases? Is there a pattern across sectors?

18.3 **Fama-MacBeth in Currency Markets**:

a. Consider the following set of currency pairs:
   - AUDUSD
   - USDCAD
   - EURUSD
   - USDJPY
   - NZDUSD
   - USDNOK
   - GBPUSD
   - USDSEK
   - USDCHF

   Download historical data for each currency pair.

b. Build an expected return model that forecasts the next period return as a function of the prior period's return. Try different values for the forecast horizon and lookback window and comment on which values you believe are optimal.

c. Estimate a regression model for each currency pair using the expected return model with the optimized forecast horizon and lookback window. Create a table of coefficients for each currency pair and comment on how consistent they are. Are the signs consistent?

d. Build a panel regression model with fixed effects and compare the results to the results obtained above. Comment on the intercept terms you get for each currency pair.

e. Employ the Fama-Macbeth technique to estimate the expected return model cross-sectionally and over time. Comment on how this impacts the coefficient estimates and standard errors.

18.4 **Realized Volatility Estimators**:

a. Download data for the ETF ACWI and clean the data for splits and other anomalies. In addition to closing price data, download data for the open, high and low. Clean these data as well and ensure they are in line with the closing levels.

b. Using an expanding window, calculate the realized volatility of the ETF over all periods with at least one-year of data.

c. Calculate the realized volatility using a rolling window of overlapping observations and a one-year lookback window.

d. Now consider an EWMA volatility estimate that using an expanding window and a one-year half-life. Calculate the rolling realized volatility using this model over all overlapping one-year periods. Discuss the difference between

the rolling window approach and the EWMA approach with a similar half-life.

e. Finally, use the open, high, low and close data to estimate a Garmin-Klass range based volatility estimate. Create a realized volatility time series using this range based volatility estimate.

f. Plot the time series of volatility estimates and comment on the difference in behavior of the three approaches. Discuss how this relates to the strengths and weaknesses of each model and which one you would use in practice. Whic is most stable? Which is most dynamic?

18.5 **GARCH Model Estimation**:

a. Download data for the ETF SPY and calculate rolling one-month, three-month and one-year realized volatility estimates. Plot these estimates and comment qualitatively on the series. Do they appear mean-reverting? Are they structurally similar?

b. Using Maximum Likelihood estimate a GARCH(1,1) to SPY returns. Comment on the coefficients that you obtain and what that implies about the level of mean-reversion of volatility in SPY.

c. Re-estimate the parameters on a rolling basis with a given rolling window length of τ. Comment on how you choose τ and why you expect it to be reasonable. Are the estimated GARCH coefficients consistent over time?

d. Download data for the ETF VXX and calculate an analogous set of rolling one, three and twelve-month realized volatilities. Plot these estimates and comment on how similar or different they are relative to the SPY charts above.

e. Describe conceptually what phenomenon we are exploring when modeling the volatility of an ETF that is already linked to volatility. Does this lead to conceptual differences that should manifest in a volatility model such as GARCH?

f. Using MLE estimate a GARCH(1,1) model on VXX returns. Compare the coefficients to those obtain for SPY and try to explain any fundamental differences.

18.6 Consider the following set of ETFs:

ETF Ticker	Country
SPY	US
EWA	Australia
EWC	Canada
EWD	Sweden
EWG	Germany
EWH	Hong Kong
EWI	Italy
EWJ	Japan
EWK	Belgium
EWL	Switzerland
EWN	Netherlands
EWO	Austria
EWP	Spain
EWQ	France
EWS	Singapore
EWU	United Kingdom
EWW	Mexico
EWZ	Brazil

a. Calculate an empirical covariance matrix of the country ETFs and perform PCA analysis to determine the rank of the matrix. Comment on how many principal components appear significant.

b. Introduce shrinkage into the covariance using equation (**??**) with different values of δ ranging from 0 to 1 and a diagonal covariance matrix as the normalizing matrix. How does the rank of the matrix improve as the shrinkage coefficient, δ is increased? How much shrinkage appears to be needed to ensure the matrix is full rank?

c. Create a constant correlation matrix with correlation $\rho = 0.8$ between all country ETFs and volatilities that match the historical data set. As before begin to introduce shrinkage into the model. Does it still improve the rank of the matrix? If so is more or less shrinkage required? Comment on why this is different than the diagonal case above.

d. Repeat this exercise for different values of ρ and δ and comment on how varying them impacts the rank of the matrix. Are there any restrictions you would apply to the specification of ρ in the constant correlation matrix, or δ in the shrinkage estimator, when using this technique in practice?

CHAPTER 19

Risk Management

19.0.1 Coding Example: Calculating Common Risk Metrics

In this section we show the Python code for computing common risk metrics discussed above:

```
df_ret_port = df_ret.mean(axis=1)
eps = 0.05

vol = df_ret_port.std()*np.sqrt(252)   # annualized volatility
down_vol = df_ret_port[df_ret_port<0].std()*np.sqrt(252)   #
    annualized downside deviation
VaR = df_ret_port.quantile(eps)   # daily VaR
CVaR = df_ret_port[df_ret_port<=VaR].mean()   # daily CVaR
```

As we can see above, we start with a return data frame and then compute annualized volaitlity, downside deviation [1] and daily VaR and CVaR. The reader should note that the VaR and CVaR quantities are not annualized. In order to conduct this analysis, we leverage the same return dataset as the efficient frontier example in . The reader should also note that the calculations are based on an equally weighted portfolio, and specify a 5% confidence level for the VaR and CVaR calculations via the variable *eps*.

19.0.2 Coding Example: Calculating Value at Risk via Bootstrapping

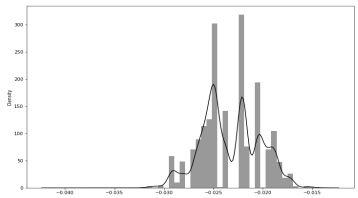
In this example, we define a VaR function that takes a historical returns data frame and a confidence level ϵ. In this function, bootstrapping is used to simulate returns for assets within the portfolio. After generating each new path, we calculate the VaR the same way as we did in **??**.

```
import random

def var_boot(df_ret, eps, paths, pathLength):

    N = df_ret.shape[0]
    n = df_ret.shape[1]
    ret_simu = np.zeros((paths,n))
```

[1] Based on daily returns

```
8
9       for i in range(paths):
10          idx_simu = random.sample(range(N), pathLength)
11          path_rets = np.zeros((pathLength,n))
12          for j in range(pathLength):
13              ret_simu[i,:] += df_ret.iloc[idx_simu[j],:]
14
15      df_simu = pd.DataFrame(ret_simu)
16      df_simu_port = df_simu.mean(axis=1)
17      VaR = df_simu_port.quantile(eps)
18
19      return VaR
```

Note that again we assume an equally weighted portfolio in our VaR calculation. It should be emphasized that every time we call this function, a new set of return paths will be generated, which results in a new VaR estimate. Each such estimate is statistically equivalent, but will naturally vary because of the randomness in the simulated paths. By running the algorithm repeated times, we can get a sense of the distribution of these estimates. For example, the plot below shows the density of 10,000 simulated VaR's at a 0.05 confidence level, using two years of historical ETF data:



The reader might notice that this distribution of VaR estimates is not as smooth as they might expect. The reason for this is two-fold. First, in this example, we bootstrap a daily VaR. This means that each bootstrapped path consists of a single historical return. A byproduct of this is that the possible set of simulated returns is limited to the historically observed set of returns. In traditional applications of bootstrapping we would stitch together many returns in the underlying dataset on each bootstrapped path, and this would no longer be the case. Secondly, in this calculation a relatively short historical window is used, further limiting the set of potential outcomes. The reader is welcome to relax these assumptions by trying a multi-day VaR or using a longer historical calculation window and experiment with how the distribution of VaR estimates becomes smoother.

EXERCISES

19.1 Describe in detail how you would build a risk model for each type of instrument listed below. Be explicit about the assumptions you are making and why they are appropriate.

a. A long-short equity portfolio of 100 assets

b. A portfolio of Corporate bonds

c. A single exotic option

d. A book of European Options in Equities, FX and Commodities

19.2 Describe three ways that you would validate a VaR or CVaR model to ensure it is robust and accurate.

19.3 **Portfolio Risk Management**:

a. Consider an equally weighted portfolio of the following assets:

Ticker	**Description**
SPY	US Equity
DBC	Commodities
SVXY	Short Volatility
HYG	High Yield
EEM	Emerging Market Equity
EAFE	Europe and East Asia Equity
AGG	US AGG
IAGG	International AGG

b. Download daily historical data for the last ten years for the assets. Clean as appropriate.

c. Using a two-year rolling window and the historical simulation method discussed in the chapter, calculate a daily VaR and CVaR for the portfolio using a 5% threshold. Plot both on the same chart and comment on their relationship to each other, as well as their evolution over time.

d. Plot the VaR against the returns of the portfolio. Comment on how often the VaR is breached.

e. Calculate the total VaR contribution for each asset by setting each weight to zero and re-calculating the VaR. Comment on which assets seems to be contributing the most and least to the total VaR.

19.4 **Risk Management in Rates**:

a. Download data for 2y, 5y, 10y and 30y treasury yields and swap rates. Clean the data as you see fit.

b. Build a historical simulation of one-month forward treasury yields of all maturities. Comment on the fifth percentile of the distribution of yields.

c. Assume that you purchased the 2y, 5y, 10y and 30y treasuries at the latest available yield and that the coupon is equal to the yield. Estimate the change in price to each treasury along each path based on the change in yields. Plot the changes to the treasury prices for each maturity over all paths.

d. Calculate both VaR and CVaR for each maturity treasury, as well as an equally weighted portfolio that includes all of them.

e. Build a comparable historical simulation of one-month forward swap rates. How does the magnitude of the movements in treasury yields compare to that of swap rates?

f. Assume you entered a swap contract at the current par swap rate (for the latest available date). Calculate the implied mark-to-market changes in the value of the swap along each path.

g. Using these quantities, calculate both VaR and CVaR for each swap.

19.5 **Risk Management of Options**:

a. Consider an options portfolio with the following positions and hedges:

Ticker	**Structure**	Shares
SPY	Short one-month at-the-money Put	100
SPY	Long stock	50
VIX	Long three-month Put	100
VIX	Short one-month Put	100
TLT	Long one-month at-the-money Put	100
TLT	Long one-month at-the-money Call	100

Download data for each underlying asset and clean as needed.

b. Looking at this portfolio, what does the investor want to happen? What are the biggest risks for this investor? Can you explain this portfolio in terms of its Greek exposures?

c. Generate a time series of realized volatility for each underlying asset, which will be used as a proxy for implied volatility. Plot the time series and comment on their shape.

d. Assuming a flat volatility surface as defined by the realized volatility series that you calculate, use historical simulation to estimate VaR and CVaR for this options portfolio.

19.6 **Stress Tests**:

a. Download data for all 500 S&P constituents. Generate a random portfolio of the underlying constituents.

b. Consider the following set of stress tests:

Stress	**Description**
Equity Drawdown	Equities Sell-off by 20%
Yield Curve Steepens	Long End rates sell-off by 100 bps
Dollar Sell-off	A -3 standard deviation move in USD vs. all currencies
Equity Rotation	A +5 standard deviation move of the Fama-French value factor

Carefully describe the correlation assumptions that you would make in order to translate shifts to these variables into shocks to the S&P constituents. Calculate the value of the portfolio in each stress test. Which is the worst? Why?

c. Repeat this exercise many times and comment on how the value of the portfolio in the stress test changes among random paths.

19.7 **Monte Carlo Simulation Based VaR & CVaR**

a. Consider the following asset universe:

Ticker	Description
SPY	US Equity
DBC	Commodities
VXX	Volatility
LQD	Investment Grade Bonds
ACWI	Global Equities
AGG	US AGG
IAGG	International AGG
UNG	Natural Gas

Download data for this set of ETFs from a source of your choice and validate the data for anomalies.

b. Calculate an inverse volatility weighted portfolio of the assets using the entire period in your volatility calculation.

c. Calculate a vector of expected returns and a covariance matrix of the underlying assets.

d. Test the individual return series to see if they are Gaussian, using a QQ plot or another technique of your choice. Does it appear the returns are normally distributed?

e. Perform an eigenvalue decomposition of the covariance matrix, Is it positive definite? Positive semi-definite? Explain how you know.

f. Implement the Monte Carlo based approach to calculating VaR and CVaR assuming the joint distribution is Gaussian. Comment on any assumptions that you believe may be violated and how you think that might impact the results.

19.8 Discuss how you would incorporate undefined risks into a risk management process and explain the pros and cons of integrating this into your model.

CHAPTER 20

Quantitative Trading Models

20.0.1 Coding Example: Pairs Trading Algorithm

In this coding example we provide the basic structure for implementing a pairs trading strategy. In this code we can see that we initially develop the optimal hedge ratio between two specified assets, and then check for stationarity of the spread using the augmented Dickey Fuller test. We then proceed to iterate through the data and trade the spread when a specified z-score threshold is breached:

```
import statsmodels.tsa.stattools as stools

def pairs_trade(series1, series2, lookWindow, tradeThreshold):
    """
    series1 and series2: numpy arrays or pandas series containing the
     prices of the two assets
    lookWindow (int): the window length of the used to calculate the
    rolling mean and standard deviation of the ratio
    tradeThreshold (float): the z-score (abs value) above which we
    would trade
    """
    regr = linear_model.LinearRegression(fit_intercept=False)
    s1 = series1.values.reshape(-1,1)
    s2 = series2.values.reshape(-1,1)
    regr.fit(s1, s2)

    spread = pd.DataFrame(s1 * regr.coef_ - s2, index=series1.index)
    adf, pvalue = stools.adfuller(spread)

    rollSpreadMean = spread.rolling(lookWindow).mean()
    rollSpreadStd = spread.rolling(lookWindow).std()
    rollSpreadZ = (spread - rollSpreadMean) / rollSpreadStd

    pnls = np.zeros(spread.shape[0]-1)
    capital = 0
    pos1 = 0
    pos2 = 0

    for day in range(spread.shape[0] - 1):
        if rollSpreadZ.values[day] < - tradeThreshold: #buy spread
            pos1 += 1
            pos2 += - spread.values[day]
```

```
            capital += spread.values[day] * series2[day] - series1[
    day]
        elif rollSpreadZ.values[day] > tradeThreshold: #sell spread
            pos1 += -1
            pos2 += spread.values[day]
            capital += series1[day] - spread.values[day] * series2[
    day]
        pnls[day] = capital

    #clear the positions on the last day
    capital += pos1 * series1[-1] + pos2 * series2[-1]

    pos1, pos2 = 0, 0
    return capital, pnls, rollSpreadZ
```

It should be emphasized that in this example we leave out many of the practicalities of a pairs trading algorithm described above for simplicity. For example, we neglect to incorporate a stop-loss criteria and calibrate a single optimal hedge ratio over the entire period, thus looking ahead. Extending this piece of code to be suitable for practical use is an exercise that is left to the reader.

20.0.2 Coding Example: PCA Analysis

In this coding example we detail how PCA analysis can be conducted in Python using numpy's built-in Eigenvalue Decomposition function:

```
def pcaAnalysis(stockReturns):
    """
    stockReturns: Pandas DataFrame of stock returns
    """
    corrMat = stockReturns.corr()
    eigenvalues, eigenvectors = np.linalg.eig(corrMat)
    explainedVarianceRatio = eigenvalues/np.sum(eigenvalues)
    return eigenvalues, eigenvectors, explainedVarianceRatio
```

20.0.3 Coding Example: Momentum Strategy

In this coding example we show how a simple momentum signal could be incorporated in Python code:

```
def momentumStrategy(monthlyPrices, n):
    """
    monthlyPrices (pandas dataframe): dataframe of end of month
    prices for stocks
    n (int): the number of top stocks to buy/ bottom stocks to sell
    """
    nStocks = len(monthlyPrices.columns)
    monthlyReturns = monthlyPrices.pct_change(1)
    annualReturns = monthlyPrices.pct_change(12)
    #computing the signal as annual minus monthly return
    momentumSignal = (annualReturns - monthlyReturns).dropna()
    #ranking in each row in descending order of signal
    rankedSignal = momentumSignal.rank(axis = 1, ascending = False)
```

```
    #names of the stocks that are ranked in the top n
    longs = rankedSignal.apply(lambda x: x.index[x <= n].tolist(),
    axis = 1)
    longs.name = 'long'
    #names of the stocks that are ranked in the bottom n
    shorts = rankedSignal.apply(lambda x: x.index[x  > nStocks - n ].
    tolist(),axis = 1)
    shorts.name = 'short'
    #putting longs and shorts into a dataframe
    longShortDf = pd.concat([longs, shorts], axis = 1)
    return longShortDf
```

In this example, we consider the case where we buy the assets with the top N prior period returns and sell assets with the lowest prior period returns. In this example, we show how the highest and lowest returns can be identified, and mark the position in each asset that would be taken. We then leave it as an exercise for the reader to turn this into a back-test quantitative strategy.

20.0.4 Coding Example: Covered Calls Back-test

In this section we provide a coding example for a simplified options back-test for covered calls. Recall from chapter 11 that covered calls are popularly traded options structures, especially by retail investors. A covered call is defined by a long position in the underlying asset and a corresponding short position in a call option, therefore selling upside but potentially collecting a volatility premium.

In the following coding example, we show how a simple options back-test can be incorporated in Python:

```
from pyfinance.options import BSM as BlackScholes

def coveredCallsBacktest(priceAndVolData):
    """
    priceAndVolData: DataFrame with columns giving the price of the
    security and the historical market volatility (in that order)
    """
    priceAndVolData['CallPx'] = BlackScholes(S0 = priceAndVolData.
    iloc[:,0],
                                        K = priceAndVolData.iloc
    [:,0], T= 1/12,
                                        r=.01, sigma=priceAndVolData.
    iloc[:,1]/100).value()
    priceAndVolData['CoveredCallPayoff'] = np.nan
    priceAndVolData['Profit/Loss'] = np.nan
    priceAndVolData['PnL'] = 0

    for i in range(1,len(priceAndVolData)):
        underlyingPayoff = priceAndVolData.iloc[i,0] -
    priceAndVolData.iloc[i-1,0]
        callPayoff = np.maximum(priceAndVolData.iloc[i,0] -
    priceAndVolData.iloc[i-1,0], 0)
        priceAndVolData.iloc[i,priceAndVolData.columns.get_loc('
    CoveredCallPayoff')] = underlyingPayoff - callPayoff
        priceAndVolData.iloc[i,priceAndVolData.columns.get_loc('
```

```
    Profit/Loss')] = priceAndVolData['CoveredCallPayoff'][i] +
    priceAndVolData['CallPx'][i-1]
19      priceAndVolData.iloc[i,priceAndVolData.columns.get_loc('PnL')
    ] = priceAndVolData['Profit/Loss'][i] + priceAndVolData['PnL'][i
    -1]
20
21  return monthlyData
```

An important caveat of this back-testing code is that, for simplicity and brevity we consider the value of the structure only when they are initially traded and then at expiry. This simplifies the code greatly but in practice a daily mark-to-market for the structure should be incorporated, as should a more robust re-balancing / rolling strategy. The reader is encouraged to think about how this code could be generalized to incorporate these features, as well as different options payoff structures.

EXERCISES

20.1 Describe how you would choose between a single-stock model, a cross asset auto-correlation model, pairs trading or a PCA based factor model. Explain the alpha research process you would follow and how you would avoid overfitting.

20.2 **Judging Performance of Strategies**:

a. Write a set of Python functions that, given a back-tested set of returns calculate the list of performance statistics defined in **??**.

b. Consider a strategy of investing in a standard 60% equity 40% fixed income. Using the above function compute a performance summary of this portfolio.

c. Repeat this analysis with equity allocations ranging from 20% to 80%. Which equity allocation leads to the most compelling performance metric.

d. Consider an overlay to this strategy which increases the equity weight in the portfolio by 10% if the VIX index is above 25. Re-calculate the performance statistics for this strategy. What do you think of this overlay signal? Would you do it? Why or why not?

20.3 **Momentum Trading**:

a. Download data for 100 single-name stocks of your choice. These stocks may be members of an index or chosen in some other way. Comment on why your methodology for choosing stocks may or may not lead to bias in a back-test.

b. Perform any necessary data quality checks on the data and handle outliers or anomalous data as needed.

c. Back-test a strategy that is long the five stocks with the highest returns over the past twelve months. Try this with and without the most recent months return. Is this strategy profitable? Does removing the last month improve performance?

d. Now consider varying the number of stocks in your portfolio. Try 10, 20 & 30. Do the results deteriorate as we add stocks? Why or why not?

e. Next lets consider the short side of a momentum strategy. Back-test a strategy that is short the five lowest momentum stocks using the same twelve month lookback, with and without the previous one-month. How do the results of the short side compare to what you obtained above on the long side? Do most of the profits in a long-short momentum strategy appear to come from longs, shorts, or is it mixed?

f. Finally, vary the number of stocks included on the short side. Does perform suffer as we add stocks to the short side?

20.4 **Fama French Residual Analysis**:

a. Download historical data for the Fama-French factors as well as the S&P sector ETFs and clean the data for anomalies.

b. Explain how you would create a portfolio with high exposure to a single Fama-French factor, size, with minimal exposures to the other factors.

c. Test the Fama-French factors for autocorrelation. Explain how you would build a sector ETF strategy that would take advantage of autocorrelation in the factor returns. Back-test this strategy on the Fama-French factors. Is it profitable?

d. Using the set of Fama-French factors, calculate the residuals for each sector ETF. Test these residuals for autocorrelation. Back-test a strategy based on the z-score of the residuals and takes a long positions when $\hat{z} < -x$ and a short position when $\hat{z} > x$. How does this strategy do?

20.5 **Pairs Trading**:

a. Download data for Gold and Gold Miners, cleaning as appropriate.

b. Using an expanding window, calculate whether Gold and Gold Miners are stationary as a pair. Are the results consistently statistically significant? Compare these results to a single stationarity test including the entire dataset.

c. Using the static pairs model that relies on the entire dataset, use a z-score methodology and a threshold of +/- 1 as entry signals, back-test Gold and Gold Miners as a pair. Does it work?

d. Describe how you would update your pairs trading algorithm to benefit from convergence of the spread while also being long volatility? Short volatility?

20.6 **PCA Strategies**:

a. Download data for the constituents of the S&P 500 and identify and handle any data anomalies.

b. Using the entire period, perform an eigenvalue decomposition of the correlation matrix of returns. How many eigenvalues are significant? What do the first few eigenvalues seem to represent?

c. Using rolling five-year periods, calculate the eigenvalue decomposition and plot a time series of the percentage of the variance explained by each eigenvalue over time. Is it consistent? When does it seem to change the most?

d. Describe how you would identify a set of principal components to use as the factors in a PCA trading model.

e. Calculate a time series of residuals for each asset and test them for mean-reversion. Comment on the magnitude and consistency of the results.

f. Develop a strategy that uses a z-score methodology with a threshold of 2 to trade the residuals of this PCA model. Comment on the performance of the strategy.

g. Compare this approach to the more naive approach of using sector ETF return as the factors. Comment on the number of eigenvalues included vs. the number of sectors in the market.

20.7 **Options Strategies**:

a. Download data for the SPY ETF as well as the VIX index.

b. Plot the level of VIX against one-month, three-month and six-month realized volatility. If we assume VIX is a proxy for implied volatility, what does this tell you about an implied vs. realized premium? What factors might bias this calculation?

c. Assuming the volatiilty surface is flat with its implied volatility specified by the level of VIX. Construct a strategy that sells an at-the-money straddle on SPY every day and holds the position until expiry, with no intermediate hedging. Explain the factors that determine the profit or loss of the strategy, and comment on the distribution of returns for this strategy.

d. Next, consider a strategy that sells a 25-delta put every day and holds it until expiry. How does this compare to the short straddle structure above?

e. Finally, consider a strategy that enters a new butterfly contract every day that is centered on the at-the-money strike, with a reasonable width of your choice. How does this strategy compare to the previous two?

f. Comment on which strategy you would prefer to invest in and why. Also comment on any biases introduced in this analysis and how you would handle them.

20.8 **Leveraged ETFs**:

a. Download data for the ETFs DUST (3x Inverse) and NUGT (3x Long), both triple leveraged gold ETFs. Check the data for splits and other anomalies.

b. Plot the data and comment on the time series

c. Build a strategy that is short both DUST and NUGT in equal market value and re-balances monthly. How does this strategy perform throughout the back-test? When does it profit? When does it lose? Explain why.

d. Comment on how any missing factors might impact this strategy's performance.

e. Compare this strategy on leveraged ETFs to an options structure. What options structure has the most similarities? Why?

CHAPTER 21

Incorporating Machine Learning Techniques

21.0.1 Coding Example: Optimizing Model Parameters via Cross Validation

Python has many libraries that provide functions and methods for implementing cross validation in practice. To give the reader an example, this coding example shows how to optimize a LASSO model using sklearn's **cross_validate** function[1], where we try to find the optimal value of the parameter alpha for this model.

```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_validate

diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]

alpha_list = [0.0, 0.1, 0.2, 0.5, 1.0]
score_list = []

for alpha in alpha_list:
    model = linear_model.Lasso(alpha=alpha)
    cv_results = cross_validate(model, X, y, cv=5)
    score = cv_results['test_score'].mean()
    score_list.append(score)

opt_idx = np.argmax(score_list)
print('best model parameter: alpha={}'.format(alpha_list[opt_idx]))
```

21.0.2 Coding Example: Clustering

This coding example shows how to implement k-means clustering using the **sklearn** package. The number of clusters, k is a hyperparameter that may need tuning to achieve the optimal model performance. This could be done, for example, via cross validation.

[1] The original example is from sklearn's documentation.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=n, random_state=0).fit(data)
labels = kmeans.labels_
centers = kmeans.cluster_centers_
```

In the above code we can see that the **labels_** attribute indicates what cluster each observation belongs to. Similarly, the center of each cluster can be accessed through the **cluster_centers_** attribute. The number of centers coincides with k, or **n_clusters**.

21.0.3 Coding Example: K-Nearest Neighbor Algorithm

In this coding example, we introduce the implementation of k-nearest neighbor using **sklearn**. The number of neighbors is a hyperparameter that may need tuning to achieve the optimal model performance, for example via cross validation. Using the **predict()** method, we can use the trained model to predict the class of a new datapoint, with the probability of each class given by **predict_proba()**.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=n).fit(X_train, y_train)
knn.predict(X_test)
knn.predict_proba(X_test)
```

21.0.4 Coding Example: Classification via SVM

This coding example shows a general SVM algorithm utilizing **sklearn**. We pass the training data to the initialized model and fit it. We can access the support vectors using the **support_vectors_** attribute. To make predictions for new datapoints, simply pass them to the trained model using the **predict()** method. Note that the function **svm.SVC()** provides multiple parameters (here we just show the default model) for choosing different regularization levels, kernel functions, etc. These parameters may need to be tuned to achieve optimal model performance. Readers are free to play around with them.

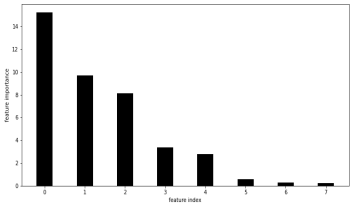
```
from sklearn import svm
model = svm.SVC()
model.fit(X_train, y_train)
model.support_vectors_
model.predict(X_test)
```

21.0.5 Coding Example: Feature Importance Tools

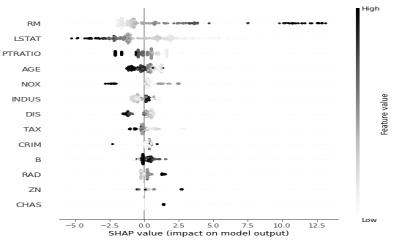
In this example, we introduce several Python packages that are widely used in the context of feature importance.

The first is sklearn, which we have been using throughout this chapter. In addition to providing various types of machine learning models, sklearn allows users to easily access the feature importance scores by calling certain model attributes. For instance, when using logistic regressions, we retrieve the feature importance through the **coef_**

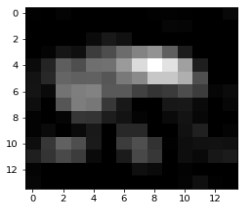
attribute. For decisions trees, calling **feature_importances_** provides similar results. Simple visualization provides a straight-forward demonstration.



SHAP (Shapley Additive Explanations) is a powerful model interpretation tool that usually works with tree ensembles and neural networks. SHAP borrows the idea of Shapley values from game theory and incorporate that into the neural network framework. SHAP works perfectly with common deep learning frameworks such as TensorFlow, Keras, and PyTorch, making feature analysis only one step away from existing models. Take ensemble models for example, SHAP receives a trained model to generate an explainer through the **Explainer()** method. The Shapley values will then be computed by calling the explainer with training data. A sample visualization is provided below. More information about SHAP can be found on the GitHub repository.[2]



For models dealing with images and therefore have visualization needs, Grad-CAM (Gradient-weighted Class Activation Mapping) offers a user-friendly approach that converts feature vectors into visible graphical patterns. It generates a heatmap that measures the importance of all pixels in an image, and we can further combine this heatmap with the original image to understand which part contributes most to the model prediction. A sample heatmap is shown velow. More information about Grad-CAM can be found on CloudCV.[3]



[2]SHAP repository: https://github.com/slundberg/shap
[3]Grad-CAM site: http://gradcam.cloudcv.org/

```
# sklearn: logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
score = model.coef_[0]

# sklearn: decision tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
score = model.feature_importances_

# SHAP: XGBoost
import shap  # remember to install shap first
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(X_train, y_train)
explainer = shap.Explainer(model)
score = explainer(X_train)

# Grad-CAM: neural network
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import backend as K
model = VGG16(weights='imagenet', input_shape=(224, 224, 3))
preds = model.predict(preprocess_input(x))  # x is an image in the
    format of numpy array
label_ind = np.argmax(preds[0])

output = model.output[:, label_ind]
last_conv_layer = model.get_layer('block5_conv3')
grads = K.gradients(output, last_conv_layer.output)[0]
pooled_grads = K.mean(grads, axis=(0, 1, 2))
iterate = K.function([model.input], [pooled_grads, last_conv_layer.
    output[0]])
pooled_grads_value, conv_layer_output_value = iterate([x])
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
heatmap = np.mean(conv_layer_output_value, axis=-1)

heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```

EXERCISES

21.1 **Machine Learning In Practice**:

a. Identify three applications of machine learning in quant finance and describe why you think machine learning is best suited to solve these problems. Describe in detail why ML techniques are superior to their alternatives in these applications.

b. Discuss the main challenges with applying machine learning when building an alpha signal, and how you would attempt to overcome these challenges, if possible.

c. Describe how you would implement machine learning techniques in a quantamental context. That is, describe how you could leverage machine learning algorithms while still retaining interpretability and an underlying economic rationale.

21.2 Discuss the pros and cons of using classification techniques to forecast asset returns instead of regression.

21.3 Match the following list of techniques that you would use to solve the problems below:

Technique
Linear Regression
K-means Clustering
Principal Component Analysis
Support Vector Machines

a. Identifying a set of assets that are most closely related

b. Estimated an expected future return

c. Reducing dimensionality in a covariance matrix

d. Forecasting whether we are in a high or low volatility regime

Use each technique only once and explain why your choice is optimal.

21.4 **Classification in Practice**:

a. Download data for the VIX implied volatility index and clean as needed. Plot the data and comment on the time periods where VIX is most elevated.

b. Download data for the S&P 500 index and calculate returns. Also download data for 30 year and 2 year treasury yields and calculate the 30y to 2y slope of the yield curve.

c. Using k-means, cluster the data contemporaneously including the level of VIX, daily S&P return and the slope of the yield curve. Try for $k = \{1, 2, 3, 4\}$ and use the method of your choice to determine the optimal number of clusters.

d. Comment on the statistical properties within each cluster. What market conditions does each represent?

e. Create a variable that represents high volatility, which is defined as whether the level of VIX exceeds 25. Comment on the conditional returns and yield curve slopes as a function of this indicator variable, including any helpful visualizations.

f. Use this high vol classifier to build a logistic regression model that predicts the probability of being in a high volatility regime in the next one-month as a function of the current yield curve slope and preceeding S&P returns. Do these variables have significant explanatory power over future high volatility periods? Why do you think this is the case?

g. Repeat this exercise with different subsets of the data. Are the coefficients you obtain similar or do they vary greatly? What does this imply about model stability?

h. Discuss three other potential features that you would include as explanatory variables in this classification model.

21.5 **Clustering in Practice**:

a. Download historical data for the S&P sector ETFs and check the data for outliers and other anomalies.

b. Using a correlation distance metric and daily returns, cluster the data into 3 clusters and comment on which sectors end up together.

c. Calculate the average pairwise correlation between the sector ETFs within each cluster and compare this to the average pairwise correlation between ETFs in different clusters.

d. Build a portfolio for each cluster that relies on an inverse volatility weighting.

e. Use a risk parity approach to combine the three cluster portfolios into a single overarching portfolio. Comment on the largest / smallest weights and high-level characteristics of the portfolios.

f. Calculate the same clusters on a rolling basis of all available two-year periods. How consistent are the clusters over time?

g. Repeat the above exercise using a Euclidean distance metric. Are the clusters similar to what you found using a correlation based distance?

21.6 **Comparison of PCA and Clustering**:

a. Download data for the following country ETFs and clean as appropriate:

ETF Ticker	Country
SPY	US
EWA	Australia
EWC	Canada
EWD	Sweden
EWG	Germany
EWH	Hong Kong
EWI	Italy
EWJ	Japan
EWK	Belgium
EWL	Switzerland
EWN	Netherlands
EWO	Austria
EWP	Spain
EWQ	France
EWS	Singapore
EWU	United Kingdom
EWW	Mexico
EWZ	Brazil

b. Calculate a covariance matrix of daily returns for the country ETFs and perform PCA on this covariance matrix. Plot the first three principal components and describe what they represent.

c. Using k-means clustering, split the country ETFs into three clusters. Compare the three clusters that you obtain to the principal components obtained above, emphasizing any commonality.

d. Repeat the PCA exercise over all rolling, overlapping, one-year periods. Plot the rolling weights of the top three principal components. Are they stable over time?

e. Repeat this exercise with the country clusters as well. Compare the stability of the clusters to the stability of the PCs.

f. Discuss what implications these differences in stability would have in practice and what you think are the strengths and weaknesses of each approach..

21.7 **Classification via SVM**:

a. Download data for the US Small Cap ETF IWM. Calculate daily returns, plot these returns and search for any anomalies.

b. Construct a classification signal that is based on the sign of the daily return of IWM. Additionally, construct a reversal and momentum feature based on previous returns. You are free to choose the lookback period of the reversal and momentum features, but should justify why you believe they are optimal.

c. Build an SVM model that classifies one-day forward returns using these

reversal and momentum features. Try different SVM kernels and values of the cost parameter, C. Comment on their impact on the model.

d. Separate the dataset into a training and test period and comment on how you chose these periods.

e. Using a cross-validation technique, such as k-fold cross validation or another built-in technique, find the optimal hyperparameters for the SVM model.

f. Optimize your SVM model over the training set including tweaking the look-back windows of the reversal and momentum features if needed. Once you believe the model has been optimized, try it on the test dataset as well.

g. Print the confusion matrix of the model both during the test and training datasets. How well do you think this model works? What slippage should we expect out of sample?

h. Discuss how you would properly back-test a model such as this in a professionally managed quant strategy.

Bibliography

[1] Kai Lai Chung. *A Course in Probability Theory, Third Edition.* Academic Press, New York NY, USA, 2001.

[2] Tobias Moskowitz Clifford Asness and Lasse Pedersen. Value and Momentum Everywhere. *The Journal of Finance*, 2013.

[3] DQ Nykamp. Introduction to Taylor's theorem for multivariable functions: `http://mathinsight.org/taylors_theorem_multivariable_introduction`.

[4] Git Documentation. Website: `https://git-scm.com/doc`.

[5] Python Programming Database Wikibook. Website: `https://en.wikibooks.org/wiki/Python_Programming/Databases`.

[6] Gilbert Strang. *Calculus.* Wellesley-Cambridge, 2010.