Universiteit Utrecht

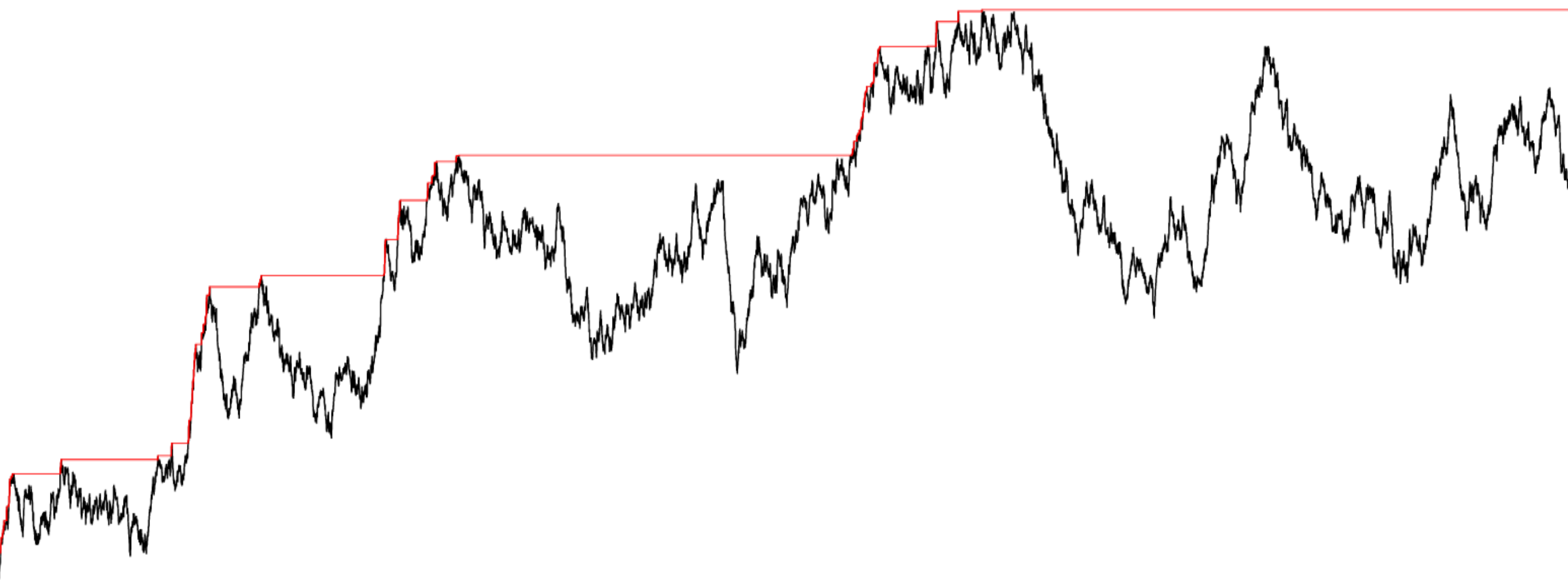
Faculteit Bètawetenschappen

Lookback Option Pricing with the COS Method

BACHELOR THESIS

Jord van Eldik

Wiskunde en Toepassingen



Supervisors:

Prof. Dr. Ir. C.W. Oosterlee

B. Négyesi MSc

June 23rd 2023

Abstract

In this thesis we describe the use of the COS method, a numerical option pricing method based on characteristic functions, developed by Fang and Oosterlee [1]. The novelty in my thesis is the adaptation of this method to price lookback options, an option style highly dependent on the maximum of a chosen stochastic process. The main focus of this thesis is therefore the description of the characteristic function of the maximum of a Geometric Brownian Motion process. To showcase the effectiveness of the COS method we have devised a Python implementation, comparing COS-estimates with both known option values and Monte Carlo simulations. We will find that the COS method is a highly efficient pricing method, far outperforming for example Monte Carlo estimates. As a final chapter this thesis also includes a draft for an alternative method of describing the maximum of a process. This method, applicable to a broad range of stochastic processes known as exponential Lévy processes, recursively defines the characteristic function of a process' maximum. While this chapter is only a proof of concept, this method does give a very promising outlook to a much broader application of the COS method for lookback option pricing.

Contents

1 Introduction

Our story starts in Amsterdam in the year 1602. This time and place was host to one of the most important developments in both Dutch an international financial history, the founding of the VOC. This is because the VOC was the first ever company to split its ownership into tiny pieces and sell these pieces of ownership as stocks. Not long after the issuing of these stocks, businessmen found out that more could be done with these stocks than just buying and selling. *"Confusion des Confusiones"* [2], a book from 1688 by Spanish merchant Joseph de la Vega describes the trading of "opsies" at the Amsterdam Stock Exchange. Merchants found that they could use these options to take risk cheaply. Most traded options were what we would now call vanilla call and put options, contracts giving its holder the right but not the obligation to buy/sell the underlying stock at a predetermined price at some predetermined moment in time. This period also gave birth to the first *'exotic option'*, what we would now call the straddle option (Petram, [3]). This option combines a call and a put option for the same strike price and expiration date to ensure the buyer an insurance for strong price fluctuations, allowing the writer to profit from the premium received for selling this 'insurance' and potentially benefiting if the underlying asset's price remains relatively stable. Option trade was and had been unstandardised, making it a very illiquid form of investment and sensitive to fraud. It was seen as a form of gambling and was therefore even prohibited at the London Stock Exchange in 1734, only becoming widespread again, while still prohibited, in 1832 (Michie, [4]).

One and a half century after this reintroduction of options, the first big change in the option trade occurred with the establishment of the Chicago Board of Option Exchange (CBOE) and the Options Clearing Corporation (OCC). These corporations standardised the option trade. All option contracts had the same terms and had a performance guarantee from the OCC. This was the first time options were accessible for the general public. With the trade of options becoming more widespread, new innovations in the field of option types began to arise in the 1990's. This new group of *"first generation exotic options"* includes Barrier Options, Asian options and the topic of this thesis, lookback options.

The first generation exotics are 'exotic' in the sense that their payoff is not necessarily dependent on the price of an underlying asset at some expiry time T, but on the path the asset took to get there. Barrier options focus on whether the asset price crosses some boundary, Asian options focus on the average value of the asset price and the lookback option allows the holder to pick any time between issue and expiry date and buy/sell the asset at the price the asset had at this time. This makes for very interesting value trajectories, which will be discussed in this thesis. More specifically, we will discuss the valuation of the fixed strike lookback call option (as this is the only variant we discuss in detail in this thesis, it is often just denoted as "lookback option"), which will be defined later.

This valuation of options can be done using a wide variety of methods. The most straightforward and efficient way of doing this is by finding an analytic or closed form expression of the option value. The most well-known example of such an analytic solution is the value of the European option developed by Black and Scholes [5]. In the case of very complex option definitions or underlying asset dynamics, such solutions might not be available. In such cases, one must resort to numerical pricing methods. A very well known and easily implemented numerical method is the Monte Carlo Method, a financial equivalent to rolling a thousand dice to guess its average value. The drawback of this method is that it is very inefficient. An alternative numerical method we will describe in this thesis is the COS Method, a numerical valuation method devised by Fang and Oosterlee [1]. To apply this method to the case of lookback options, we have organised the thesis as follows.
In Section 2 we will provide the reader with the mathematical background and concepts related to option pricing. We will introduce Geometric Brownian Motion (GBM), a common stochastic process used to model the dynamics of financial assets. Also, we will lay out rigorous definitions for both European and lookback options and discuss their analytical solutions under GBM. In Section 3 we will lay out the COS Method procedure, derive the parameters to be used in European and lookback option pricing and show through our implementation the efficiency and accuracy of this pricing method. Finally, in Section 4 we will extend the scope of our implementation to a broader group of asset dynamics, giving the reader a broad implementation of the COS Method regarding lookback options.

2 Theoretical Introduction

Before we get into pricing European and lookback options, a theoretical background needs to be established. In this chapter, we will first formulate Geometric Brownian Motion, a stochastic process used to describe the behaviour of stock prices. After this we will give definitions for the options that will be discussed in this thesis. We will begin with the European option, which we will use throughout this paper as a steppingstone towards the second kind of option, the fixed strike lookback option. Then, we will give analytical solutions to European and fixed strike lookback call options. Lastly, this section will define a first implementation of numerical option pricing, the Monte Carlo method.

2.1 Geometric Brownian Motion

In this subsection we will define Geometric Brownian Motion (GBM). We will do this by starting with a symmetric random walk (SRW), and build our way up to the desired stochastic process.

2.1.1 Symmetric Random Walk

In this subsection we will define the symmetric random walk. Additionally, we will discuss multiple properties of this stochastic process. The SRW is defined as follows:

Definition 2.1.1 (Symmetric Random Walk)**.** *The symmetric ransom walk is a discrete stochastic process defined by*

$$M_k = \sum_{i=1}^{k} X_j \ , \quad k = 1, 2, 3, \ldots$$

with $M_0 = 0$ and $X_j, j = 1, 2, 3, \ldots$ independent identically distributed (i.i.d.) random variables with

$$\mathbb{P}\big(X_j = 1\big) = \mathbb{P}\big(X_j = -1\big) = \frac{1}{2}$$

A first property of the symmetric random walk we will discuss is that of independent increments. First, choose $0 \leqslant k_1 < k_2 < k_3 < k_4$ and define the increments $D_1 = M_{k_2} - M_{k_1}$ and $D_2 = M_{k_4} - M_{k_3}$. Then, D_1 and D_2 are independent. This is because we know by definition that $D_1 = \sum_{i=k_1}^{k_2} X_i$, $D_2 = \sum_{i=k_3}^{k_4} X_i$ and that X_k, $k = 1, 2, 3, \ldots$ are independent and identically distributed variables. From this we can immediately derive that D_1 and D_2 are independent, as the sets of random variables they depend on are disjoint. Furthermore, we can derive expected values and variances for such increments, looking at the expected value we get

$$\mathbb{E}\left[D\right] = \mathbb{E}\left[\sum_{j=k_i+1}^{k_{i+1}} X_i\right] = \sum_{j=k_i+1}^{k_{i+1}} \mathbb{E}\left[X_j\right] = \sum_{j=k_i+1}^{k_{i+1}} 0 = 0,$$

and for the variance over an increment, given $Var(X_j) = \mathbb{E}\left[X_j^2\right] = 1$, we can derive

$$Var(D) = Var\left(\sum_{j=k_{i+1}}^{k_i} X_i\right) = \sum_{j=k_{i+1}}^{k_i} Var(X_j) = \sum_{j=k_{i+1}}^{k_i} 1 = k_{i+1} - k_i.$$

Next, we will prove that the SRW is a martingale process, we choose non-negative integers $k < l$. What follows is

$$\begin{aligned}\mathbb{E}\left[M_l|\mathcal{F}_k\right] =& \mathbb{E}\left[(M_l - M_k) + M_k|\mathcal{F}_k\right] \\ =& \mathbb{E}\left[(M_l - M_k)|\mathcal{F}_k\right] + \mathbb{E}\left[M_k|\mathcal{F}_k\right] \\ =& \mathbb{E}\left[M_l - M_k\right] + M_k = M_k,\end{aligned}$$

fulfilling the definition of a martingale process. Lastly, we will derive that the quadratic variation of a SRW is

$$[M, M]_k = \sum_{j=1}^{k} (M_j - M_{j-1})^2 = k.$$

2.1.2 Scaled Symmetric Random Walk and Brownian Motion

As a second step towards defining GBM we will first define Arithmetic Brownian Motion (ABM). We will do this by scaling a SRW.

Definition 2.1.2 (Scaled Symmetric Random Walk)**.** *A symmetric random walk scaled by factor n is defined as*

$$W^{(n)}(t) = \frac{1}{\sqrt{n}} X_{nt}, \quad W^{(n)}(0) = 0$$

provided that nt is an integer and X_j is defined as in Definition 2.1.1.

Notice that, as can be proven analogous to the previously given proof for the SRW, scaled symmetric random walks have independent increments,

$$\mathbb{E}\left[W^{(n)}(t) - W^{(n)}(s)\right] = 0 \quad \text{and} \quad Var(W^{(n)}(t) - W^{(n)}(s)) = t - s$$

and are martingale with quadratic variation $[W^{(n)}, W^{(n)}](t) = t$. We will now prove another property of the scaled symmetric random walk, namely that its evaluation at time t converges to a normal distribution.

Theorem 2.1.3 (Central Limit Theorem)**.** *For a given $t \geqslant 0$ and $n \to \infty$, a scaled SRW $W^{(n)}$ evaluated at time t approaches a normal distribution with mean 0 and variance t.*

What results from this limit of scaled random walks $W^{(n)}(t)$ as $n \to \infty$, is Brownian Motion, giving us the following rigorous definition for BM:

Definition 2.1.4 (Brownian Motion)**.** *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. For each $\omega \in \Omega$, suppose there is a continuous function $W(t)$ of $t \geqslant 0$ with $W(0) = 0$ that depends on ω. Then $W(t)$ is a BM if for all $0 = t_0 < t_1 < \ldots < t_n$ the increments $W(t_1), W(t_2) - W(t_1), \ldots, W(t_n) - W(t_{n-1})$ are independent with*

$$\mathbb{E}\left[W(t_{i+1}) - W(t_i)\right] = 0 \quad \textit{and} \quad Var(W(t_{i+1}) - W(t_i)) = t_{i+1} - t_i, \quad i = 0, 1, 2, \ldots$$

2.1.3 Geometric Brownian Motion

In this subsection we will define GBM, and discuss its most essential properties. We will use the definition for GBM as noted in Oosterlee and Grzelak [6].

Definition 2.1.5 (Geometric Brownian Motion)**.** *Let r and $\sigma > 0$ be constants, we define GBM as a process following the following stochastic differential equation (SDE)*

$$\mathrm{d}S(t) = r\, S(t)\mathrm{d}t + \sigma\, S(t)\mathrm{d}W(t)$$

Here, r represents the process' percentage drift and σ its volatility.

Say a GBM process has initial value $S(0)$, this SDE gives an analytic solution. This derivation requires Itô calculus, specifically the Itô-Doeblin Formula, see Shreve [7]:

Lemma 2.1.6 (Itô-Doeblin Formula for Itô Processes)**.** *Let $f(t, x)$ be a continuous function with partial derivatives $f_t(t, x)$, $f_x(t, x)$, and $f_{xx}(t, x)$ defined and continuous, and let $X(t)$ be a stochastic process fulfilling*

$$\mathrm{d}X(t) = \Delta(t)\mathrm{d}W(t) + \Theta(t)\mathrm{d}t,$$

where $\Delta(t)$ and $\Theta(t)$ are adapted stochastic processes, then

$$df(t, X(t)) = f_t(t, X(t))\mathrm{d}t + f_x(t, X(t))\mathrm{d}X(t) + \frac{1}{2} f_{xx}(t, X(t))\mathrm{d}X(t)\mathrm{d}X(t).$$

We can use this lemma to calculate the solution to the differential equation of a GBM with initial value $S(t_0)$. For this we will use $f(t,x) = \log x$, giving $f_t(t,x) = 0, f_x(t,x) = \frac{1}{x}$, and $f_{xx}(t,x) = \frac{-1}{x^2}$. This gives

$$\begin{aligned}
\mathrm{d}\log(S(t)) &= \frac{\mathrm{d}S(t)}{S(t)} - \frac{\mathrm{d}S(t)\mathrm{d}S(t)}{2S(t)^2} \\
&= \frac{r\,S(t)\mathrm{d}t + \sigma\,S(t)\mathrm{d}W(t)}{S(t)} - \frac{(r\,S(t)\mathrm{d}t + \sigma\,S(t)\mathrm{d}W(t))^2}{2S(t)^2} \\
&= r\,\mathrm{d}t + \sigma\,\mathrm{d}W(t) - \frac{r^2\,S(t)^2\mathrm{d}t^2 + r\sigma S(t)^2\mathrm{d}W(t)\mathrm{d}t + \sigma^2\,S(t)^2\mathrm{d}W(t)\mathrm{d}W(t)}{2S(t)^2} \\
&= r\,\mathrm{d}t - \frac{1}{2}\sigma^2\,\mathrm{d}t + \sigma\,\mathrm{d}W(t).
\end{aligned}$$

It follows that

$$\log(S(t)) = \log(\frac{S(t)}{S(0)}) = \int_0^t \left(r\ - \frac{\sigma^2}{2}\right)\mathrm{d}t + \int_0^t \sigma\,\mathrm{d}W(t) = \left(r\ - \frac{\sigma^2}{2}\right)t + \sigma\,W(t).$$

Exponentiation giving us

$$S(t) = S(0)\exp\left(\left(r\ - \frac{\sigma^2}{2}\right)t + \sigma\,W(t)\right). \tag{2.1}$$

2.2 Option Payoff

Now that we have established a mathematical background, we will now move on to what we are going to apply this to. In this section we will note the payoff of the two types of financial options discussed in this thesis. First the European option, then the lookback option.

2.2.1 European Option

The European option gives the owner of the option the right but not the obligation to buy or sell a stock for a predetermined price at a predetermined time.

Definition 2.2.1 (European Option Value at Exercise Time). *The value of a European Option at exercise time (denoted 'payoff') is*

$$V(T,S(T)) = \begin{cases} (S(T)-K)^+ & \textit{for a Call Option} \\ (K-S(T))^+ & \textit{for a Put Option} \end{cases}$$

where $[h(t,y)]^+ = \max[h(t,y),0]$, $S(t)$ *is the price of the underlying stock at time* $t \geqslant 0$, K *is the strike price of the option,* T *is the exercise time and.*

2.2.2 Lookback Option

The fixed strike lookback option gives the holder of the contract the right but not the obligation to buy the stock at the option contract's strike price (at any price reached for puts) and sell it at any price the stock reached between option issue date and exercise date (at strike price for puts). This gives lookback options the following payoff. (In this thesis we will only discuss the valuation of call options, but for a more general understanding of fixed strike lookback options, we have included put options here as well.)

Definition 2.2.2 (Fixed Strike Lookback Payoff). *The value of a fixed strike lookback option at exercise time (payoff) is*

$$V(T,S(T)) = \begin{cases} \left(\max_{0\leqslant t\leqslant T} S(t) - K\right)^+ & \textit{for a Call Option} \\ \left(K - \min_{0\leqslant t\leqslant T} S(t)\right)^+ & \textit{for a Put Option} \end{cases}$$

where $t = 0$ *is the issue date,* $S(t)$ *is the price of the underlying stock at time* $t \geqslant 0$, K *is the strike price, and* T *is the exercise time.*

2.3 Analytic Option Value

In this section, we will discuss analytic option prices. We will start by discussing the analytic solutions for European options, as described in Oosterlee and Grzelak [6]. Then, we will look at lookback options, for which the closed form value is described in Conze and Viswanathan [8].

2.3.1 European Options

For European options we will look at Oosterlee and Grzelak [6]. We will not derive the following result ourselves, as it is outside the scope of this thesis.

Theorem 2.3.1 (European Call and Put Option Value). *Given a the GBM $S(t)_{t\geqslant 0}$ as defined in Equation (2.1). European call and put options with issue date 0, exercise date T and strike price K under this stock dynamic $S(t)$ have the following value at any time $t \in [0,T]$.*

$$\begin{aligned} V_{\text{call}}(t,S(t)) &= S(t)\Phi_N(d_1) - Ke^{-r\tau}\Phi_N(d_2), \\ V_{\text{put}}(t,S(t)) &= Ke^{-r\tau}\Phi_N(-d_2) - S(t)\Phi_N(-d_1), \end{aligned}$$

where

$$\begin{aligned} d_1 &= \frac{\log\frac{S(t)}{K} + (r+\frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}}, \\ d_2 &= d_1 - \sigma\sqrt{\tau}, \end{aligned}$$

$\tau = T - t$, and Φ_N is the cumulative distribution function of a standard normal distribution.

2.3.2 Lookback Options

In this subsection we will look at lookback options, taking our result from Conze and Viswanathan [8]. Before stating the solution, note that because we want to be able to valuate option contracts anywhere in $[0,T]$, where 0 is the issue date and T is the expiration date, we have to take into account the running maximum up to the point in time from which we valuate.

Theorem 2.3.2 (Fixed Strike Lookback Call Option Value). *Given the GBM $S(t)_{t\geqslant 0}$ as defined in Equation (2.1) and a fixed strike lookback call (resp. put) option with issue date 0, exercise date T and strike price K. We denote $\widehat{S}_p^q = \max_{p\leqslant t\leqslant q} S(t)$. We can now express the value of a lookback call option as*

$$\begin{aligned} V_{\text{call}}(t,S(t)) &= S(t)\Phi(d_K) - e^{-r\tau}K\Phi\Big(d_K - \sigma\sqrt{\tau}\Big) \\ &\quad + e^{-r\tau}\frac{\sigma^2}{2r}S(t)\left[-\left(\frac{S(t)}{K}\right)^{-\frac{2r}{\sigma^2}}\Phi\left(d_K - \frac{2r}{\sigma}\sqrt{\tau}\right) + e^{r\tau}\Phi(d_K)\right] \end{aligned}$$

in the case that $\widehat{S}_0^t \leqslant K$, and

$$\begin{aligned} V_{\text{call}}(t,S(t)) &= e^{-r\tau}(\widehat{S}_0^t - K) + S(t)\Phi(d_{\widehat{S}_0^t}) - e^{-r\tau}\widehat{S}_0^t\Phi\Big(d_{\widehat{S}_0^t} - \sigma\sqrt{T}\Big) \\ &\quad + e^{-r\tau}\frac{\sigma^2}{2r}S(t)\left[-\left(\frac{S(t)}{\widehat{S}_0^t}\right)^{-\frac{2r}{\sigma^2}}\Phi\left(d_{\widehat{S}_0^t} - \frac{2r}{\sigma}\sqrt{\tau}\right) + e^{r\tau}\Phi(d_{\widehat{S}_0^t})\right] \end{aligned}$$

in the case that $\widehat{S}_0^t \geqslant K$, where

$$d_x = \left(\log\frac{S(t)}{x} + r\tau + \frac{1}{2}\sigma^2\tau\right)/\sigma\sqrt{\tau},$$

$\tau = T - t$, and Φ_N is the cumulative distribution function of a standard normal distribution.

2.4 Monte Carlo Option Pricing

As a first dipping of the toes into numerical option pricing, this subsection will briefly introduce Monte Carlo option pricing. The general Monte Carlo method was first implemented in the 1930's, but was only introduced to option pricing in 1976 in Boyle [9]. The method works by simulating a large number of asset paths according to a selected asset dynamic. A large number of properties can then be derived from these paths. In these thesis we will look at the implied probability density function (pdf) of a process at any time $t \in [0, T]$, and at the Monte Carlo risk neutral option price. In Listing 1 we have given the `def MonteCarlo(...)` function in `class Option` responsible for generating an option value estimate.

```
def MonteCarlo(self, iterations, steps, t, st, prevex=None):
    vs = np.zeros(iterations)

    for i in range(iterations):
        gbm = self.samplepath(self.T - t, steps, st)
        vs[i] = self.payoff(gbm, prevex)

    return np.exp(-self.mu*(self.T-t)) * np.average(vs)
```

Listing 1: Monte Carlo Option Pricing

Here, `self.samplepath(...)` returns an scaled SRW approximation of a GBM process, and `self.payoff(...)` returns the option type specific payoff for a given path. Note that Monte Carlo simulations can be, and are used to determine other properties of stock/option value trajectories, these different applications require only slight modification to the code above.

3 COS method Option Valuation

In this section, we will discuss the theory behind the COS method of option pricing, a way to numerically estimate the value of an option contract. This method rests on numerically estimating the probability density function of a process using only its characteristic function, which can be very useful in cases where the probability density function of a process is not available. We will first describe the general procedure of the COS method, after which we derive option-specific parameters to be used in the estimation procedure.

3.1 COS Method Procedure

To introduce the COS method as described in Fang and Oosterlee [1], we will start by introducing some notation and basic principles. Then, we will describe the COS method density approximation procedure. Lastly, we will look at how we can value options using the approximated density function.

3.1.1 Notation and Basic principles

In this thesis we will be considering a variety of processes with an initial date t_0, with $x = X(t_0)$ known. And a maturity date T where $y = X(T)$. We will therefore introduce this simplified notation for the probability density function, $f_X(y) := f_X(T, y; t_0, x)$. We use this to define a Fourier pair.

Definition 3.1.1 (Fourier Pair)**.** *The density function and the characteristic function of a process, $f_X(y)$ and $\phi_X(u)$, form a Fourier pair:*

$$\phi_X(u) = \int_{\mathbb{R}} e^{iyu} f_X(y) dy, \tag{3.1}$$

and,

$$f_X(y) = \frac{1}{2\pi} \int_{\mathbb{R}} e^{-iuy} \phi_X(u) du.$$

Another greatly important part of the COS method is the Fourier expansion, defined as follows.

Definition 3.1.2 (Fourier Expansion)**.** *The Fourier expansion of a function $g(x)$ on an interval $[-\pi, \pi]$ is as follows,*

$$g(\theta) = {\sum_{k=0}^{\infty}}' \bar{A}_k \cos(k\theta) + \sum_{k=1}^{\infty} \bar{B}_k \sin(k\theta),$$

where the accented sum Σ' indicates that the first term in the summation is multiplied by half. The coefficients in the sum are given by

$$\frac{1}{\pi}\bar{A}_k = \int_{-\pi}^{\pi} g(\theta)\cos(k\theta)d\theta, \ \frac{1}{\pi}\bar{B}_k = \int_{-\pi}^{\pi} g(\theta)\sin(k\theta)d\theta.$$

Setting $\bar{B}_k = 0$ results in the Fourier cosine expansion, allowing the exact representation of even functions around $\theta = 0$.

We will now derive how we can apply these general principles to get to a Fourier cosine expansion for any function with finite support. Firstly, note that we can extend any function $g : [0, \pi] \to \mathbb{R}$ to become an even function on $[-\pi, \pi]$ as follows

$$\bar{g}(\theta) = \begin{cases} g(\theta) & \theta \geqslant 0 \\ g(-\theta) & \theta < 0. \end{cases}$$

Then, note that this function can now be expressed as the Fourier cosine expansion

$$\bar{g}(\theta) = {\sum_{k=0}^{\infty}}' \bar{A}_k \cos(k\theta),$$

where

$$\bar{A}_k = \frac{1}{\pi}\int_{-\pi}^{\pi} \bar{g}(\theta)\cos(k\theta)d\theta = \frac{2}{\pi}\int_{0}^{\pi} g(\theta)\cos(k\theta)d\theta.$$

This gives us a way to calculate Fourier expansion of functions that can, but don't have to be even on $[0, \pi]$. We now want to generalise this method to functions on any other finite interval $[a, b] \subset \mathbb{R}$, and can do so with a change in variables

$$\theta := \frac{y-a}{b-a}\pi.$$

Meaning we define the Fourier cosine expansion more generally as follows.

Theorem 3.1.3 (Fourier Cosine Expansion for Functions with Finite Support)**.** *Given a function $g : [a, b] \to \mathbb{R}$, we can define the Fourier cosine expansion of this function as below.*

$$g(y) = {\sum_{k=0}^{\infty}}' \bar{A}_k \cos\left(k\pi\frac{y-a}{b-a}\right).$$

with the coefficients

$$\bar{A}_k = \frac{2}{b-a}\int_a^b g(y)\cos\left(k\pi\frac{y-a}{b-a}\right)dy. \tag{3.2}$$

3.1.2 Density Approximation with COS method

In this subsection we will describe how we can use the COS method to approximate the pdf of some random variable X, given its characteristic function. We will work our way towards this estimate, starting at the Fourier cosine expansion of f_X,

$$f_X(y) = {\sum_{k=0}^{\infty}}' \bar{A}_k \cos\left(k\pi\frac{y-a}{b-a}\right) \quad \text{where} \quad \bar{A}_k = \frac{2}{b-a}\int_a^b f_X(y)\cos\left(k\pi\frac{y-a}{b-a}\right)dy. \tag{3.3}$$

Note that this expression for $\bar{A}_k$ has a form similar to that of Equation (3.1). We will therefor try to express $\bar{A}_k$ into a form that resembles this equation. For this we will use the following definition of the truncated characteristic function

$$\hat{\phi}_X(u) = \int_a^b e^{iyu} f_X(y)dy \approx \int_{\mathbb{R}} e^{iyu} f_X(y)dy = \phi_X(u).$$

Note that we can transform this truncated integral as

$$\hat{\phi}_X(u)e^{i\lambda} = \int_a^b e^{i(uy+\lambda)} f_X(y)dy. \tag{3.4}$$

We introduce the substitutions $u = \frac{k\pi}{b-a}$ and $\lambda = -\frac{ka\pi}{b-a}$ into Equation (3.4) giving

$$\hat{\phi}_X\left(\frac{k\pi}{b-a}\right)\cdot\exp\left(-i\frac{ka\pi}{b-a}\right) = \int_a^b \exp\left(iy\frac{k\pi}{b-a} - i\frac{ka\pi}{b-a}\right) f_X(y)dy.$$

Taking the real part of the expressions on both sides, using Euler's formula $e^{ix} = \cos(x) + i\sin(x)$, gives us

$$\text{Re}\left\{\hat{\phi}_X\left(\frac{k\pi}{b-a}\right)\cdot\exp\left(-i\frac{ka\pi}{b-a}\right)\right\} = \int_a^b \cos\left(k\pi\frac{y-a}{b-a}\right) f_X(y)dy.$$

Allowing us to define $\bar{A}_k$ in Equation (3.3) as

$$\bar{A}_k = \frac{2}{b-a}\text{Re}\left\{\hat{\phi}_X\left(\frac{k\pi}{b-a}\right)\cdot\exp\left(-i\frac{ka\pi}{b-a}\right)\right\}.$$

Now that we have rewritten the definition of f_X, we can start approximating this function. The first step in doing this is replacing the truncated characteristic function with the true characteristic function. We define this first approximation as

$$f_{X,\text{I}}(y) = {\sum_{k=0}^{\infty}}' \bar{F}_k \cos\left(k\pi\frac{y-a}{b-a}\right) \quad \text{where} \quad \bar{F}_k = \frac{2}{b-a}\text{Re}\left\{\phi_X\left(\frac{k\pi}{b-a}\right)\cdot\exp\left(-i\frac{ka\pi}{b-a}\right)\right\}. \tag{3.5}$$

The next step in approximation is truncating the infinite summation, giving the following estimate of the density function.

Definition 3.1.4 (COS Method Density Approximate)**.** *Given a characteristic function of a random variable X denoted by ϕ_X, and appropriate bounds of integration $a, b \in \mathbb{R}$, we can approximate f_X by*

$$f_{X,\text{II}}(y) = {\sum_{k=0}^{N-1}}' \bar{F}_k \cos\left(k\pi\frac{y-a}{b-a}\right) \quad \textit{where} \quad \bar{F}_k = \frac{2}{b-a} Re\left\{\phi_X\left(\frac{k\pi}{b-a}\right) \cdot exp\left(-i\frac{ka\pi}{b-a}\right)\right\}. \tag{3.6}$$

Since cosine expansions of entire functions (i.e a function without poles, fractional powers, logarithms, other branch points, or discontinuities in a function or in any of its derivatives, except at ∞) exhibit exponential convergence according to Boyd [10], we can expect highly accurate approximations from this procedure for a low N to density functions that are entire functions on $[a, b]$.

Remark 3.1.5 (COS Method Density Approximation Error)**.** In the derivation of the COS estimate of the density function, we introduce errors in two places, firstly in the truncation of the Fourier cosine expansion, and secondly in the usage of coefficients $\bar{F}_k$ instead of $\bar{A}_k$. The resulting error can be described as

$$\varepsilon_X(y) = \left[{\sum_{k=0}^{N-1}}'(\bar{A}_k - \bar{F}_k) + \sum_{k=N}^{\infty} \bar{A}_k\right] \cos\left(k\pi\frac{y-a}{b-a}\right)$$

Where $\bar{A}_k$ is as defined in Equation (3.2) and $\bar{F}_k$ as defined in Equation (3.5).

3.1.3 Option Pricing with COS Method

Now that we have established a procedure to efficiently approximate density functions, we can move on to valuating options. We will start with the theoretical expressions describing option payoff, and from there on we will derive a way to estimate these options values numerically.

Definition 3.1.6 (Option Value at Initial Time)**.** *Given a stochastic process $X(t)$ taking values $X(t_0) = x$ and $X(T) = y$ and a path-independent payoff function $V(t, y)$, the value of the option under $X(t)$ at time t_0 is given by:*

$$V(t_0, x) = e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[V(T, y)|\mathcal{F}(t_0)\right] = e^{-r\tau}\int_{\mathbb{R}} V(T, y) f_X(T, y; t_0, x) dy,$$

where $\tau = T - t_0$, f_X is the density function of $X(t)$ given $X(t_0) = x$, and r is the interest rate.

The first and most straightforward way we are going to approximate this expression is by truncating the infinite integral to the domain $[a, b]$ just as we did in Subsection 3.1.2. The choice we make for these bounds will be discussed in Subsection 3.1.9. Also, we will revert back to the simplified notation of $F_X(y)$. The first approximation is as follows

$$V_{\text{I}}(t_0, x) = e^{-r\tau}\int_a^b V(T, y) f_X(y) dy$$

We can rewrite this by incorporating the Fourier expansion of Theorem 3.1.3 to give

$$V_{\text{I}}(t_0, x) = e^{-r\tau}\int_a^b V(T, y) {\sum_{k=0}^{\infty}}' \bar{A}_k \cos\left(k\pi\frac{y-a}{b-a}\right) dy.$$

By the boundedness, thus integrability, of the $V(T, y)$, $\bar{A}_k$ and the cosine function, and the limited integration range, we can apply Fubini's Theorem (See Tao [11]) to switch the order of summation and integration, giving

$$V_{\text{I}}(t_0, x) = \frac{b-a}{2} e^{-r\tau} {\sum_{k=0}^{\infty}}' \bar{A}_k \cdot H_k$$

where

$$H_k := \frac{2}{b-a}\int_a^b V(T, y) \cos\left(k\pi\frac{y-a}{b-a}\right) dy. \tag{3.7}$$

We can now make the next step in approximation, which, due to decay of the coefficients, is the truncation of the summation, giving us V_{II} :

$$V_{\mathrm{II}}(t_0, x) = \frac{b-a}{2} e^{-r\tau} \sum_{k=0}^{N-1}{}' \bar{A}_k \cdot H_k$$

The approximation step in this process is the incorporation of Equation (3.6) into the estimate, giving us the following definition.

Definition 3.1.7 (COS Method Pricing Formula)**.** *Given a characteristic function of a stochastic process X at time T, denoted by ϕ_X, appropriate bounds of integration $a, b \in \mathbb{R}$, and a path-independent payoff function $V(t, y)$. We can estimate the risk-neutral price of an option with the given payoff function at time t_0 with*

$$V_{\mathrm{III}}(t_0, x) := e^{-r\tau} \sum_{k=0}^{N-1}{}' Re\left\{\phi_X\left(\frac{k\pi}{b-a}\right) \cdot exp\left(-i\frac{ka\pi}{b-a}\right)\right\} \cdot H_k,$$

where $\tau = T - t_0$ is the time until expiration and $x = X(t_0)$ is the start value of the stochastic process.

Before moving on, we will first quantify the error introduced by the COS method valuation procedure in accordance with Oosterlee and Grzelak [1].

Remark 3.1.8 (COS Method Valuation Error)**.** During the COS method derivation we introduced approximation errors in three instances. The first of which is introduced by integration range truncation, this error can be expressed as

$$\varepsilon_1 := V_{\mathrm{I}}(t_0, x) - V(t_0, x) = \int_{\mathbb{R}\backslash[a,b]} V(T, y) f_X(y).$$

The second error term is a result of the series truncation error on $[a, b]$, described as

$$\varepsilon_2 := V_{\mathrm{II}}(t_0, x) - V_{\mathrm{I}}(t_0, x)(t_0, x) = \frac{b-a}{2} e^{-r\tau} \sum_{k=N}^{\infty}{}' \bar{A}_k \cdot H_k.$$

Finally, the third error term follows from approximating $\bar{A}_k$ by $\bar{F}_k$, giving the expression

$$\varepsilon_3 := V_{\mathrm{III}}(t_0, x) - V_{\mathrm{II}}(t_0, x)(t_0, x) = e^{-r\tau} \sum_{k=0}^{N-1}{}' \mathrm{Re}\left\{\int_{\mathbb{R}\backslash[a,b]} \exp\left(ik\pi\frac{y-a}{b-a}\right) f_X(y)\right\} \cdot H_k.$$

The total error introduced by the COS method is, as one would expect, the sum of these error terms.

Before we can start on defining option and process specific calculation parameters, we will first define the integration range, previously denoted $[a, b]$, for COS calculation.

Remark 3.1.9 (Choice of Integration Range)**.** Oosterlee and Grzelak [6] provides a way to determine efficient boundaries according to process-specific *'cumulants'*. Because these cumulants can be hard to determine for complicated distributions, they also provide a less efficient, but effective alternative. This alternative range is

$$[a, b] = [-L\sqrt{T}, L\sqrt{T}],$$

where $L > 0$ can be chosen. For simplicity and compatibility with both European and lookback option pricing, we choose $L = 10$.

3.2 Characteristic Functions for European Option Pricing

In this subsection we will derive characteristic functions used in European Option pricing. We will start by deriving it for ordinary Brownian Motion, then making the step towards the log price of Geometric Brownian motion.

Theorem 3.2.1 (Characteristic Function for BM at Exercise Time). *Given a Brownian Motion $W(t)_{t\geqslant 0}$, the characteristic function of this process at time T is*

$$\phi_{W(T)}(u) = e^{-\frac{1}{2}Tu^2} \tag{3.8}$$

Proof. From Theorem 2.1.3 we know that a BM at time T has a normal distribution with mean 0 and variance T, filling this in in the definition for the characteristic function of a normal distribution gives the desired function. □

With this theorem we can now derive the adjusted log price of a GBM process as follows

Theorem 3.2.2 (Characteristic Function of Adjusted GBM Log Price). *Suppose that $S(t)$ is defined as in Theorem 2.1. We will define the adjusted log price of $S(t)$ as $\bar{S}(t) = \log\big(S(t)/K\big)$. The characteristic function of $\bar{S}(t)$ at time T is*

$$\phi_{\bar{S}(T)} = e^{iu\Big(\log(S(0)/K)+[r-(\sigma^2/2)]T\Big)}e^{-\frac{1}{2}T\sigma^2u^2}. \tag{3.9}$$

Proof. First, notice that $\log S(t) = \log S(0) + (r\ -(\sigma^2/2))t + \sigma\, W(t)$, allowing us to derive the characteristic function for this process as

$$\begin{aligned}\phi_{\bar{S}(T)}(u) &= \mathbb{E}\left[e^{iu\sigma W(t)}e^{iu\Big(\log S(0)+[r-(\sigma^2/2)]t-\log K\Big)}|\mathcal{F}(T)\right]\\ &= \mathbb{E}\left[e^{iu\sigma W(T)}\right]\mathbb{E}\left[e^{iu\Big(\log(S(0)/K)+[r-(\sigma^2/2)]T\Big)}\right]\\ &= \phi_{W(T)}(\sigma u)\cdot e^{iu\Big(\log(S(0)/K)+[r-(\sigma^2/2)]T\Big)}.\end{aligned}$$

Filling this in for Equation (3.8) gives the desired result. □

3.3 Characteristic Function for Lookback Option Pricing

In this subsection we will derive the characteristic functions to be used in the COS method for pricing lookback options under BM and GBM. Note that we cannot simply fill re-use the characteristic functions from the last paragraph, replacing $V(t, S(t))$ with the functions given in Definition 2.2.2, as lookback option payoff is path-dependent, and $f_X(y) = f_X(T, y, t_0, x)$ only carries information on stock value y at $t = T$. To resolve this, we will derive characteristic functions of the extrema of the relevant stochastic processes.

3.3.1 Mirror Principle for Brownian Motion

A first step in this derivation for (Geometric) Brownian Motion is the reflection principle. Before we can start with this proof we need the following lemma, the proof for which can be found in Weir [12].

Lemma 3.3.1 (Bounded Convergence Theorem). *If (f_n) is a sequence of uniformly bounded complex-valued measurable functions which converges pointwise on a bounded measure space (S, Σ, μ) to a function f, then the limit f is an integrable function and*

$$\lim_{n\to\infty}\int_S f_n d\mu = \int_S f d\mu.$$

We can now give the proof of the reflection principle for BM, in which we follow the steps of Dilworth and Wright [13].

Theorem 3.3.2 (Strong Reflection Principle for Brownian Motion). *Given a Brownian Motion $(W(t))_{t\geqslant 0}$, and let τ be a stopping time with respect to $(\mathcal{F}_t)_{t\geqslant 0}$. Then*

$$W_\tau(t) := \begin{cases} W(t) & 0\leqslant t\leqslant\tau\\ 2W(\tau)-W(t) & t>\tau\end{cases}$$

Is a Brownian Motion.

Proof. First, note that $W_\tau(t)$ has continuous paths. Now, using the fact that the distribution of an $\mathbb{R}^n$- valued random vector is uniquely determined by its characteristic function $\varphi_X(u) = \mathbb{E}\left[\exp\left(iuX\right)\right]$, it is enough to show for each $n \geqslant 1$ and $0 = t_0 < t_1 < \cdots < t_n < t_{n+1} = \infty$ and $u_j \in \mathbb{R}$ $(1 \leqslant j \leqslant n)$, that $\mathbb{E}\left[e^{iX}\right] = \mathbb{E}\left[e^{iX_\tau}\right]$, where

$$X := \sum_{j=1}^{n} u_j W(t_j) \quad \text{and} \quad X_\tau := \sum_{j=1}^{n} u_j W_\tau(t_j).$$

Suppose now that τ takes one of many values $0 < \tau_1 < \cdots < \tau_m < \infty$ (We will come back to general values of τ later). We can now, for each $1 \leqslant r \leqslant m$, choose a value k_r so that $t_{k_r} \leqslant \tau_r \leqslant t_{k_r+1}$, and define

$$Y_r := \sum_{r=1}^{k_r} u_j W(t_j) + \left(\sum_{r=k_r+1}^{n} u_j\right) W(\tau_r) \quad \text{and} \quad Z_r := \sum_{r=k_r+1}^{n} u_j (W(t_j) - W(\tau_r)),$$

Note that Y_r is $\mathcal{F}_{\tau_r}$- measurable, Z_r is independent of $\mathcal{F}_{\tau_r}$, and

$$X = \sum_{r=1}^{m} (Y_r + Z_r)\mathbb{I}_{\{\tau=\tau_r\}} \quad \text{and} \quad X_\tau = \sum_{r=1}^{m} (Y_r - Z_r)\mathbb{I}_{\{\tau=\tau_r\}}.$$

With this information and with the fact that Z_r is symmetric, we can derive

$$\begin{aligned}
\mathbb{E}\left[e^{iX_\tau}\right] &= \sum_{r=1}^{m} \mathbb{E}\left[e^{i(Y_r-Z_r)}\mathbb{I}_{\{\tau=\tau_r\}}\right] \\
&= \sum_{r=1}^{m} \mathbb{E}\left[e^{iY_r}\mathbb{I}_{\{\tau=\tau_r\}}\right]\mathbb{E}\left[e^{-iZ_r}\right] \\
&= \sum_{r=1}^{m} \mathbb{E}\left[e^{iY_r}\mathbb{I}_{\{\tau=\tau_r\}}\right]\mathbb{E}\left[e^{iZ_r}\right] \\
&= \sum_{r=1}^{m} \mathbb{E}\left[e^{i(Y_r+Z_r)}\mathbb{I}_{\{\tau=\tau_r\}}\right] = \mathbb{E}\left[e^{iX}\right].
\end{aligned}$$

We now only need to extend our result to accept a general stopping time τ. Which we will do by approximating τ. Define

$$\tau_j = \begin{cases} 2^j & \text{if} \quad \tau > 2^j \\ k \cdot 2^j & \text{if} \quad (k-1)2^{-j} < \tau < k2^{-j} \leqslant 2^j \quad \text{for a} \quad k \in \mathbb{N}. \end{cases}$$

Now we have $\tau_j \to \tau$ almost surely, which implies $X_{\tau_j} \to X_\tau$ because of continuity of paths of $W(t)$. We can now conclude our proof with Lemma 3.3.1, which now implies

$$\mathbb{E}\left[e^{iX_\tau}\right] = \lim_{j\to\infty} \mathbb{E}\left[e^{iX_{\tau_j}}\right] = \mathbb{E}\left[e^{iX}\right].$$

completing our proof. □

3.3.2 Joined Density of Drifted BM and its Running Maximum

Now that we have derived the mirror principle for BM we can continue with the next step in describing the maximum of a (G)BM, which is the derivation of the joined density function of a BM and its maximum, from which we will then derive the same function for drifted BM. In the following proof we take inspiration from the proof in Privault [14].

Theorem 3.3.3 (Joined pdf of BM and its Running Maximum)**.** *We suppose that $W(t)_{t>0}$ is a BM, the joint probability distribution of $W(T)$ and $\widehat{W}_0^T = \max_{0\leqslant t\leqslant T} W(t)$ can be described as*

$$f(x,y) = \sqrt{\frac{2}{\pi}}\frac{(2y-x)}{T^{3/2}}e^{-(2y-x)^2/2T}, \quad x < y.$$

Proof. We denote the first hitting time of this Brownian Motion to value y as $\tau_y = \inf\{t \mid W(t) = y\}$. Now, for any $x \leqslant y$, we can use Theorem 3.3.2, the fact that $2y - x \geqslant y$ and Theorem 2.1.3 to derive

$$\begin{aligned}\mathbb{P}\big(\widehat{W}_0^T > y, W(T) \leqslant x\big) &= \mathbb{P}\big(\widehat{W}_0^T > y, W(T) \geqslant 2y - x\big)\\ &= \mathbb{P}\big(W(T) \geqslant 2y - x\big)\\ &= \frac{1}{\sqrt{2\pi T}} \int_{2y-x}^{\infty} e^{-u^2/(2T)} du.\end{aligned}$$

By definition of $f(x, y)$ we can now say

$$\begin{aligned}f(x,y) &= \frac{\partial^2}{\partial x \partial y}\mathbb{P}\big(\widehat{W}_0^T \leqslant y, W(T) \leqslant x\big)\\ &= \frac{\partial^2}{\partial x \partial y}\big(\mathbb{P}\big(W(T) \leqslant x\big) - \mathbb{P}\big(\widehat{W}_0^T > y, W(T) \leqslant x\big)\big)\\ &= -\frac{\partial^2}{\partial x \partial y}\mathbb{P}\big(W(T) \geqslant 2y - x\big)\\ &= -\frac{\partial^2}{\partial x \partial y}\frac{1}{\sqrt{2\pi T}}\int_{2y-x}^{\infty} e^{-u^2/(2T)} du\, \mathbb{I}_{\{x<y\}}\\ &= -\frac{1}{\sqrt{2\pi T}}\frac{\partial^2}{\partial x \partial y}\left[\int_0^{\infty} e^{-u^2/(2T)} du - \int_0^{2y-x} e^{-u^2/(2T)} du\right] \mathbb{I}_{\{x<y\}}\\ &= \frac{1}{\sqrt{2\pi T}}\frac{\partial^2}{\partial x \partial y}\int_0^{2y-x} e^{-u^2/(2T)} du\, \mathbb{I}_{\{x<y\}}\\ &= \sqrt{\frac{2}{\pi T}}\frac{\partial}{\partial x} e^{-(2y-x)^2/(2T)}\mathbb{I}_{\{x<y\}}\\ &= \sqrt{\frac{2}{\pi}}\frac{(2y-x)}{T^{3/2}} e^{-(2y-x)^2/(2T)}\mathbb{I}_{\{x<y\}}\end{aligned}$$

Concluding our proof. □

Now, we want to make the transition to drifted BM, for this we first need Girsanov's theorem, which can be found in Shreve [7].

Theorem 3.3.4 (Girsanov's Theorem in One Dimension)**.** *Let $W(t)_{0 \leqslant t \leqslant T}$ be a BM on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Let $\Theta(t)_{0 \leqslant t \leqslant T}$, be an adapted process. Define*

$$\begin{aligned}Z(s) &= exp\left(-\int_0^t \Theta(u) dW(u) - \frac{1}{2}\int_0^t \Theta^2(u) du\right),\\ B(T) &= W(T) + \int_0^T \Theta(u) du,\end{aligned}$$

and assume that $\mathbb{E}\left[\int_0^T \Theta^2(u) Z^2(u) du\right] < \infty$. Set $Z = Z(T)$. Then $\mathbb{E}[Z] = 1$ and under probability measure $\widetilde{\mathbb{P}}$ given by

$$\widetilde{\mathbb{P}}(A) = \int_A Z(\omega) d\mathbb{P}(\omega) \quad \textit{for all} \quad A \in \mathcal{F}$$

the process $B(t)_{0 \leqslant t \leqslant T}$ is a Brownian Motion.

Say we define our drifted BM to be $W(t) + \alpha t$, where α is a constant drift factor. We can now use Girsanov's Theorem to derive the pdf for the joined density of a drifted Brownian Motion and its maximum.

Theorem 3.3.5 (Joined pdf of Drifted BM and its Running Maximum)**.** *We suppose that $W(t)_{t>0}$ is a BM, α is a constant and define the drifted Brownian Motion $B(t) = W(t) + \alpha t$. The joint probability distribution of $B(T)$ and $\widehat{B}_0^T = \max_{0 \leqslant t \leqslant T} B(t)$ can be described as*

$$f(x,y) = e^{-\alpha x - (1/2)\alpha^2 T}\sqrt{\frac{2}{\pi}}\frac{(2y-x)}{T^{3/2}} e^{-(2y-x)^2/2T}, \quad x < y.$$

Proof. By definition of a pdf we are looking to derive

$$f(x,y) = \frac{\partial^2}{\partial x \partial y}\mathbb{P}\big(\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\big). \tag{3.10}$$

Following Girsanov's theorem we will define

$$Z(t) = e^{-\alpha W(T) - \frac{1}{2}\alpha^2 T},$$

with which we can rewrite the right hand side of Equation (3.10) as

$$\begin{aligned}
\mathbb{P}\big(\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\big) &= \mathbb{E}\left[\mathbb{I}_{\{\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\}}\right] \\
&= \int_\Omega \mathbb{I}_{\{\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\}} d\mathbb{P} \\
&= \int_\Omega Z\, \mathbb{I}_{\{\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\}} d\widetilde{\mathbb{P}} \\
&= \widetilde{\mathbb{E}}\left[e^{-\alpha W(T) - \frac{1}{2}\alpha^2 T}\mathbb{I}_{\{\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\}}\right] \\
&= \widetilde{\mathbb{E}}\left[e^{-\alpha \widetilde{W}(T) - \frac{1}{2}\alpha^2 T}\mathbb{I}_{\{\widetilde{M}_0^T \leqslant y, \widetilde{W}(T) \leqslant x\}}\right] \\
&= \sqrt{\frac{2}{\pi}} \int_0^y \int_{-\infty}^x \mathbb{I}_{\{v \geqslant u\}}\, e^{-\alpha u - \frac{1}{2}\alpha^2 T}\, \frac{(2v-u)}{T^{3/2}} e^{-(2v-u)^2/2T} du dv.
\end{aligned}$$

Leaving the desired equation after differentiation. □

3.3.3 Density Function of Drifted BM Maximum

With the acquired joined pdf we can now derive the marginal pdf for $\widehat{B}_0^T$. We will from now on denote the pdf, the cdf and the characteristic function of the standard normal distribution as f_N, Φ_N and ϕ_N respectively. The derivation of the pdf for the running maximum of a BM goes as follows. This proof takes inspiration from Wang, Zou and Yin [15].

Theorem 3.3.6 (PDF of Drifted BM Maximum)**.** *We suppose that $W(t)_{t>0}$ is a BM, α is a constant and define the drifted Brownian Motion $B(t) = W(t) + \alpha t$. The probability distribution of $\widehat{B}_0^T = \max_{0 \leqslant t \leqslant T} B(t)$ can be described as*

$$f_{\widehat{B}_0^T}(y) = \sqrt{\frac{2}{T\pi}} e^{2y\alpha}\left(e^{-(y+T\alpha)^2/2T} - \alpha\sqrt{2\pi T}\, \Phi_N\left(\frac{-y - T\alpha}{\sqrt{T}}\right)\right)$$

Proof. From Theorem 3.3.5 and the definition of marginal pdf we can say

$$\begin{aligned}
f_{\widetilde{M}_0^T}(y) &= \sqrt{\frac{2}{\pi}} \frac{1}{T^{3/2}} \int_{-\infty}^y e^{-\alpha x - (1/2)\alpha^2 T}(2y - x) e^{-(2y-x)^2/2T} dx \\
&= \sqrt{\frac{2}{\pi}} \frac{1}{T^{3/2}} \left[2y e^{2y\alpha} \int_{-\infty}^y e^{-[x-(2y+T\alpha)]^2/2T} dx - e^{2y\alpha} \int_{-\infty}^y x e^{-[x-(2y+T\alpha)]^2/2T} dx\right].
\end{aligned} \tag{3.11}$$

Since

$$\begin{aligned}
\int_{-\infty}^y x e^{-[x-(2y+T\alpha)]^2/2T} dx = &\int_{-\infty}^y [x - (2y + T\alpha)] e^{-[x-(2y+T\alpha)]^2/2T} dx \\
&+ \int_{-\infty}^y (2y + T\alpha) e^{-[x-(2y+T\alpha)]^2/2T} dx,
\end{aligned}$$

we can substitute this into Equation (3.11) to give

$$\begin{aligned}
f_{\hat{B}_0^T}(y) = \sqrt{\frac{2}{\pi}}\frac{1}{T^{3/2}} \quad & \bigg[2ye^{2y\alpha}\int_{-\infty}^{y} e^{-[x-(2y+T\alpha)]^2/2T}dx \\
& - e^{2y\alpha}\int_{-\infty}^{y}[x-(2y+T\alpha)]e^{-[x-(2y+T\alpha)]^2/2T}dx \\
& - e^{2y\alpha}\int_{-\infty}^{y}(2y+T\alpha)e^{-[x-(2y+T\alpha)]^2/2T}dx\bigg]. \\
= -\sqrt{\frac{2}{\pi}}\frac{1}{T^{3/2}}e^{2y\alpha}\bigg[& \int_{-\infty}^{y}[x-(2y+T\alpha)]e^{-[x-(2y+T\alpha)]^2/2T}dx \\
& + T\alpha\int_{-\infty}^{y} e^{-[x-(2y+T\alpha)]^2/2T}dx\bigg].
\end{aligned} \tag{3.12}$$

Next, we will simplify the integrals in this expression to get to the desired result. We can rewrite the second integral using the substitution $u = [x-(2y+T\alpha)]/\sqrt{T}$, also using the fact that $\Phi_N(x) = \int_{-\infty}^{x} e^{-t^2/2}dt$ as follows

$$\begin{aligned}
&\int_{-\infty}^{y} e^{-[x-(2y+T\alpha)]^2/2T}dx \\
&\quad = \int_{x=-\infty}^{x=y} e^{-u^2/2}\sqrt{T}du \\
&\quad = \sqrt{T}\int_{-\infty}^{-(y-T\alpha)/\sqrt{T}} e^{-u^2/2}du \\
&\quad = \sqrt{2\pi T}\,\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right).
\end{aligned}$$

We can rewrite the first integral using a similar substitution $u = [x-(2y+T\alpha)]/\sqrt{2T}$, and noting that $\int xe^{-x^2/2}dx = -\frac{1}{2}e^{-x^2} + C$, as follows

$$\begin{aligned}
&\int_{-\infty}^{y}[x-(2y+T\alpha)]e^{-[x-(2y+T\alpha)]^2/2T}dx \\
&\quad = 2T\int_{-\infty}^{-(y+T\alpha)/\sqrt{2T}} ue^{-u^2}du \\
&\quad = 2T\left[-\frac{1}{2}e^{-u^2}\right]_{-\infty}^{-(y+T\alpha)/\sqrt{2T}} \\
&\quad = -Te^{-(y+T\alpha)^2/2T}.
\end{aligned}$$

Lastly, we substitute these expressions into Equation (3.12) to get the desired result

$$\begin{aligned}
f_{\hat{B}_0^T}(y) &= -\sqrt{\frac{2}{\pi}}\frac{1}{T^{3/2}}e^{2y\alpha}\left[-Te^{-(y+T\alpha)^2/2T} + T\alpha\sqrt{2\pi T}\,\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)\right] \\
&= \sqrt{\frac{2}{T\pi}}e^{2y\alpha}\left[e^{-(y+T\alpha)^2/2T} - \alpha\sqrt{2\pi T}\,\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)\right].
\end{aligned}$$

Concluding our proof. □

3.3.4 Characteristic Functions for Maxima of Drifted BM and GBM Log Price

Now that we have the pdf for the maximum of a drifted BM we can finally derive its characteristic function. Again, this proof takes inspiration from Wang, Zou and Yin [15].

Theorem 3.3.7 (Characteristic Function of Drifted BM Maximum)**.** *We suppose that $W(t)_{t>0}$ is a BM, α is a constant and define the drifted Brownian Motion $B(t) = W(t) + \alpha t$. The characteristic function of $\widehat{B}_0^T = \max_{0\leqslant t\leqslant T} B(t)$ can be described as*

$$\phi_{\widehat{B}_0^T}(u) = 2\left(1 - \frac{\alpha}{iu + 2\alpha}\right)e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u - 2\alpha i)\big).$$

Proof. From the definition of the characteristic function and Theorem 3.3.6 we can say that

$$\begin{aligned}\phi_{\widehat{B}_0^T}(u) &= \sqrt{\frac{2}{T\pi}}\int_{-\infty}^{\infty} e^{iuy}e^{2y\alpha}\left[e^{-(y+T\alpha)^2/2T} - \alpha\sqrt{2\pi T}\,\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)\right]dy\\ &= \sqrt{\frac{2}{T\pi}}\left[\int_{-\infty}^{\infty} e^{i(u-2\alpha i)y}e^{-(y+T\alpha)^2/2T}dy - \alpha\sqrt{2\pi T}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)y}\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)dy\right].\end{aligned} \tag{3.13}$$

Taking on this expression part by part, the first integral in the brackets in Equation (3.13) can be simplified using the substitution $p = (y + T\alpha)/\sqrt{T}$, keeping in mind that $\phi_N(u) = \int_{-\infty}^{\infty} e^{iuy}(1/\sqrt{2\pi})e^{-y^2/2}dy$, as follows

$$\begin{aligned}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)y}e^{-(y+T\alpha)^2/2T}dy &= \int_{-\infty}^{\infty} e^{i(u-2\alpha i)(\sqrt{T}p-T\alpha)}e^{-p^2/2}\sqrt{T}dp\\ &= \sqrt{2\pi T}e^{-i(u-2\alpha i)T\alpha}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)\sqrt{T}p}\frac{1}{\sqrt{2\pi}}e^{-p^2/2}dp\\ &= \sqrt{2\pi T}e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u-2\alpha i)\big).\end{aligned} \tag{3.14}$$

Then, simplifying the second integral in brackets in Equation (3.13) through integration by parts, followed by using the substitution $p = (y + T\alpha)/\sqrt{T}$ gives

$$\begin{aligned}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)y}\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)dy &= \frac{1}{i(u-2\alpha i)}e^{i(u-2\alpha i)y}\Phi_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)\Bigg|_{-\infty}^{\infty}\\ &\quad - \int_{-\infty}^{\infty}\frac{1}{i(u-2\alpha i)}e^{i(u-2\alpha i)y}f_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)\left(-\frac{1}{\sqrt{T}}\right)dy\\ &= \frac{1}{i(u-2\alpha i)}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)y}\frac{1}{\sqrt{T}}f_N\left(\frac{-y-T\alpha}{\sqrt{T}}\right)dy\\ &= \frac{1}{i(u-2\alpha i)}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)(\sqrt{T}p-T\alpha)}\frac{\sqrt{T}}{\sqrt{T}}f_N(p)dp\\ &= \frac{1}{i(u-2\alpha i)}e^{-i(u-2\alpha i)T\alpha}\int_{-\infty}^{\infty} e^{i(u-2\alpha i)\sqrt{T}p}f_N(p)dp\\ &= \frac{1}{i(u-2\alpha i)}e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u-2\alpha i)\big).\end{aligned} \tag{3.15}$$

Lastly, we substitute the expressions from Equations (3.14) and (3.15) into (3.13) to get the desired function

$$\begin{aligned}\phi_{\widehat{B}_0^T}(u) &= \sqrt{\frac{2}{T\pi}}\bigg[\sqrt{2\pi T}e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u-2\alpha i)\big)\\ &\qquad - \alpha\sqrt{2\pi T}\frac{1}{i(u-2\alpha i)}e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u-2\alpha i)\big)\bigg]\\ &= 2\left(1 - \frac{\alpha}{iu+2\alpha}\right)e^{-i(u-2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(u-2\alpha i)\big).\end{aligned}$$

□

In the following theorem we will take the step from describing the maximum of drifted Brownian Motion to describing the maximum of a Geometric Brownian Motion. To make this step we will take advantage of the property that taking the natural logarithm of a GBM leaves us with a drifted Brownian motion, allowing us to use Theorem 3.3.7.

Theorem 3.3.8 (Characteristic Function of GBM Log Price Maximum)**.** *We suppose that $W(t)_{t>0}$ is a BM, r and σ are constants. We define the Geometric Brownian Motion $S(t) = S(0)\,exp\big(\big(r\ -(\sigma^2/2)\big)\,t + \sigma\,W(t)\big)$ as in Equation* (2.1)*. The characteristic function of the maximum log price $\widehat{Z}_0^T = \max_{0\leqslant t\leqslant T}\big(log\,S(t)\big)$ for this GBM can be described as*

$$\phi_{\widehat{Z}_0^T}(u) = 2e^{iu\ \log S(0)}\left(1 - \frac{\alpha}{i\sigma u + 2\alpha}\right)e^{-i(\sigma u - 2\alpha i)T\alpha}\phi_N\big(\sqrt{T}(\sigma u - 2\alpha i)\big) \tag{3.16}$$

where $\alpha = \frac{1}{\sigma}(r\ -\sigma^2/2)$.

Proof. Note that by writing out the definition of $\widehat{Z}_0^T$ we get

$$\widehat{Z}_0^T = \log S(0) + \sigma \max_{0\leqslant t\leqslant T}\big(\alpha t + W(t)\big)$$

We determine $\phi_{\widehat{B}_0^T}(u)$ for $\alpha t + W(t)$ by filling in the equation in Theorem 3.3.7. The desired equation can now be constructed as

$$\phi_{\widehat{Z}_0^T}(u) = \mathbb{E}\left[e^{iu\log S(0) + iu\sigma\max_{0\leqslant t\leqslant T}(\alpha t + W(t))}\right] = e^{iu\ \log S(0)}\phi_{\widehat{B}_0^T}(\sigma u)\ \text{ and}$$

concluding our proof. □

Now that we have found our way towards GBM we will transform this characteristic functions into a form suited for our implementation. A first change we will make to our previous expression is to adjust it to the strike price. Secondly, we will adjust the time frame for which we determine the characteristic function. Instead of calculating the characteristic function for the maximum at time T starting at 0, we will now calculate it for the maximum at time T starting at time t. This last alteration is necessary because we aim to valuate lookback options not only at issue date, but at any point in time between issue- and exercise date.

Theorem 3.3.9 (Characteristic Function of GBM Adjusted Log Price Maximum)**.** *Suppose that $S(s)_{s\geqslant 0}$ is defined as in Equation* (2.1) *and that we are considering a fixed strike lookback option. We will now define the adjusted log price of the maximum of $S(s)_{p\geqslant s\geqslant q}$ as $\widehat{\zeta}_p^q = \log\big[\big(\max_{p\leqslant s\leqslant q} S(s)\big)/K\big]$. What we find for this expression's characteristic function is*

$$\phi_{\widehat{\zeta}_t^T}(u) = 2e^{iu\ \log(S(t)/K)}\left(1 - \frac{\alpha}{i\sigma u + 2\alpha}\right)e^{-i(\sigma u - 2\alpha i)\tau\alpha}\phi_N\big(\sqrt{\tau}(\sigma u - 2\alpha i)\big),$$

where $\alpha = \frac{1}{\sigma}(r\ -\sigma^2/2)$ and $\tau = T - t$.

Proof. Note that

$$\widehat{\zeta}_0^T = \widehat{Z}_0^T - \log K \qquad .$$

Incorporating the definition of the characteristic function, we get

$$\phi_{\widehat{\zeta}_0^T}(u) = \mathbb{E}\left[e^{iuZ_0^T}e^{-iu\log K}\right] = e^{-iu\log K}\phi_{\widehat{Z}_0^T}(u).$$

Filling in our definition for $\phi_{\widehat{Z}_0^T}(u)$ from Equation (3.16), followed by substituting in t for 0 as a lower bound, replacing T with τ, gives the desired expressions. □

3.4 Payoff Coefficients

In this subsection we will describe the empty spots we still have left in our COS equation for option pricing, namely, the payoff functions and the respective payoff coefficients. First for our implementation of European options, then for fixed strike lookback options. Before we start, we introduce some notation we take from Oosterlee and Grzelak [6].

Remark 3.4.1 (Notation for Frequently Recurring Integrals)**.** The cosine series coefficients of $g(y) = e^y$, of $g(y) = 1$, and of $g(y) = y$ on an integration range $[c, d] \subset [a, b]$

$$\chi_k(c,d) = \int_c^d e^y \cos\left(k\pi \frac{y-a}{b-a}\right) dy, \tag{3.17}$$

$$\psi_k(c,d) = \int_c^d \cos\left(k\pi \frac{y-a}{b-a}\right) dy, \tag{3.18}$$

are known analytically. We know that

$$\begin{aligned}\chi_k(c,d) = {} & \frac{1}{1+\left(\frac{k\pi}{b-a}\right)^2}\Bigg[\cos\left(k\pi\frac{d-a}{b-a}\right)e^d - \cos\left(k\pi\frac{c-a}{b-a}\right)e^c \\ & + \frac{k\pi}{b-a}\sin\left(k\pi\frac{d-a}{b-a}\right)e^d - \frac{k\pi}{b-a}\sin\left(k\pi\frac{c-a}{b-a}\right)e^c\Bigg], \\ \psi_k(c,d) = {} & \begin{cases}\left[\sin\left(k\pi\frac{d-a}{b-a}\right) - \sin\left(k\pi\frac{c-a}{b-a}\right)\right]\frac{b-a}{k\pi} & k \neq 0, \\ (d-c) & k = 0.\end{cases}\end{aligned}$$

Proofs for these results will be omitted.

3.4.1 European Options

With this notation established, we can get to computing payoff coefficients for European option valuation under GBM. We will use the characteristic function from Equation (3.9) and the payoff function used in Fang and Oosterlee [1]

$$V(T,y) = \begin{cases}\left[K(e^y-1)\right]^+ & \text{for a Call Option} \\ \left[K(1-e^y)\right]^+ & \text{for a Put Option.}\end{cases}$$

Computing the payoff coefficients via Equation (3.7) now gives

$$\begin{aligned}H_k^{\text{call}} &= \frac{2}{b-a}\int_0^b K(e^y-1)\cos\left(k\pi\frac{y-a}{b-a}\right)dy = \frac{2K}{b-a}\big(\chi_k(0,b) - \psi_k(0,b)\big) \\ H_k^{\text{put}} &= \frac{2}{b-a}\int_a^0 K(1-e^y)\cos\left(k\pi\frac{y-a}{b-a}\right)dy = \frac{2K}{b-a}\big(\psi_k(a,0) - \chi_k(a,0)\big)\end{aligned}$$

Under the condition that $a < 0 < b$. This completes the COS method preparation regarding European Options.

3.4.2 Lookback Options

Next, we will compute the coefficients for lookback call options. An important thing to note here is that when valuating lookback options at some time t, there is a running maximum up to t that could determine the payoff. So, define $S(s)_{s\geqslant 0}$ as in Equation (2.1), denote $\widehat{S}_p^q = \max_{p\leqslant s\leqslant q} S(s)$. Suppose that $t \in [0, T]$. Note that when the maximum between 0 and t holds until T, we have to compute the payoff with this running maximum. Only when this previous maximum is surpassed between t and T, do we compute payoff using the maximum value at T. Concretely, this gives payoff function

$$V(T,y) = \begin{cases}[\widehat{S}_0^t - K]^+\mathbb{I}_{\{y<\log(\widehat{S}_0^t/K)\}} + \left[K(e^y-1)\right]^+\mathbb{I}_{\{y>\log(\widehat{S}_0^t/K)\}} & \text{for a Call Option} \\ [K - \check{S}_0^t]^+\mathbb{I}_{\{y>\log(\check{S}_0^t/K)\}} + \left[K(1-e^y)\right]^+\mathbb{I}_{\{y>\log(\check{S}_0^t/K)\}} & \text{for a Put Option}\end{cases} \tag{3.19}$$

We can now compute the payoff coefficients via Equation (3.7), giving

$$\begin{aligned}
H_k^{\text{call}} &= \frac{2}{b-a}\int_a^{\log(\widehat{S}_0^t/K)} \left(\widehat{S}_0^t - K\right)\cos\left(k\pi\frac{y-a}{b-a}\right) dy\, \mathbb{I}_{\{\widehat{S}_0^t > K\}} \\
&\quad + \frac{2}{b-a}\int_{\log(\widehat{S}_0^t/K)}^{b} K(e^y - 1)\cos\left(k\pi\frac{y-a}{b-a}\right) dy\, \mathbb{I}_{\{\widehat{S}_0^t > K\}} \\
&\quad + \frac{2}{b-a}\int_0^{b} K(e^y - 1)\cos\left(k\pi\frac{y-a}{b-a}\right) dy\, \mathbb{I}_{\{\widehat{S}_0^t < K\}} \\
&= \frac{2\left(\widehat{S}_0^t - K\right)}{b-a}\psi_k\left(a, \log(\widehat{S}_0^t/K)\right)\mathbb{I}_{\{\widehat{S}_0^t > K\}} \\
&\quad + \frac{2K}{b-a}\left[\chi_k\left(\log(\widehat{S}_0^t/K), b\right) - \psi_k\left(\log(\widehat{S}_0^t/K), b\right)\right]\mathbb{I}_{\{\widehat{S}_0^t > K\}} \\
&\quad + \frac{2K}{b-a}\left[\chi_k(0,b) - \psi_k(0,b)\right]\mathbb{I}_{\{\widehat{S}_0^t < K\}},
\end{aligned}$$

under the condition that $a < 0 < b$. This completes the COS method preparation regarding Lookback Options.

3.5 Python Implementation

The the last step in the lookback valuation procedure is the implementation. For this we used Python, the complete script can be found in Appendix A, but can also be accessed through this GitHub page github.com/JordPBvE/bachelor-project. In the following paragraphs we will provide a summary of the program and its workings.

3.5.1 The classes.py file

In the `classes.py` file all of the parameters of the COS method procedure are defined. At the basis of the COS method implementation stands the `class GBM`. A `GBM` object contains the GBM parameters μ and σ, and contains functions allowing us to calculate the characteristic function for the adjusted log process at time T as in Equation (3.8), namely `phi_adjlog`, and the adjusted log maximum from Equation (3.16) at time T with `phi_adjlogmax`.

The next class in this file is the 'template class' `class Option`, having two children `class European` and `class Lookback`. These objects contain all option parameters `stockmechanic, exercisetime, strikeprice` and `optionvariant`, and two parameters used in most COS method calculations `upperintegration` and `lowerintegration`. When it comes to functionality this class defines the following functions. `def chi` and `def psi`, implementations of Equation (3.17) resp. (3.18) as helping functions to define the H_k-coefficients. Furthermore, to implement Monte Carlo Simulations we have `def samplepath` to generate a sample path for the given `GBM` stock mechanic and `def MonteCarlo` to generate N of these sample paths, deriving a Monte Carlo option value estimate from this.

Regarding the children `class Lookback` and `class European`, which don't have child-specific constructor functions, we have defined functions returning the COS method parameters, namely `def H` used to calculate the H_k-coefficients, `def phi_adjlog_from` returning the value of the required characteristic function given a starting point $t \in [0, T]$ and an input value u. Furthermore we have defined `def payoff` to give option payoff for a given sample path in Monte Carlo simulation, and a function `def analytic` to be used for comparing our results to.

3.5.2 The main.py file

The purpose of the `main.py` file is the COS method implementation given all needed parameters. The two most important functions defined in this file are `def EU` and `def LB`, these functions take in all needed parameters for a COS Estimation of an European resp. Lookback option, and output the analytical option value, followed by a series of procedurally more accurate COS estimates. Both these functions are based on the `def COS_Estimate` function, taking in an option object and values for t, x and N and outputting a single COS estimate. Furthermore, there is a multiple of functions that can be used for error evaluation, COS density estimation, and Monte Carlo estimate validation.

3.6 Results

In this section we will discuss the numerical results we achieved for COS Method option valuation and in the process leading up to these results. Firstly, we will validate our analytic option values. Following that, we will give the computational results we acquired from COS method density approximation and for COS method lookback option valuation.

3.6.1 Analytical Solution validation

To validate the analytical solutions listed in Section 2.3, we use Monte Carlo simulations with sufficiently many iterations (N_{MC}), in Figure 1 you will find the results for valuating an EU option with the following parameters.

$$\mu = 0.1 \quad \sigma = 0.4 \quad K = 100 \quad t = 0.2 \quad S(t) = 100$$

	$T = 1/4$			$T = 1/2$			$T = 1$			$T = 2$		
N_{MC}	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**
10^2	3.81	4.02	0.21	10.15	12.32	2.17	17.83	17.16	0.67	28.70	30.33	1.63
10^3	3.81	3.86	0.05	10.15	10.42	0.27	17.83	16.99	0.84	28.70	29.32	0.62
10^4	3.81	3.68	0.13	10.15	9.84	0.31	17.83	17.86	0.03	28.70	29.29	0.59
10^5	3.81	3.81	0.0	10.15	10.16	0.01	17.83	17.62	0.21	28.70	28.70	0.0

Figure 1: EU Option Analytic Value Validation

We can do a similar simulation for lookback options, in Figure 2 we have listed the results for a lookback call option with the following parameters.

$$\mu = 0.1 \quad \sigma = 0.4 \quad K = 100 \quad t = 0.2 \quad S(t) = 100 \quad \widehat{S}_0^t = 110$$

	$T = 1/4$			$T = 1/2$			$T = 1$			$T = 2$		
N_{MC}	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**	**ANA**	**MC**	**ERR**
10^2	11.47	11.36	0.11	21.61	19.16	2.45	35.67	36.13	0.46	55.79	53.42	2.37
10^3	11.47	11.49	0.02	21.61	20.98	0.63	35.67	34.37	1.3	55.79	57.34	1.55
10^4	11.47	11.37	0.1	21.61	21.33	0.28	35.67	35.36	0.31	55.79	55.33	0.46
10^5	11.47	11.40	0.07	21.61	21.47	0.14	35.67	35.65	0.02	55.79	55.87	0.08

Figure 2: Lookback Option Analytic Value Validation

Note that in both cases the Monte Carlo simulations converge towards the analytical solutions. From this we can confirm that the equations in Section 2.3 are indeed correct. Hence, we will use these expressions to compare our COS approximations to.

3.6.2 COS Density Approximation

Moving on to the COS Method, a first step in analysing this procedure is to validate the density approximation of the cos method. To do this we can, analogous to the last paragraph, compare the cos method results we get with the results we get from a Monte Carlo simulation. We want to compare the COS method approximation for the pdf the adjusted log price of a GBM at time T with the pdf of the adjusted log price from a Monte Carlo simulation. In Figure 3 you will find the achieved result for a simulation with the following parameters.

$$\mu = 0.1 \quad \sigma = 0.4 \quad t = 0.5 \quad S(t) = 100 \quad T = 1 \quad K = 100 \quad N_{COS} = 256 \quad N_{MC} = 10^5$$

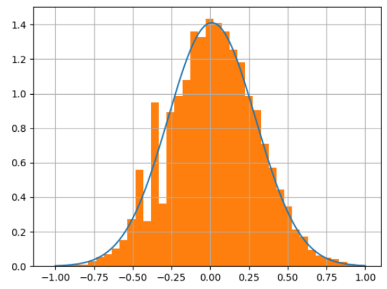


Figure 3: Density Approximation Adjusted Log Price

We can do the same regarding the pdf for the adjusted log maximum, in Figure 4 you will find the achieved result for the following parameters.

$$\mu = 0.1 \quad \sigma = 0.2 \quad t = 0.1 \quad S(t) = 100 \quad T = 1 \quad K = 100 \quad N_{COS} = 256 \quad N_{MC} = 10^5$$

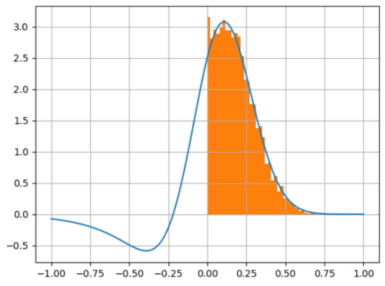


Figure 4: Density Approximation Adjusted Log Maximum

An important thing to note here is the values of the pdf for inputs below 0 $\big($as $\log(S(t)/K) = 0\big)$ do not need to be considered for our valuation process. Informally, this can be justified by the fact that the maximum of a process cannot go below its starting point. In more formal terms, the payoff for values below $\log(S(t)/K)$ is zero (see Equation (3.19)).

Unfortunately, we found that in this implementation, the GBM drift term $(r - \sigma^2/2)$ had to be positive to mimic the Monte Carlo approximation. In Figure 5 you will find an example of this, this simulation used the following parameters.

$$\mu = -0.1 \quad \sigma = 0.2 \quad t = 0.1 \quad S(t) = 100 \quad T = 1 \quad K = 100 \quad N_{COS} = 256 \quad N_{MC} = 10^5$$

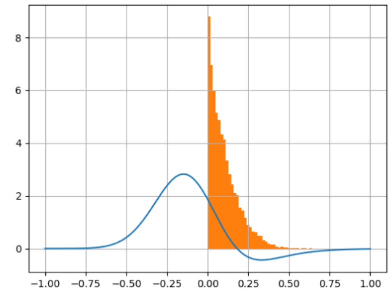


Figure 5: Density Approximation for Negative GBM Drift Term

3.6.3 COS Option Valuation

Now that we have successfully implemented COS method density approximation, we are going to look at option pricing using the COS Method. To highlight both its accuracy and its efficiency, we will compare it to both the analytical solutions from Section 2.3 and a sufficiently accurate Monte Carlo estimate. In the following table you will find the results for a European call option simulation with the following parameters.

$$\mu = 0.1 \quad \sigma = 0.2 \quad t = 0.5 \quad S(t) = 100 \quad T = 1 \quad K = 80 \quad N_{MC} = 10^5$$

For which the reference values are $22.174561\ldots, 8.277803\ldots$ and $1.418624\ldots$ respectively.

N	16	32	64	128	256	Monte Carlo
msec.	1.0	1.0	2.0	3.6	8.1	4.0E+3
abs. error $K = 80$	6.21E+02	5.11E+01	6.19E+00	3.36E-03	4.62E-11	1.09E-01
abs. error $K = 100$	6.41E+02	1.22E+02	7.52E+00	4.17E-03	3.30E-11	5.38E-02
abs. error $K = 120$	3.17E+02	2.68E+01	8.30E+00	2.02E-04	4.62E-11	2.79E-11

Figure 6: EU Option COS Method Valuation

Next, we turn to lookback options. In the table below, you will find the results for a lookback call option simulation with the following parameters.

$$\mu = 0.05 \quad \sigma = 0.2 \quad t = 0.1 \quad S(t) = 100 \quad T = 0.5 \quad \widehat{S}_0^t = 110 \quad K = 80 \quad N_{MC} = 10^5$$

For which we have reference values $34.783529\ldots, 15.277331\ldots$ and $1.960005\ldots$ respectively.

N	16	32	64	128	256	512	Monte Carlo
msec.	1.2	1.4	3.0	9.0	14.0	28.5	4.7E+3
abs. error $K = 80$	2.96e+04	6.15e+03	1.13e+02	6.73e+00	1.36e-05	4.95e-07	2.84e-01
abs. error $K = 100$	3.30e+04	7.77e+03	9.56e+02	1.25e+01	1.86e-05	2.53e-07	5.73e-02
abs. error $K = 120$	2.72e+04	1.46e+03	9.21e+02	9.79e+00	9.96e-06	4.51e-07	1.10e-01

Figure 7: Lookback Option COS Method Valuation

4 Characteristic Functions for Exponential Levy Processes

In the previous sections we have only focused on applying the COS method using Geometric Brownian Motion as a stock dynamics. In this section we will make the step towards more general dynamics, namely the class of stochastic processes known as exponential Levy processes.

4.1 Notation and Basic Principles

To start, the aforementioned exponential Levy processes can be defined as follows, according to Oosterlee and Grzelak [6].

Definition 4.1.1 (Exponential Levy Process)**.** *An exponential Levy Process is a process $S(t) = S(0)e^{X_L}$, where $X_L(t)_{t\geqslant 0}$ starts as the origin, $X_L(0) = 0$, so that on $\big(\Omega, \mathcal{F}(t), \mathbb{P}\big)$, hte following conditions hold:*

- *Independent increments: for any $0 \leqslant s_1 < t_1 < s_2 < t_2$, the random variables $X_L(t_1) - X_L(s_1)$ and $X_L(t_2) - X_L(s_2)$ are independent.*
- *Stationary Increments: for any $0 \leqslant s < t$, $X_L(t) - X_L(s)$ is equal in distribution to $X_L(t-s)$.*
- *Stochastic continuity: for any $\epsilon > 0$ and $s, t > 0$ we have*

$$\lim_{s\to t} \mathbb{P}\big(|X_L(t) - X_L(s)| > \epsilon\big) = 0$$

- *Sample paths are right-continuous with left limits, almost surely.*

Before we can start with computing characteristic functions, we will first need to define the discretely monitored maximum, using notation from Haslip and Kaishev [16].

Remark 4.1.2 (Discretely Monitored Maximum)**.** Given issue time 0, expiry time T and a series of $q+1$ evenly spaced monitoring times $t_j = \frac{T}{q}j,\ j = 0, \cdots, q$ and a stock dynamic where $S(t_j) = S(0)e^{L_{t_j}}$. Let j^* be such that $t_{j^*} \leqslant t < t_{j^*+1}$ where $t \in [0, T]$ is the current time. We now denote

$$\widehat{S}_0^t = \max_{j=0,\cdots,j^*} S(t_j)$$

and

$$X_k = \max_{j=1,\cdots,k} L_{j^*+j} \qquad k = 1, \cdots, m = q - j^*$$

to be the observed future maximum log-return process over the monitoring times $t_{j^*+1}, \cdots, t_{j+k}$.

Now that we have defined this discrete monitoring procedure, it is important to note that by increasing the number of monitoring points, the discretely monitored maximum will approach a continuously monitored maximum as we see in Definition 2.2.2.

4.2 Characteristic Function of Discretely Measured Maximum

Now that we have established a basis for discrete monitoring we can move on to determining the characteristic functions for the asset price maximum. We can do this by using the Spitzer formula, as adapted by Öhgren [17]. This Spitzer formula establishes a recursive definition for the characteristic function of X_m as

$$\phi_{X_0}(u) = 1, \qquad \phi_{X_m} = \frac{1}{m}\sum_{k=0}^{m-1} \phi_{X_k}(u) a_{m-k}(u), \qquad \text{for } m = 1, 2, \ldots \tag{4.1}$$

where $a_{m-k}(u) = \mathbb{E}^{\mathbb{Q}}\left[e^{iuL^+_{j*+m-k}}\right]$. This recursive relation allows us to determine the characteristic function of the maximum X_m using the log return process L_{j*+i} for $i = 1, \cdots, m$, which is easily determined for exponential Levy Processes. The calculation of the $a_{m-k}(u)$ coefficients will be discusses later in Sections 4.3 and 4.6.

The disadvantage of this recursive definition for ϕ_{X_m} is that it is computationally inefficient for large values of m. Since we plan to approach a continuously monitored maximum efficiency for large m is important, therefore, we will follow a derivation from Haslip and Kaishev [16], giving us a more efficient method of determining ϕ_{X_m}.

Theorem 4.2.1 (Spitzer-recurrence Expansion Formula)**.** *The characteristic function of the log-return of the asset price maximum X_m is given by*

$$\phi_{X_m}(u) = \mathrm{Coeff}_{t^m}\left[\sum_{j=0}^{m}\frac{1}{j!}\left(\sum_{k=0}^{m}\frac{a_k(u)}{k}t^k\right)^j\right]$$

where $\mathrm{Coeff}_{t^m}(.)$ *is the coefficient of t^m in (.).*

Proof. Before we start with the proof, for brevity, we omit the dependence of $\phi_{X_m}(u)$ and $a_{m-k}(u)$ on u and introduce the notation $\phi_i = \phi_{X_i}$. We can now rewrite Expression (4.1) by multiplying both sides with m as

$$m\phi_m = \sum_{k=0}^{m}\phi_k a_{m-k} - \phi_m a_0.$$

Note that $a_0 = 1$, allowing us to rewrite this expression as

$$(m+1)\phi_m = \sum_{k=0}^{m}\phi_k a_{m-k}. \tag{4.2}$$

Let us now introduce the function

$$U(z) = \sum_{m=0}^{\infty} z^m\phi_m, \text{ with } U(0) = 1. \tag{4.3}$$

Noting that $U'(z) = \frac{\mathrm{d}}{\mathrm{d}z}U = \sum_{m=0}^{\infty} m u^{m-1}\phi_m$, we have

$$U + zU' = \sum_{m=0}^{\infty}(m+1)z^m\phi_m. \tag{4.4}$$

By multiplying Equation (4.2) by z^m on both sides and summing over m from 0 to ∞, we get

$$\sum_{m=0}^{\infty}(m+1)z^m\phi_m = \sum_{m=0}^{\infty}\sum_{k=0}^{m}\phi_k z^k a_{m-k}z^{m-k} = UA.$$

where $A(z) = \sum_{j=0}^{\infty} a_j z^j$ and $a_0 = 1$. From Equations (4.4) and (4.4) we get $U + zU' = UA$, which can be rewritten as

$$\frac{U'(z)}{U(z)} = \frac{A(z)-1}{z}$$

With this we can derive an expression for $U(t)$ as

$$U(t) = e^{\log(U(t))} = \exp\left(\int_0^t \mathrm{d}\log(U(z))\right) = \exp\left(\int_0^t \frac{A(z)-1}{z}\mathrm{d}z\right). \tag{4.5}$$

The exponent can be evaluated as follows, where we assume boundedness of a_k, allowing the use of Fubini's theorem (See Tao [11]).

$$\begin{aligned}\int_0^t \frac{A(z)-1}{z}\mathrm{d}z &= \int_0^t \frac{\sum_{j=0}^{\infty} a_j z^j - 1}{z} = \int_0^t \sum_{k=1}^{\infty} a_j z^{k-1} \\ &= \sum_{k=1}^{\infty} a_j \int_0^t z^{k-1} = \sum_{k=1}^{\infty} a_j \frac{t^k}{k}\end{aligned} \tag{4.6}$$

We can conclude this proof by combining Equations (4.3), (4.5), and (4.6) to give

$$U(t) = \sum_{m=0}^{\infty} t^m \phi_m = \exp\left(\sum_{k=1}^{\infty} a_j \frac{t^k}{k}\right) = \sum_{j=0}^{m} \frac{1}{j!}\left(\sum_{k=0}^{m} \frac{a_k(u)}{k} t^k\right)^j$$

From which the desired equation follows directly. □

Now that we have found an expression for the characteristic function of the log return of the asset price maximum, the next step is to convert this characteristic function into a discrete approximation of the characteristic function of the adjusted log maximum asset price we found in Theorem 3.3.9. We will do this, as mentioned before, by generating a characteristic function for the discrete adjusted log maximum for an increasing number of monitoring points. This procedure is defined in the following theorem.

Theorem 4.2.2 (Characteristic Function of Adjusted Log Maximum via Spitzer-recurrence Expansion)**.** *Suppose that $S(s)_{s\geqslant 0}$ is defined as in Equation* (2.1) *and that we are considering a fixed strike lookback option. We will now define the adjusted log price of the maximum of $S(s)_{p\geqslant s\geqslant q}$ as $\widehat{\zeta}_p^q = \log\left[\left(\max_{p\leqslant s\leqslant q} S(s)\right)/K\right]$. The characteristic function $\phi_{\widehat{\zeta}_t^T}$ can be approximated by*

$$\phi^m_{\widehat{\zeta}_t^T}(u) = e^{iu\log(S(t)/K)} \cdot \phi_{X_m}(u)$$

for $m \to \infty$.

Proof. First, note that we can approximate continuous monitoring by increasing the number of monitoring points

$$\widehat{\zeta}_p^q = \log\left[\left(\max_{t\leqslant s\leqslant T} S(s)\right)/K\right] = \lim_{m\to\infty} \log\left[\left(\max_{j=1,\cdots,m} S(t_{j*+j})\right)/K\right].$$

This last expression inside the limit can now be rewritten as

$$\begin{aligned}\log\left[\left(\max_{j=1,\cdots,m} S(t_{j*+j})\right)/K\right] &= \log\left[\left(\max_{j=1,\cdots,m} S(t)e^{L_{j*+j}}\right)/K\right] \\ &= \log\left(S(t)/K\right) + \max_{j=1,\cdots,m} L_{j*+j} \\ &= \log\left(S(t)/K\right) + X_m\end{aligned}$$

Giving us the desired result after applying the definition of a characteristic function. □

4.3 Analytical Calculation of GBM $a_k$oefficients

In this subsection we will derive the last thing we need before we can start with the implementation of the Spitzer formula, the a_k coefficients. From Haslip and Kaishev [16] we get the following expression (but not the proof) for these coefficients.

Theorem 4.3.1 (Fourier Transform Representation of a_i)**.** *If $A_{L_{j*+k}}(v) = \mathbb{E}^{\mathbb{Q}}\left[e^{iuL^+_{j*+k}}\right]$ exists for all $v \in (a,b)$, where $a < 0$ and $b > -\operatorname{Im} z > \frac{1}{2}$, then the characteristic function $a_k(u)$ is given by*

$$a_k(z) = 1 - \frac{i}{2\pi}\int_{i\alpha-\infty}^{i\alpha+\infty} \phi_{L_{j*+k}}(-\xi)\frac{z}{\xi(\xi+z)}\mathrm{d}\xi \tag{4.7}$$

where $\alpha > \operatorname{Im} z > \frac{1}{2}$ and $k = 1, \cdots, m$.

Proof. The proof for this theorem is based on the Cauchy residue theorem, as this goes outside of the scope of this thesis, this proof will be omitted. □

This integral can be evaluated both analytically and numerically (further elaboration on numerical calculation in Section 4.6). First, we will give the coefficients regarding GBM, form Haslip and Kaishev [16].

Theorem 4.3.2 (Coefficients for GBM)**.** *Given* $L_{j*+\tau} = \sigma W(\tau) + \mu\tau \sim N\big(\mu\tau, \sigma^2\tau_k\big)$ *where* $W(\tau)$ *is a BM,* $\mu = r - \frac{1}{2}\sigma^2$ *and* $\sigma > 0$ *then, if we denote* $\tau_k = t_{j*+k} - t_{j*}$*, we get*

$$a_k(z) = \Phi\left(-\frac{\mu\sqrt{\tau_k}}{\sigma}\right) + \frac{1}{2}e^{-\frac{1}{2}z^2\sigma^2\tau_k + i\tau_k z\mu}\,\mathrm{erfc}\left[-\frac{1}{\sqrt{\tau_k}}\left(\frac{\mu}{\sigma} + iz\sigma\right)\right]$$

where $erfc(.)$ *is the complex complementary function and* $k = 1, \cdots, m$.

Proof. In the case of GBM we us the characteristic function of a BM at time τ_k, we get $\phi_{L_{j*+k}}(u) = \exp\big(iu\mu\tau_k - \frac{1}{2}\sigma^2u^2\tau_k\big)$, where $\mu = r - \frac{1}{2}\sigma^2$. Integral (4.7) can then be written as

$$a_k(z) = 1 - a_k^{(1)}(z) + a_k^{(2)}(z) \tag{4.8}$$

where

$$\begin{aligned} a_k^{(1)} &= \frac{i}{2\pi}\int_{i\alpha-\infty}^{i\alpha+\infty} \exp\left(-iu\mu\tau_k - \frac{1}{2}\sigma^2u^2\tau_k\right)\frac{\mathrm{d}u}{u}, \\ a_k^{(2)}(z) &= \frac{i}{2\pi}\int_{i\alpha-\infty}^{i\alpha+\infty} \exp\left(-iu\mu\tau_k - \frac{1}{2}\sigma^2u^2\tau_k\right)\frac{\mathrm{d}u}{u+z}. \end{aligned}$$

We now use the substitution $u = v - \frac{i\mu}{\sigma^2}$ which gives us

$$\begin{aligned} a_k^{(1)} &= \frac{i}{2\pi}e^{-\frac{1}{2}\tau_k\mu^2/\sigma^2}\int_{i\beta-\infty}^{i\beta+\infty} e^{-\frac{1}{2}\sigma^2v^2\tau_k}\frac{\mathrm{d}v}{v - i\mu/\sigma^2}, \\ a_k^{(2)}(z) &= \frac{i}{2\pi}e^{-\frac{1}{2}\tau_k\mu^2/\sigma^2}\int_{i\beta-\infty}^{i\beta+\infty} e^{-\frac{1}{2}\sigma^2v^2\tau_k}\frac{\mathrm{d}v}{v + z - i\mu/\sigma^2}, \end{aligned}$$

where $\beta = \alpha + \mu/\sigma^2$. Our next step is shifting down the integration contour in both integrals to the real axis, picking up residues at poles $i\mu/\sigma^2$ resp. $-z + i\mu/\sigma^2$. Via the definition of the Faddeeva function from Weideman [18], we obtain

$$\begin{aligned} a_k^{(1)} &= \frac{i}{2\pi}e^{-\frac{1}{2}\tau_k\mu^2/\sigma^2}\int_{-\infty}^{\infty} e^{-\frac{1}{2}\sigma^2v^2\tau_k}\frac{\mathrm{d}v}{v - i\mu/\sigma^2} + 1 = 1 - \frac{1}{2}\,\mathrm{erfc}\left[\frac{\mu}{\sigma}\sqrt{\frac{1}{2}\tau_k}\right], \\ a_k^{(2)}(z) &= \frac{i}{2\pi}e^{-\frac{1}{2}\tau_k\mu^2/\sigma^2}\int_{-\infty}^{\infty} e^{-\frac{1}{2}\sigma^2v^2\tau_k}\frac{\mathrm{d}v}{v + z - i\mu/\sigma^2} + e^{-\frac{1}{2}\sigma^2\tau_k z^2 + itz\mu} \\ &= e^{-\frac{1}{2}\sigma^2\tau_k z^2 + i\tau_k z\mu}\left(1 - \frac{1}{2}\,\mathrm{erfc}\left[\left(\frac{\mu}{\sigma} + z\sigma i\right)\sqrt{\frac{1}{2}\tau_k}\right]\right), \end{aligned}$$

where erfc [.] is the complementary error function, which can also be found in Weideman [18]. Combining these expressions via Equation (4.8) gives

$$a_k(z) = \frac{1}{2}\,\mathrm{erfc}\left[\frac{\mu}{\sigma}\sqrt{\frac{1}{2}\tau_k}\right] + e^{-\frac{1}{2}\sigma^2\tau_k z^2 + i\tau_k z\mu}\left(1 - \frac{1}{2}\,\mathrm{erfc}\left[\left(\frac{\mu}{\sigma} + z\sigma i\right)\sqrt{\frac{1}{2}\tau_k}\right]\right).$$

Finally we can use the properties of this complementary error function that $\mathrm{erfc}\,[x] = 2\Phi(x\sqrt{2})$ and $\frac{1}{2}\,\mathrm{erfc}\,[-x] = (1 - \frac{1}{2}\,\mathrm{erfc}\,[x])$ to give

$$a_k(z) = \Phi\left(-\frac{\mu\sqrt{\tau_k}}{\sigma}\right) + \frac{1}{2}e^{-\frac{1}{2}z^2\sigma^2\tau_k + i\tau_k z\mu}\,\mathrm{erfc}\left[-\sqrt{\frac{\tau_k}{2}}\left(\frac{\mu}{\sigma} + z\sigma i\right)\right]$$

concluding our proof. □

4.4 The Algorithm

In this subsection we will implement Theorem 4.2.1, as a starting point to the implementation we will take the algorithm described in pseudocode in Haslip and Kaishev [16]. The first and most important alteration we will make to this algorithm is that of vectorisation. The algorithm in Haslip and Kaishev [16] takes as an input a value for m and a value for u, then outputting $\phi_{X_m}(u)$. Because the algorithm is the same for each u input, this implementation will turn out needlessly slow for large values of m. The solution to this is vectorisation, the rewriting of the algorithm to take in a array of values $(u_i)_{0<i<N}$ and m, returning an array $\left(\phi_{X_m}(u_i)\right)_{0<i<N}$. We combine the resulting algorithm with Theorem 4.2.2 to give the following python implementation (small alterations due to computing limitations are excluded for readability. For the complete program, including the implementation of this method into the COS method, see B).

```python
def spitzer_recurrence(self, m : int, u : np.array, t, T):
    p = np.zeros((m, m, u.size), dtype=complex)
    for I in range(m):
        p[0,I] = self.ana_a(I, u, t, (T-t)/m) / (I+1)

    out = p[0, m-1]

    for I in range(1, m):
        for J in range(m - I):
            for k in range(J+1):
                p[I, J] += p[I-1, k] * p[0, J-k]
        out += p[I, m-1-I] / math.factorial(I+1)

    return np.exp(i*u*np.log(x/K)) * out
```

In this snippet `def ana_a(...)` is defined as in Theorem 4.3.2.

4.5 Results

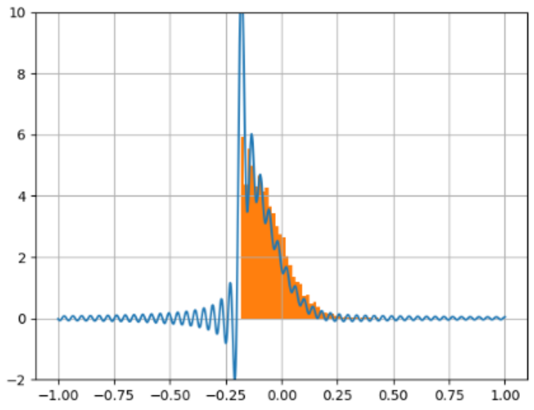
Finally, we can use this algorithm to compute our first density estimates for the maximum of a GBM. As a first test, we will examine the effect of different choices for m on the approximation of the density function. In Figure 8, we have shown approximations for the pdf of the adjusted log maximum corresponding to the following parameters, with different values of m. In an aim to only examine the effect the chosen m has no these approximations, we have chosen the highest amount of cos iterations our machine allows.

$$\mu = 0.05 \quad \sigma = 0.2 \quad t = 0.5 \quad S(t) = 100 \quad T = 1 \quad K = 120 \quad N_{COS} = 1024 \quad N_{MC} = 10^5 \tag{4.9}$$

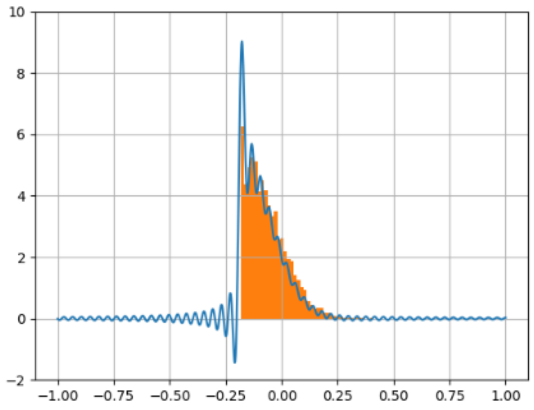
In an attempt to gauge the effect the choice for m has on the accuracy of the approximation and on the computation time, we have devised Figure 9. As a measure for accuracy, we have chosen the maximum of the approximation. This is of course not a definitive measure for function accuracy, but in our case it does give an indication.

There are two things we can take from this table, firstly, that with doubling the amount of iterations, the computation time roughly increases by a factor 7. Also, considering the difference in maximum scales down by roughly 1.69 each iteration, we can determine the limit for this maximum will be around 5.45 (for the algorithm determining this limit, see Appendix C). Tanking these two factors into consideration, we conclude that approximations for $m = 800$ provide the optimal balance between accuracy and computational cost.

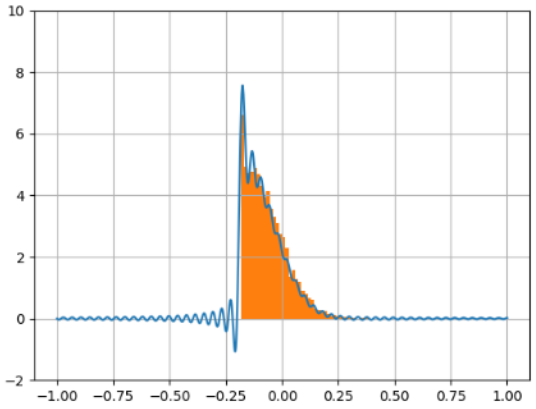
A last step we will take in approving the current approximations is the introduction of spectral filters. This was introduced to the COS method in Ruijter, Versteegh and Oosterlee [19] to account for the *Gibbs Phenomenon*, being the result of Fourier approximations overshooting in cases of jump discontinuity. Though the inaccuracies in Figure 8 are not, at least not completely, the result of Fourier approximation results, they do look similar. Thus, in an attempt to cancel out these inaccuracies, we will follow the procedure in Ruijter, Versteegh and Oosterlee [19]. As a first step in this procedure, we will define spectral filters as follows.



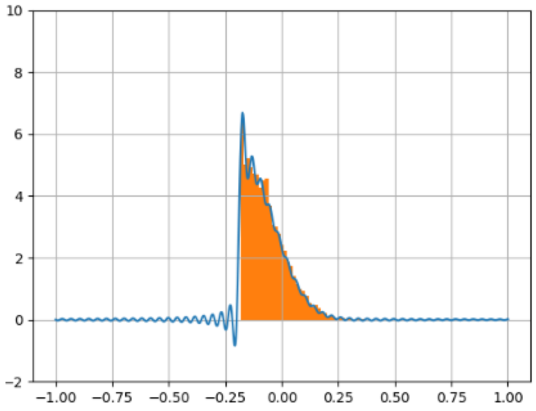
(a) $m = 25$



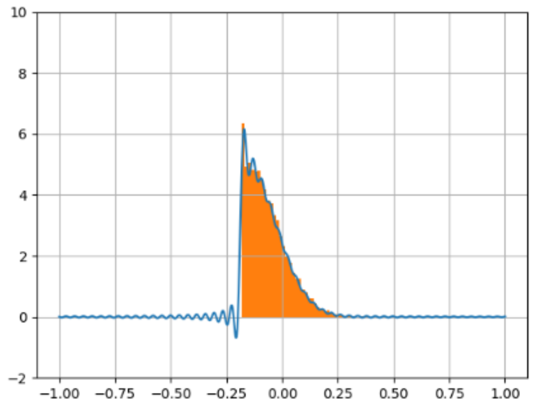
(b) $m = 50$



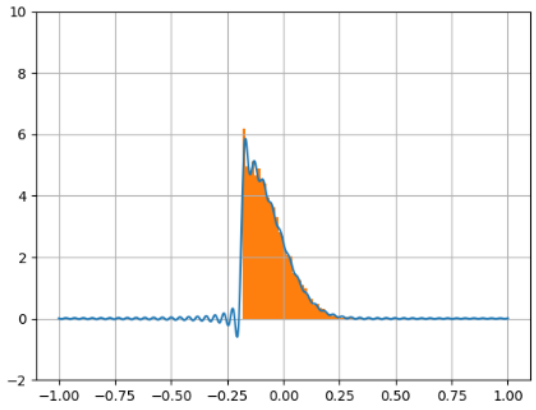
(c) $m = 100$



(d) $m = 200$



(e) $m = 400$



(f) $m = 800$

Figure 8: Density Approximation via Spitzer Formula

m	25	50	100	200	400	800
max	11.370	9.029	7.569	6.688	6.164	5.859
Δt	0.0412	0.145	1.001	6.703	56.152	443.817

Figure 9: Effect of Choice of m on Maximum and Computation Time

Definition 4.5.1 (Spectral Filter)**.** *Any $C^\infty([0,1])$ even function $\hat{s}$, whose support is $[-1,1]$ is a filter if*

(i) $\hat{s}(0) = 1$

(ii) $\hat{s}^(k)(0) = 0$ *for some* $k \geqslant 1$

(iii) $\hat{s}(\eta) = 0$ *for* $|\eta| \geqslant 1$

The filtering is carried out by premultiplying the COS Fourier-expansion coefficients by a filtering factor to make it so that the higher order Fourier coefficients get dampened, leading to a smoother density approximation, and ultimately more efficient convergence. This gives us the following transformed COS method valuation function, derived from Theorem 3.1.7, from Ruijter, Versteegh and Oosterlee [19].

Theorem 4.5.2 (Filter-COS Method)**.** *Given a suitable filter $\hat{s}$, the filter-COS method pricing formula is acquired by multiplying each of the Fourier-coefficients by a factor $\hat{s}(k/N)$, giving us*

$$V_{\mathrm{IV}}(t_0, x) := e^{-r\tau} {\sum_{k=0}^{N-1}}' \hat{s}\left(\frac{k}{N}\right) Re\left\{\phi_X\left(\frac{k\pi}{b-a}\right) \cdot exp\left(-i\frac{ka\pi}{b-a}\right)\right\} \cdot H_k.$$

Similarly, the filter-COS method density expression, derived from Definition 3.1.4, is given by

$$f_{X,\mathrm{III}}(y) = {\sum_{k=0}^{N-1}}' \hat{s}\left(\frac{k}{N}\right) \frac{2}{b-a} Re\left\{\phi_X\left(\frac{k\pi}{b-a}\right) \cdot exp\left(-i\frac{ka\pi}{b-a}\right)\right\} \cos\left(k\pi\frac{y-a}{b-a}\right).$$

Regarding the choice of what filter to use, Ruijter, Versteegh and Oosterlee [19], gives a couple options. Three viable options are listed below.

- *Fejér Filter:* $\hat{s}(\eta) = 1 - |\eta|$
- *Raised Cosine Filter:* $\hat{s}(\eta) = \frac{1}{2}(1 + \cos(\pi\eta))$
- *Exponential Filter:* $\hat{s}(\eta) = exp(-\alpha\eta^p)$, where p is even and $\alpha = -\log \epsilon_m$, ϵ_m being the machine epsilon.

To test the compatibility of each of these filters with our simulations, Figure 10 shows simulations for the pdf of a GBM maximum process with parameters as in Expression (4.9) $m = 400$ for these filters, choosing $p = 2, 4$ for the exponential filter.

What we conclude from these figures is that for the chosen parameters the raised cosine filter is most suitable. To show the versatility of these density approximates, Figure 11 shows Spitzer approximations for an array of GBM parameters.

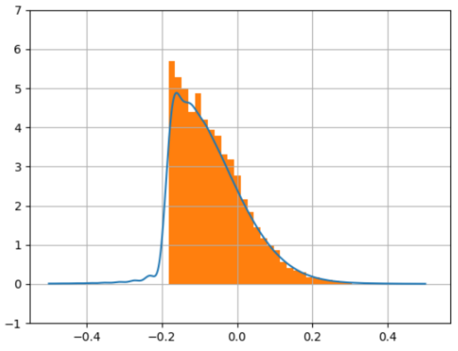
Next, we will look at the COS method value approximations using the Spitzer algorithm. In Table 12 we have listed value approximations for a lookback option contract with the following parameters, for various values of N_{COS} and m.

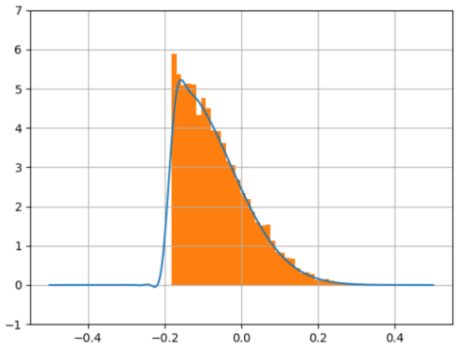
$$\mu = 0.05 \quad \sigma = 0.2 \quad t = 0.5 \quad S(t) = 100 \quad T = 1 \quad \widehat{S}_0^t = 110 \quad K = 120 \quad N_{MC} = 10^5$$

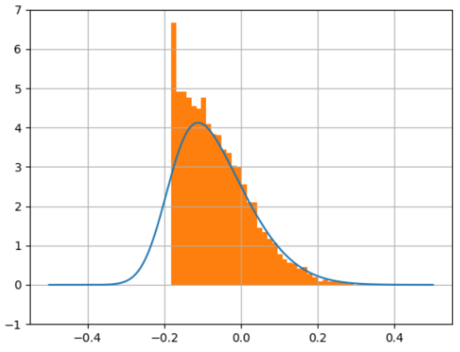
For reference, the analyitic option value for this simulation is $6.149784\ldots$.

4.6 Numerical Calculation of $a_k$oefficients

In an aim to make our algorithm compatible for any exponential Lévy process for which you can determine the characteristic function of the underlying levy process, we attempted at implementing numerical calculation of the integral in Equation (4.7). The algorithm we used for this is the trapezoid rule of integration. We found that these estimates were not accurate, see Figure 13. According to Haslip and Kaishev [16] though, other numerical integration methods like the Gauss-Kronrod method, or an adaptation of the pricing method in their paper do work for general exponential Lévy processes.


(a) Fejér Filter


(b) Raised Cosine Filter


(c) Exponential Filter, $p = 2$

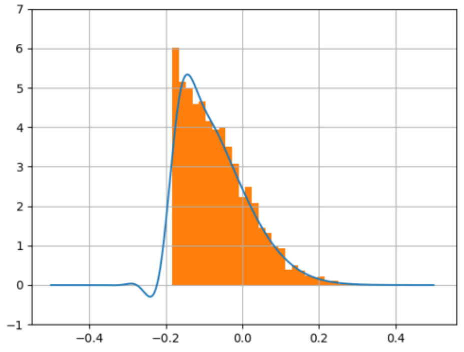
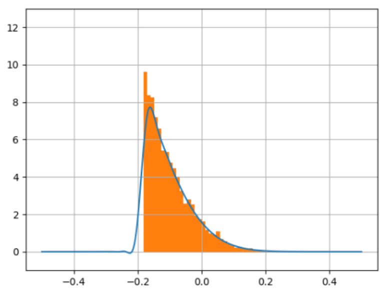
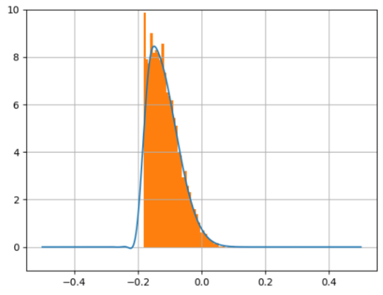
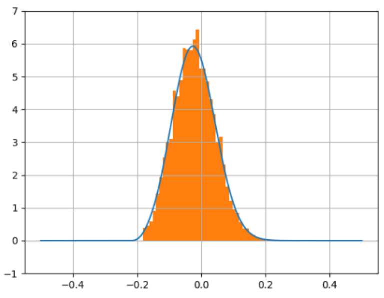
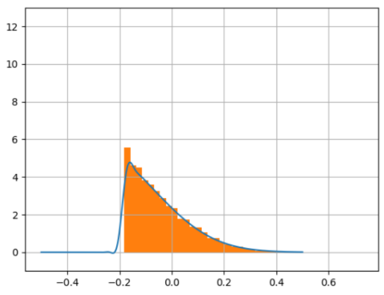

(d) Exponential Filter, $p = 4$

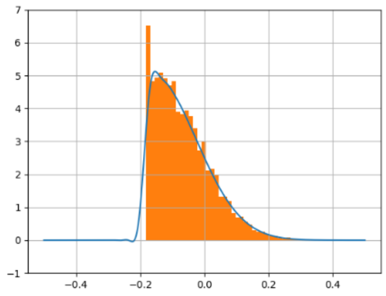
Figure 10: Density Approximation via Spitzer Formula


(a) $\mu = -0.1,\ \sigma = 0.2$


(b) $\mu = 0.05,\ \sigma = 0.1$


(c) $\mu = 0.3,\ \sigma = 0.1$


(d) $\mu = -0.05,\ \sigma = 0.3$


(e) $\mu = 0.05,\ \sigma = 0.2$

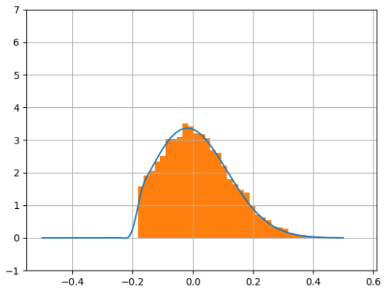

(f) $\mu = 0.3,\ \sigma = 0.2$

Figure 11: Spitzer Density Approximation for Various GBM Parameters

	N_{COS}				
m	64	128	256	512	1024
25	10.282	5.483	3.921	3.503	3.396
50	11.285	6.351	4.747	4.314	4.204
100	11.999	6.984	5.359	4.919	4.807
200	12.496	7.430	5.793	5.348	5.235
400	12.837	7.737	6.093	5.646	5.532
800	13.071	7.949	6.299	5.851	5.736

Figure 12: COS Method Estimates for Increasing m (Analytic Value is $6.149784\ldots$)

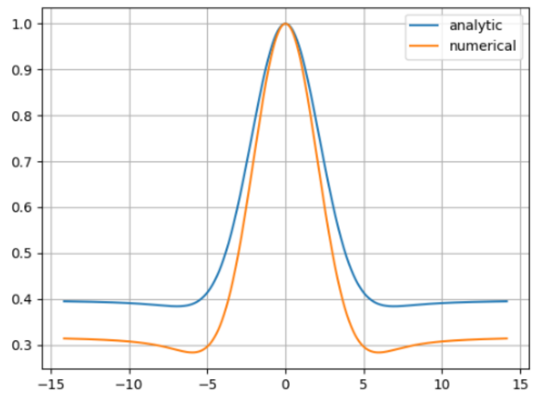


Figure 13: Calculation of a_k coefficients for GBM ($k = m = 1000,\ \mu = 0.1,\ \sigma = 0.3,\ T = 2$)

5 Conclusion

This thesis has introduced the COS method of option pricing to the pricing of fixed strike lookback options. The COS method for option pricing is a numerical option pricing method dependent solely on the characteristic function of a stochastic process for an asset dynamic and an option specific payoff function. To be able to apply this method to lookback option pricing, we derived the characteristic function describing the maximum process of a Geometric Brownian Motion, together with working out the COS method payoff coefficients.

A Python implementation of the aforementioned pricing method and parameters showed that COS method value estimates converged exponentially towards the analytical solutions, far outperforming for example Monte Carlo simulations. We did find however that this pricing method worked only in cases where the GBM drift parameter $(r - \sigma^2/2) > 0$. Though this may look like a quite significant curtailment, with the current European interest rate at $r = 0.04$, this allows for asset volatility in the range $0 < \sigma \leqslant 0.283$. Looking at historical data of the CBOE Volatility Index [20], this is an acceptable range.

In the last chapter we introduced an alternative approach to determining the characteristic function of a process' maximum. This new approach is noteworthy because of its compatibility with a whole group of asset dynamics, namely all exponential Lévy processes. Additionally, this approach lifts the previously mentioned restriction to the GBM drift term. This approach should however be seen as a proof of concept, as the COS method value estimates acquired via this method were not accurate. The last paragraph of the last chapter also hinted at numerical calculation of a_k coefficients, but our (albeit primitive) implementation did not generate workable results.

6 Discussion

During this project we came across a number of difficulties or possible improvements that we did not have the possibility to expand upon, in this section we will discuss these points.

A first and obvious point of improvement to our method, as previously mentioned in the conclusion, is further research into the restriction posed by the deterioration of the COS method for lookback option pricing under GBM where the drift term is negative. We have studied our implementation extensively, and could only conclude that it was in line with the literature.

A second, purely computational, improvement that could be made to our implementation is a different choice of programming language. Python is a very readable and easily implemented programming language, making it ideal for the demonstration of numerical methods such as the COS method. The drawback however is that it is very slow in comparison to compiled programming languages like C#, C++, or Rust. The improvement a different choice in programming language would make is the significant speeding up of calculations (see [21]) for ϕ_{X_m}. We found that due to the limitations of python, we could not compute estimates for $m > 800$. A possible result of the use of these higher order approximations is the improvement of value approximations, as we found that for an increasing m, the estimates got closer to the analytical option value.

A third, also computational improvement, could be the use of a 128-bit machine (also rewriting our program to be 128-bit compatible), where we only had access to a 64-bit machine. In our calculations we noticed that for the program to work with sufficiently high values of N_{COS} and m, we had to limit the input values of the GBM variance term t0 $\sigma < 0.3$, and values of the COS iterations to $N_{\text{COS}} \leqslant 1024$, which is of course not desirable.

Lastly, apart from improvements to our method, an obvious expansion could be made to this project by including the valuation of lookback put options. We did make a significant effort to include this in this work, but did not manage to generate positive results. The literature does mention that with small alterations to our current method, this can be achieved, see [14–16].

Acknowledgements

I would like to thank my supervisors Kees Oosterlee and Bálint Négyesi for providing me with comments and suggestions all throughout the process of my thesis. Also, I would like to thank Nico Temme for his help with the last chapter of my thesis.

References

[1] F. Fang and C. W. Oosterlee. “A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions”. In: *SIAM Journal on Scientific Computing* 31.2 (2009), pp. 826–848. DOI: 10.1137/080718061.

[2] J. de la Vega. *Confusion des Confusiones*. Amsterdam, Netherlands, 1688. ISBN: 9781614274513.

[3] L. Petram. *De bakermat van de beurs*. Atlas Contact, 2013. ISBN: 9789045024677. URL: https://books.google.nl/books?id=VOOvAgAAQBAJ.

[4] R.C. Michie. *The London Stock Exchange: A History*. Oxford University Press, 1999. ISBN: 9780198295082. URL: https://books.google.nl/books?id=DygTDAAAQBAJ.

[5] F. Black and M. Scholes. “The Pricing of Options and Corporate Liabilities”. In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. DOI: 10.1086/260062.

[6] C. W. Oosterlee and L. A. Grzelak. *Mathematical Modeling and Computation in Finance*. World Scientific, 2019. ISBN: 9781786347947. DOI: 10.1142/q0236.

[7] S. E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Springer Finance Textbooks. Springer, 2004. ISBN: 9780387401010. URL: https://link.springer.com/book/9780387401010.

[8] “Path Dependent Options: The Case of Lookback Options”. In: *The Journal of Finance* 46.5 (1991), pp. 1893–1907. DOI: 10.1111/j.1540-6261.1991.tb04648.x.

[9] Phelim P. Boyle. “Options: A Monte Carlo approach”. In: *Journal of Financial Economics* 4.3 (1976), pp. 323–338. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(77)90005-8.

[10] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Second. Dover Books on Mathematics. Dover Publications, 2001. ISBN: 9780486411835. URL: https://link.springer.com/book/9783540514879.

[11] T. Tao. *Analysis II*. Third. Texts and Readings in Mathematics. Springer Nature Singapore, 2016. ISBN: 9789811018046. DOI: 10.1007/978-981-10-1804-6.

[12] A.J. Weir. *Lebesgue Integration and Measure*. University Press Cambridge, 1973, pp. 93–118. ISBN: 0521087287. DOI: 10.1017/9781139171458.

[13] S. J. Dilworth and D. Wright. “A Direct Proof of the Reflection Principle for Brownian Motion”. In: *Theory of Stochastic Processes.* 21.2 (2016), pp. 1–3. DOI: 10.37863/tsp.

[14] N. Privault. *Notes on Stochastic Finance*. Second. CRC Press, 2023, pp. 369–397. ISBN: 9781032288260. DOI: 10.1201/9781003298670.

[15] C. Wang, H. Zou, and J. Yin. “Fourier Transform of Lookback Option Price”. In: *ISRN Applied Mathematics* (2011), p. 518172. ISSN: 2356-7872. DOI: 10.5402/2011/518172.

[16] G. G. Haslip and V. K. Kaishev. “Lookback option pricing using the Fourier transform B-spline method”. In: *Quantitative Finance* 14.5 (2014), pp. 789–803. DOI: 10.1080/14697688.2014.882010.

[17] A. Öhgrens. “A remark on the pricing of discrete lookback options”. In: *Journal of Computational Finance* 4.3 (2001). DOI: 10.21314/JCF.2001.069.

[18] J. Weideman. “Computation of the Complex Error Function”. In: *SIAM Journal on Numerical Analysis* 31.5 (1994), pp. 1497–1518. ISSN: 00361429.

[19] M. J. Ruijter, M. Versteegh, and C. W. Oosterlee. “On the application of spectral filters in a Fourier option pricing technique”. In: *Journal of Computational Finance* (2013). DOI: 10.2139/ssrn.2266323.

[20] CBOE. *Yahoo Finance - VIX Index*. Accessed on 2023/06/20. URL: https://finance.yahoo.com/quote/%5C%5EVIX/.

[21] N. Heer. *Speed Comparison of Programming Languages*. Accessed on 2023/06/21. URL: https://github.com/niklas-heer/speed-comparison.

A Implementation of COS Method

```python
import numpy as np
import matplotlib.pyplot as plt
from classes import *
from colorama import Fore, Style
import time

# function verifying analytical solutions for lookback options via monte carlo estaimtes
def MonteCarloDemo():
    print('\t|\t|          T=1/4\t|          T=1/2\t|          T=1\t\t|      T=2\t\t|')
    print(f'\t|    K\t|   ANA\t|    MC\t|   {Fore.RED}ERR{Style.RESET_ALL}\t|   ANA\t|    MC\t|   {Fore.RED}ERR{Style.RESET_ALL}\t|   ANA\t|    MC\t|  {Fore.RED}ERR{Style.RESET_ALL}\t|   ANA\t|    MC\t|  {Fore.RED}ERR{Style.RESET_ALL}\t|')
    # for loops for different option parameters
    for strike in [9, 10, 11]:
        for var in [CALL, PUT]:
            print(f'{var.value.lower()}\t| {strike}\t', end='', flush=True)
            for T in [1/4, 1/2, 1, 2]:
                gbm = GBM(0.1, 0.4)
                lower = -10*np.sqrt(2*T)
                upper =  10*np.sqrt(2*T)

                t = 0.2
                St = 9
                if var == CALL : prevex = 10
                if var == PUT  : prevex = 8

                # defining the option
                lb = Lookback(gbmprocess=gbm,
                              exercisetime=T,
                              strikeprice=strike,
                              optionvariant=var,
                              lowerintegration=lower,
                              upperintegration=upper)
                analytic   = round(lb.analytic(St, t, prevex), 2)
                montecarlo = round(lb.MonteCarlo(10000, 1000, t, St, prevex), 2)
                error      = round(abs(analytic - montecarlo), 2)
                out = ''
                out += f'| {"{:.2f}".format(analytic)} \t'
                out += f'| {"{:.2f}".format(montecarlo)}\t'
                out += f'| {Fore.RED}{str(error)}{Style.RESET_ALL}\t'
                print(out, end='', flush=True)
            print('|', flush=True)

# function returning the error a particular cos estimate makes
def COS_Error(option, value, time, N, previousextreme=None):
    estimate = COS_Estimate(option, value, time, N, previousextreme)

    if isinstance(option, European):
        analytic = option.analytic(value, time)
    elif isinstance(option, Lookback):
        analytic = option.analytic(value, time, previousextreme)

    return abs(estimate - analytic)

# function returning a cos method pdf estimate for given parameters
def COS_Estimate(option, value, time, N, previousextreme=None):
    tau = option.T - time
    ks = np.linspace(0, N-1, N)  # defining an array with all values of k
    us = ks * np.pi / (option.b - option.a)  # defining an array of the factor
        k*pi/(b-a)

    # defining the array of F_k values
    Fks = np.array([np.real(option.phi_adjlog_from(u, value, time) * np.exp(-i * u
        * option.a)) for u in us])
    Fks[0] *= 0.5

    # defining the array of H_k values according to the option specific function
    if isinstance(option, European):
        Hs = np.array([option.H(k) for k in ks])
    elif isinstance(option, Lookback):
        Hs = np.array([option.H(k, previousextreme) for k in ks])

    # multiplying the Fk values by the Hk values to give cos method sum
    vs = np.array([F * H for F, H in zip(Fks, Hs)])

    return np.exp(-option.process.mu * tau) * np.sum(vs)

# function returning the cos density estimate (vectorised)
def COS_Density(option, value, time, N, y):
    ks = np.linspace(0, N-1, N)  # defining an array with all values of k
    us = ks * np.pi / (option.b - option.a)  # defining an array of the factor
        k*pi/(b-a)

    # defining the array of F_k values
    Fks = np.array([(2.0 / (option.b - option.a)) *
        np.real(option.phi_adjlog_from(u, value, time) * np.exp(-i * u *
        option.a)) for u in us])
    Fks[0] *= 0.5

    # defining array of cos(u*(y-a))
    cs = np.cos(np.outer(us, y - option.a))

    # inproduct of F_ks and cs => desired function
    f_X = np.matmul(Fks, cs)

    return f_X

# function plotting the output of COS_Density() + monte carlo for comparison
def COS_Plot(option : Option, value, time, ns):
    y = np.linspace(-1, 1, 1000)

    # plotting the resulting density function for multiple values of N
    for n in ns:
        f_X = COS_Density(option, value, time, n, y)
        plt.plot(y, f_X, label=f'N={n}')

    values = option.MonteCarlo(10000, 1000, time, value, prevex=value,
        adjlogs=True)
    plt.hist(values, density=True, bins=40, color='k')
    plt.legend()
    plt.savefig('./option_pricing/fig.png')
```

```python

# Test function for european options
def EU(t, St, K, variant, T, mu, sigma):
    gbm = GBM(mu, sigma)
    lower = -10*np.sqrt(T-t)
    upper =  10*np.sqrt(T-t)

    # defining the option parameters
    eu = European(stockmechanic=gbm,
                  exercisetime=T,
                  strikeprice=K,
                  optionvariant=variant,
                  lowerintegration=lower,
                  upperintegration=upper)

    # generating analytical solution and monte carlo estimate
    analytic   = eu.analytic(St, t)
    t_start    = time.time()
    montecarlo = eu.MonteCarlo(10000, 1000, t, St)
    t_stop     = time.time()
    print(f'analytic:    {analytic}')
    print(f'monte carlo: {"{:.2e}".format(montecarlo - analytic)}, this took
       {round(1000*(t_stop - t_start), 2)} msec\n')

    # generating cos method value error and duration for given N_cos values
    ns = [16, 32, 64, 128]
    for n in ns:
        t_start = time.time()
        err = "{:.2e}".format(abs(COS_Estimate(eu, St, t, n) - analytic))
        t_stop  = time.time()
        print(f'N={n},\t err={err},\t this took {round(1000*(t_stop - t_start),
            3)} msec')

    # plotting the estimated pdf for given N_cos values
    COS_Plot(eu, St, t, ns)

# Test function for lookback options
def LB(t, St, K, prevex, variant, T, mu, sigma):
    gbm = GBM(mu, sigma)
    lower =  np.log(St/K)
    lower =  -10*np.sqrt(T)
    upper =  10*np.sqrt(T)

    # defining the option parameters
    lb = Lookback(stockmechanic=gbm,
                  exercisetime=T,
                  strikeprice=K,
                  optionvariant=variant,
                  lowerintegration=lower,
                  upperintegration=upper)

    # generating analytical solution and monte carlo estimate
    analytic   = round(lb.analytic(St, t, prevex), 6)
    t_start = time.time()
    montecarlo = round(lb.MonteCarlo(10000, 1000, t, St, prevex), 2)
    t_stop  = time.time()
    print(f'analytic:    {analytic}')
    print(f'monte carlo: {"{:.2e}".format(abs(montecarlo - analytic))}, this took
       {round(1000*(t_stop - t_start), 2)} msec\n')

    # generating cos method value error and duration for given N_cos values
    ns = [16, 32, 64, 128, 256, 512]
    for n in ns:
        t_start = time.time()
        err = "{:.2e}".format(abs(COS_Estimate(lb, St, t, n, prevex) - analytic))
        t_stop  = time.time()
        print(f'N={n},\t err={err},\t this took {round(1000*(t_stop - t_start),
            5)} msec')

    # plotting the estimated pdf for given N_cos values
    COS_Plot(lb, St, t, ns)

# EU(t=0.5, St=100, K=120, variant=CALL, T=1, mu=-0.1, sigma=0.2)
LB(t=0.5, St=100, prevex=110, K=120, variant=CALL, T=1, mu=0.05, sigma=0.2)
```

```python
import numpy as np
import math
import random
from enum import Enum
from scipy.stats import norm
import sys
from colorama import Fore, Style
np.seterr(all='raise')
np.set_printoptions(linewidth=np.inf)

# defining imaginary unit
i = complex(0.0, 1.0)

# custom class for option variants
class OptionVariant(Enum):
    PUT = 'PUT'
    CALL = 'CALL'
PUT = OptionVariant.PUT
CALL = OptionVariant.CALL

# sum prime operator from grzelak and oosterlee
def sump(array):
    if array.size == 0:
        return 0
    res = np.sum(array) - array[0]/2
    return res

# charactersitic function standard normal distribution
def phi_N(u):
    return np.exp(-0.5 * u**2)

# class defining a geometric brownian motion and characteristic functions for
    processes derived from geometric brownian motions
class GBM:
    def __init__(self, mu, sigma):
        self.mu     = mu
        self.sigma = sigma

    # characteristic function for adjusted log gbm process
    def phi_adjlog(self, u, x, t, T, K):
        tau = T - t

        e1 = i*u*np.log(x/K)
        e2 = i*u*(self.mu - 0.5*self.sigma**2)*tau
        e3 = -0.5*tau*(self.sigma*u)**2

        return np.exp(e1 + e2 + e3)

    # characteristic function for adjusted log maximum gbm process
    def phi_adjlogmax(self, u, x, t, T, K):
        tau = T - t
        alpha = (self.mu - 0.5*self.sigma**2) / self.sigma

        f1 = 2 * np.exp(i*u*np.log(x/K))
        f2 = (1 - (alpha / (i*self.sigma*u + 2*alpha)))
        f3 = np.exp(-i * (self.sigma*u - 2*alpha*i) * tau * alpha)
        f4 = phi_N(math.sqrt(tau) * (self.sigma * u - 2*alpha*i))

        return f1 * f2 * f3 * f4

# parent class defining option contracts
class Option:
    def __init__(self, stockmechanic, exercisetime, strikeprice, optionvariant,
        lowerintegration, upperintegration):
        self.process = stockmechanic
        self.mu      = stockmechanic.mu
        self.sigma   = stockmechanic.sigma
        self.T       = exercisetime
        self.strike  = strikeprice
        self.optvar  = optionvariant
        self.a       = lowerintegration
        self.b       = upperintegration

    # function used in H_k calculation
    def chi(self, k, c, d):
        return 1/(1 + (k * np.pi / (self.b-self.a))**2) * (
            np.cos(k*np.pi*(d-self.a)/(self.b-self.a)) * np.exp(d)
          - np.cos(k*np.pi*(c-self.a)/(self.b-self.a)) * np.exp(c)
          + np.sin(k*np.pi*(d-self.a)/(self.b-self.a)) * np.exp(d) * (k*np.pi) /
              (self.b-self.a)
          - np.sin(k*np.pi*(c-self.a)/(self.b-self.a)) * np.exp(c) * (k*np.pi) /
              (self.b-self.a))

    # function used in H_k calculation
    def psi(self, k, c, d):
        if k == 0:
            return (d-c)
        else:
            return (np.sin(k*np.pi*(d-self.a)/(self.b-self.a)) -
                np.sin(k*np.pi*(c-self.a)/(self.b-self.a))) * (self.b-self.a) /
                (k*np.pi)

    # function giving monte carlo estimate of option value (showing a progress
        percentage furing calculation)
    def MonteCarlo(self, iterations, steps, t, st, prevex=None, adjlogs = False):
        vs = np.zeros(iterations)

        percentage = 0.0
        print("{: >6.2f}".format(percentage) + '%', end='', flush=True)

```

```python
        # monte carlo iterations loop
        for i in range(iterations):
            newperc = round(100*i/iterations, 2)
            if newperc != percentage:
                percentage = newperc
                sys.stdout.write('\b' * 7 + ' ' * 7 + '\b' * 7)
                print(f'{Fore.MAGENTA + Style.BRIGHT}{"{:>6.2f}".format(newperc)}%{Style.RESET_ALL}', end='', flush=True)

            # generating sample path and determining payoff
            gbm = self.samplepath(self.T - t, steps, st)
            vs[i] = self.payoff(gbm, prevex, adjlogs)

        sys.stdout.write('\b\b\b\b\b\b\b')

        # return all value endpoints for monte carlo graphing
        if adjlogs: return vs

        # returning monte carlo estimate
        else:       return np.exp(-self.mu*(self.T-t)) * np.average(vs)

    # function generating a gbm sample path
    def samplepath(self, tau, steps, st):
        srw = (1 / np.sqrt(steps)) * np.cumsum([0] + [random.choice([-1, 1]) for _ in range(int(steps * tau) - 1)])
        ts  = np.arange(int(steps * tau)) / steps

        return st * np.exp((self.mu - np.power(self.sigma, 2)/2) * ts + self.sigma * srw)


# child class describing lookback option contracts
class Lookback(Option):
    # function determining payoff for a given path
    def payoff(self, path, prevex, adjlogs=False):
        mx = np.max(np.append(path, prevex))
        mn = np.min(np.append(path, prevex))

        if adjlogs:
            if self.optvar == CALL: return np.log(mx/self.strike)
            if self.optvar == PUT:  return np.log(mn/self.strike)

        if self.optvar == CALL: return np.maximum(mx - self.strike, 0)
        if self.optvar == PUT:  return np.maximum(self.strike - mn, 0)

    # determining payoff coefficients
    def H(self, k, prevex):
        adjprev = np.log(prevex/self.strike)

        tc1 = tc2 = tc3 = 0
        tp1 = tp2 = tp3 = 0

        if prevex > self.strike:
            tc1 = self.psi(k, self.a, adjprev) * 2*(prevex-self.strike)/(self.b-self.a)
            tc2 = (self.chi(k, adjprev, self.b) - self.psi(k, adjprev, self.b)) * 2*self.strike/(self.b-self.a)
            tp3 = (self.psi(k, self.a, 0) - self.chi(k, self.a, 0)) *
```

```
                    2*self.strike/(self.b-self.a)
        else:
            tc3 = (self.chi(k, 0, self.b) - self.psi(k, 0, self.b)) *
                2*self.strike/(self.b-self.a)
            tp1 = self.psi(k, adjprev, self.b) *
                2*(self.strike-prevex)/(self.b-self.a)
            tp2 = (self.psi(k, self.a, adjprev) - self.chi(k, self.a, adjprev)) *
                2*self.strike/(self.b-self.a)

        if self.optvar is CALL: return tc1 + tc2 + tc3
        if self.optvar is PUT:  return tp1 + tp2 + tp3

    # funtion returning characteristic function for maximum process starting at a
        time t
    def phi_adjlog_from(self, u, x, t):
        if self.optvar is CALL: return self.process.phi_adjlogmax(u, x, t, self.T,
            self.strike)
        if self.optvar is PUT:  raise Exception("Lookback puts are not available")

    # funtion generating analytic solution
    def analytic(self, x, t, prevex):
        tau = self.T - t

        tc1 = lambda input : x * norm.cdf(self.d(input, x, t)) -
            np.exp(-self.mu*tau) * input * norm.cdf(self.d(input, x, t) -
            self.sigma*math.sqrt(tau))
        tc2 = lambda input : np.exp(-self.mu*tau)*(self.sigma**2/(2*self.mu)) * x
            * (-(x/input)**(-(2*self.mu)/(self.sigma**2)) * norm.cdf(self.d(input,
            x, t) - 2*self.mu*math.sqrt(tau)/self.sigma) +
            np.exp(self.mu*tau)*norm.cdf(self.d(input, x, t)))

        tp1 = lambda input : -x * norm.cdf(-self.d(input, x, t)) +
            np.exp(-self.mu*tau) * input * norm.cdf(self.sigma*math.sqrt(tau) -
            self.d(input, x, t))
        tp2 = lambda input : np.exp(-self.mu*tau)*(self.sigma**2/(2*self.mu)) * x
            * ((x/input)**(-(2*self.mu)/(self.sigma**2)) *
            norm.cdf(2*self.mu*math.sqrt(tau)/self.sigma - self.d(input, x, t)) -
            np.exp(self.mu*tau)*norm.cdf(-self.d(input, x, t)))


        if self.optvar == CALL and prevex <  self.strike: return tc1(self.strike)
            + tc2(self.strike)
        if self.optvar == CALL and prevex >= self.strike: return tc1(prevex) +
            tc2(prevex) + np.exp(-self.mu*tau)*(prevex-self.strike)

        if self.optvar == PUT  and prevex >= self.strike: return tp1(self.strike)
            + tp2(self.strike)
        if self.optvar == PUT  and prevex <  self.strike: return tp1(prevex) +
            tp2(prevex) + np.exp(-self.mu*tau)*(self.strike-prevex)

    # d_x coefficients used in analytic solution calculation
    def d(self, sub, x, t):
        tau = self.T - t
        return (np.log(x/sub) + self.mu*tau + 0.5*self.sigma**2*tau) /
            (self.sigma*math.sqrt(tau))


# child class describing european option contracts
class European(Option):
    # function determining payoff for a given path
    def payoff(self, path, prevex, adjlogs=False):
        s = path[-1]
        if adjlogs: return np.log(s/self.strike)


        if self.optvar == CALL: return np.maximum(s - self.strike, 0)
        if self.optvar == PUT:  return np.maximum(self.strike - s, 0)

    # determining payoff coefficients
    def H(self, k):
        if self.optvar is CALL: return (self.chi(k, 0, self.b) - self.psi(k, 0,
            self.b)) * (2*self.strike)/(self.b-self.a)
        if self.optvar is PUT:  return (self.psi(k, self.a, 0) - self.chi(k,
            self.a, 0)) * (2*self.strike)/(self.b-self.a)

    # funtion returning characteristic function for maximum process starting at a
        time t
    def phi_adjlog_from(self, u, x, t):
        return self.process.phi_adjlog(u, x, t, self.T, self.strike)

    # funtion generating analytic solution
    def analytic(self, x, t):
        tau = self.T - t
        d1 = (np.log(x/self.strike) + (self.mu+self.sigma**2/2)*tau) /
            (self.sigma*math.sqrt(tau))
        d2 = d1 - self.sigma*math.sqrt(tau)

        if self.optvar is CALL: return x * norm.cdf(d1) - self.strike *
            np.exp(-self.mu*tau) * norm.cdf(d2)
        if self.optvar is PUT:  return self.strike * np.exp(-self.mu*tau) *
            norm.cdf(-d2) - x * norm.cdf(-d1)
```

B Implementation of COS Method with Spitzer Recursion

```python
import numpy as np
import matplotlib.pyplot as plt
from classes import *
from colorama import Fore, Style
np.set_printoptions(threshold=np.inf)

# function returning a cos method pdf estimate for given parameters
def COS_Estimate(option, value, time, N, previousextreme=None, filter=None):
    tau = option.T - time
    ks = np.linspace(0, N-1, N)  # defining an array with all values of k
    us = ks * np.pi / (option.b - option.a)  # defining an array of the factor
        k*pi/(b-a)

    # defining the array of F_k values
    Fks = Filter.filterterm(filter, ks, N) * np.real(option.phi_adjlog_from(us,
        value, time) * np.exp(-i * us * option.a))
    Fks[0] *= 0.5

    # defining the array of H_k values according to the option specific function
    Hs = np.array([option.H(k, previousextreme) for k in ks])

    # multiplying the Fk values by the Hk values
    vs = np.array([F * H for F, H in zip(Fks, Hs)])

    return np.exp(-option.process.mu * tau) * np.sum(vs)

# function returning the cos density estimate (vectorised)
def COS_Density(option, value, time, N, y, filter):
    ks = np.linspace(0, N-1, N)  # defining an array with all values of k
    us = ks * np.pi / (option.b - option.a)  # defining an array of the factor
        k*pi/(b-a)

    # defining the array of F_k values
    Fks = (2.0 / (option.b - option.a)) * Filter.filterterm(filter, ks, N) *
        np.real(option.phi_adjlog_from(us, value, time) * np.exp(-i * us *
        option.a))
    Fks[0] *= 0.5

    # defining array of cos(u*(y-a))
    cs = np.cos(np.outer(us, y - option.a))

    # inproduct of F_ks and cs => desired function
    f_X = np.matmul(Fks, cs)

    return f_X

# function plotting the output of COS_Density() + monte carlo for comparison
def COS_Plot(option : Option, value, time, ns, filter):
    y = np.linspace(-.5, .5, 1000)

    # plotting the resulting density function for multiple values of N
    for n in ns:
        f_X = COS_Density(option, value, time, n, y, filter)
        # print(max(f_X))
        plt.plot(y, f_X, label=f'N={n}')
        # plt.vlines(np.log(value/option.strike), -.5, 4, colors='k', linestyles='dotted')

    values = option.MonteCarlo(10000, 1000, time, value, prevex=value, adjlogs=True)
    plt.hist(values, density=True, bins=40)
    plt.grid()
    # plt.legend()
    plt.ylim(-1, 13)
    plt.savefig('./option_pricing_extended/fig.png')

    plt.figure()
    plt.grid()
    xs = np.linspace(-1, 1, 100)
    plt.plot(xs, [Filter.filterterm(filter, x, 1) for x in xs])
    plt.savefig('./option_pricing_extended/filter.png')

# Test function for lookback options
def LB(t, St, K, prevex, variant, T, mu, sigma, filter):
    gbm = GBM(mu, sigma)
    lower = -10*np.sqrt(T)
    upper =  10*np.sqrt(T)

    # defining the option parameters
    lb = Lookback(stockmechanic=gbm,
                  exercisetime=T,
                  strikeprice=K,
                  optionvariant=variant,
                  lowerintegration=lower,
                  upperintegration=upper)

    # generating analytical solution
    analytic = lb.analytic(St, t, prevex)
    print(f'analytic:    {analytic}')

    # generating cos method value estimate for given N_cos values
    ns = [1024]
    for n in ns:
        est = COS_Estimate(lb, St, t, n, prevex, filter)
        print(f'for N={n}, the estimate is {est}')

    COS_Plot(lb, St, t, ns, filter)

# function for comparing numerical and analytic a_k values
def ak_NUMvsANA(k, alpha, stepsize, T, lowerintegration, upperintegration, h):
    xmin = -10*math.sqrt(T)
    xmax =  10*math.sqrt(T)
    zs = np.linspace(xmin, xmax, 1000)

    gbm = GBM(0.1, 0.3)
    ana_a = lambda z : gbm.ana_a(k, z, 0, stepsize)
    num_a = lambda z : gbm.num_a(k, z, alpha, stepsize, lowerintegration, upperintegration, h)

    plt.figure()
    plt.grid()
    plt.plot(zs, [ana_a(z) for z in zs], label="analytic")
    plt.plot(zs, [num_a(z) for z in zs], label="numerical")
    plt.legend()

    plt.savefig('./option_pricing_extended/fig.png')

# LB(t=0.5, St=100, prevex=110, K=120, variant=CALL, T=1, mu=0.05, sigma=0.3,
    filter=COS)

ak_NUMvsANA(2000, 5, 0.001, 2, -10, 10, 1/10)

winsound.Beep(1000, 300)
winsound.Beep(1800, 1000)
```

```
import numpy as np
import math
import random
from enum import Enum
import matplotlib.pyplot as plt
from matplotlib import collections  as mc
from scipy.special import erfc
from scipy.stats import norm
import sys
from colorama import Fore, Style
import time
import winsound
np.seterr(all='raise')

# defining imaginary unit
i = complex(0.0, 1.0)

# custom class for option variants
class OptionVariant(Enum):
    PUT = 'PUT'
    CALL = 'CALL'
PUT = OptionVariant.PUT
CALL = OptionVariant.CALL

# class defining filters and generating filter terms
class Filter(Enum):
    FEJER        = 'FEJER'
    RAISEDCOSINE = 'RAISEDCOSINE'
    EXPONENTIAL  = 'EXPONENTIAL'

    def filterterm(var, n, N):
        match var:
            case Filter.FEJER:
                out = 1 - abs(n/N)
            case Filter.RAISEDCOSINE:
                out = 0.5*(1 + np.cos(math.pi * n/N))
            case Filter.EXPONENTIAL:
                alpha = - np.log(2.22*10**(-16))
                p=4
                return np.exp(-alpha*(n/N)**p)
            case _:
                return 1
        return out
FEJ = Filter.FEJER
COS = Filter.RAISEDCOSINE
EXP = Filter.EXPONENTIAL
```

```python

# sum prime operator from grzelak and oosterlee
def sump(array):
    if array.size == 0:
        return 0
    res = np.sum(array) - array[0]/2
    return res

# charactersitic function standard normal distribution
def phi_N(u):
    try:
        return np.exp(-0.5 * u**2)
    except:
        return 0

# parent class defining stochastic processes
class Process:
    # function plotting the integrand in numerical a_k calculation
    def plotintegrand(self, k, z, alpha, stepsize, lowerintegration,
        upperintegration, h):
        i = complex(0, 1)
        tau_k = k * stepsize
        xs = np.linspace(lowerintegration, upperintegration, 1000) + i * alpha
        f = lambda xi : self.phi_levy(tau_k, -xi)*z / (xi**2 + xi*z)

        plt.figure()
        plt.plot(xs, f(xs))
        xs = np.arange(lowerintegration, upperintegration + h, h) + i*alpha
        for i, x in enumerate(xs[:-1]):
            plt.plot([xs[i], xs[i+1]], [f(xs[i]), f(xs[i+1])], 'ko',
                linestyle='solid', linewidth=.7, markersize=2)


        plt.savefig('./option_pricing_extended/integrand.png')

    # function returning adjusted logged spitzer recurrence formula
    def spitzer_result(self, m : int, u : np.array, x, t, T, K, num=False):
        return np.exp(i*u*np.log(x/K)) * self.spitzer_recurrence(m, u, t, T,
            num=False)
        # return np.exp(i*u*np.log(x/K)) * (self.spitzer_recurrence(m, u, t, T,
            num=False) + self.spitzer_recurrence(m, -u, t, T, num=False))

    # spitzer recurrence algorithm from haslip and kaishev
    def spitzer_recurrence(self, m : int, u : np.array, t, T, num=False):
        p = np.zeros((m, m, u.size), dtype=complex)
        for I in range(m):
            p[0,I] = self.ana_a(I, u, t, (T-t)/m) / (I+1)

        out = p[0, m-1]

        for I in range(1, m):
            for J in range(m - I):
                for k in range(J+1):
                    try:
                        p[I, J] += p[I-1, k] * p[0, J-k]
                    except Exception as e:
                        print(p[I-1, k] * p[0, J-k])
                        raise e
            try:
                out += p[I, m-1-I] / math.factorial(I+1)
            except Exception as e:
                # print('spitzer recurrence error caught: ' + str(e))
                pass

        return out

    # function calculating a_k coefficient via trapezoid rule
    def num_a(self, k, z, alpha, stepsize, lowerintegration, upperintegration, h):
        tau_k = k * stepsize

        xs = np.arange(lowerintegration, upperintegration + h, h) + i*alpha
        f = lambda xi : self.phi_levy(tau_k, -xi)*z / (xi**2 + xi*z)

        sum = 0
        for k, x in enumerate(xs[:-1]):
            sum += 0.5 * (f(xs[k]) + f(xs[k+1])) * h

        return 1 - (i / (2*math.pi)) * sum

# child class defining geometric brownian motion
class GBM(Process):
    def __init__(self, mu, sigma):
        self.mu    = mu
        self.sigma = sigma

    # defining the charactersitic function of gbm exponent
    def phi_levy(self, tau, z):
        return np.exp(i*z*self.mu*tau - 0.5*self.sigma**2*z**2*tau)

    # analytical a_k coefficients
    def ana_a(self, k, z, t, stepsize):
        try:
            tau_k = k * stepsize
            alpha = self.mu - self.sigma**2/2

            t1 = norm.cdf(-alpha*np.sqrt(tau_k)/(self.sigma))
            f1 = 0.5*np.exp(-0.5*z**2*self.sigma**2*tau_k + 1j*tau_k*z*alpha)
            f2 = erfc(-np.sqrt(tau_k/2) * (z*self.sigma*1j + alpha/self.sigma))
            return t1 + f1 * f2
        except FloatingPointError as e:
            # print('ana_a error caught: ' + str(e))
            return 0

# parent class defining options
class Option:
    def __init__(self, stockmechanic, exercisetime, strikeprice, optionvariant,
         lowerintegration, upperintegration):
        self.process = stockmechanic
        self.mu      = stockmechanic.mu
        self.sigma   = stockmechanic.sigma
        self.T       = exercisetime
        self.strike  = strikeprice
        self.optvar  = optionvariant
        self.a       = lowerintegration
        self.b       = upperintegration

    # function used in H_k calculation
    def chi(self, k, c, d):
        return 1/(1 + (k * np.pi / (self.b-self.a))**2) * (
            np.cos(k*np.pi*(d-self.a)/(self.b-self.a)) * np.exp(d)
          - np.cos(k*np.pi*(c-self.a)/(self.b-self.a)) * np.exp(c)
          + np.sin(k*np.pi*(d-self.a)/(self.b-self.a)) * np.exp(d) * (k*np.pi) /
            (self.b-self.a)
          - np.sin(k*np.pi*(c-self.a)/(self.b-self.a)) * np.exp(c) * (k*np.pi) /
            (self.b-self.a))

    # function used in H_k calculation
    def psi(self, k, c, d):
        if k == 0:
            return (d-c)
        else:
            return (np.sin(k*np.pi*(d-self.a)/(self.b-self.a)) -
                np.sin(k*np.pi*(c-self.a)/(self.b-self.a))) * (self.b-self.a) /
                (k*np.pi)

    # function giving monte carlo estimate of option value (showing a progress
        percentage furing calculation)
    def MonteCarlo(self, iterations, steps, t, st, prevex=None, adjlogs = False):
        vs = np.zeros(iterations)

        percentage = 0.0
        print("{: >6.2f}".format(percentage) + '%', end='', flush=True)

        # monte carlo iterations loop
        for i in range(iterations):
            newperc = round(100*i/iterations, 2)
            if newperc != percentage:
                percentage = newperc
                sys.stdout.write('\b' * 7 + ' ' * 7 + '\b' * 7)
                print(f'{Fore.MAGENTA + Style.BRIGHT}{"{:
                    >6.2f}".format(newperc)}%{Style.RESET_ALL}', end='',
                    flush=True)

            # generating sample path and determining payoff
            gbm = self.samplepath(self.T - t, steps, st)
            vs[i] = self.payoff(gbm, prevex, adjlogs)
        sys.stdout.write('\b\b\b\b\b\b\b\b')


        # return all value endpoints for monte carlo graphing
        if adjlogs: return vs

        # returning monte carlo estimate
        else:       return np.exp(-self.mu*(self.T-t)) * np.average(vs)

    # function generating a gbm sample path
    def samplepath(self, tau, steps, st):
        srw = (1 / np.sqrt(steps)) * np.cumsum([0] + [random.choice([-1, 1]) for _
            in range(int(steps * tau) - 1)])
        ts  = np.arange(int(steps * tau)) / steps

        return st * np.exp((self.mu - np.power(self.sigma, 2)/2) * ts + self.sigma
            * srw)

# child class describing lookback option contracts
class Lookback(Option):
    # function determining payoff for a given path
```

```python
    def payoff(self, path, prevex, adjlogs=False):
        mx = np.max(np.append(path, prevex))
        mn = np.min(np.append(path, prevex))

        if adjlogs:
            if self.optvar == CALL: return np.log(mx/self.strike)
            if self.optvar == PUT:  return np.log(mn/self.strike)

        if self.optvar == CALL: return np.maximum(mx - self.strike, 0)
        if self.optvar == PUT:  return np.maximum(self.strike - mn, 0)

    # determining payoff coefficients
    def H(self, k, prevex):
        adjprev = np.log(prevex/self.strike)

        tc1 = tc2 = tc3 = 0
        tp1 = tp2 = tp3 = 0

        if prevex > self.strike:
            tc1 = self.psi(k, self.a, adjprev) * 2*(prevex-self.strike)/(self.b-self.a)
            tc2 = (self.chi(k, adjprev, self.b) - self.psi(k, adjprev, self.b)) * 2*self.strike/(self.b-self.a)
            tp3 = (self.psi(k, self.a, 0) - self.chi(k, self.a, 0)) * 2*self.strike/(self.b-self.a)
        else:
            tc3 = (self.chi(k, 0, self.b) - self.psi(k, 0, self.b)) * 2*self.strike/(self.b-self.a)
            tp1 = self.psi(k, adjprev, self.b) * 2*(self.strike-prevex)/(self.b-self.a)
            tp2 = (self.psi(k, self.a, adjprev) - self.chi(k, self.a, adjprev)) * 2*self.strike/(self.b-self.a)

        if self.optvar is CALL: return tc1 + tc2 + tc3
        if self.optvar is PUT:  return tp1 + tp2 + tp3

    # funtion returning characteristic function for maximum process starting at a time t
    def phi_adjlog_from(self, u, x, t):
        m=20
        out = self.process.spitzer_result(m=m, u=u, x=x, t=t, T=self.T, K=self.strike, num=False)
        return out

    # funtion generating analytic solution
    def analytic(self, x, t, prevex):
        tau = self.T - t

        tc1 = lambda input : x * norm.cdf(self.d(input, x, t)) - np.exp(-self.mu*tau) * input * norm.cdf(self.d(input, x, t) - self.sigma*math.sqrt(tau))
        tc2 = lambda input : np.exp(-self.mu*tau)*(self.sigma**2/(2*self.mu)) * x * (-(x/input)**(-(2*self.mu)/(self.sigma**2)) * norm.cdf(self.d(input, x, t) - 2*self.mu*math.sqrt(tau)/self.sigma) + np.exp(self.mu*tau)*norm.cdf(self.d(input, x, t)))

        tp1 = lambda input : -x * norm.cdf(-self.d(input, x, t)) + np.exp(-self.mu*tau) * input * norm.cdf(self.sigma*math.sqrt(tau) - self.d(input, x, t))
        tp2 = lambda input : np.exp(-self.mu*tau)*(self.sigma**2/(2*self.mu)) * x
            * ((x/input)**(-(2*self.mu)/(self.sigma**2)) *
            norm.cdf(2*self.mu*math.sqrt(tau)/self.sigma - self.d(input, x, t)) -
            np.exp(self.mu*tau)*norm.cdf(-self.d(input, x, t)))


        if self.optvar == CALL and prevex <  self.strike: return tc1(self.strike)
            + tc2(self.strike)
        if self.optvar == CALL and prevex >= self.strike: return tc1(prevex) +
            tc2(prevex) + np.exp(-self.mu*tau)*(prevex-self.strike)

        if self.optvar == PUT  and prevex >= self.strike: return tp1(self.strike)
            + tp2(self.strike)
        if self.optvar == PUT  and prevex <  self.strike: return tp1(prevex) +
            tp2(prevex) + np.exp(-self.mu*tau)*(self.strike-prevex)

    # d_x coefficients used in analytic solution calculation
    def d(self, sub, x, t):
        tau = self.T - t
        return (np.log(x/sub) + self.mu*tau + 0.5*self.sigma**2*tau) /
            (self.sigma*math.sqrt(tau))
```

C Limit Calculation Spitzer Algorithm

```
x = 5.899
d = 0.309

for i in range(200):
    xn = x - d/1.69
    d = x - xn
    x = xn

print(x)

# 5.45117391304336
```