Stochastic Models in Finance

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

DEPARTMENT OF STATISTICS

of

RHODES UNIVERSITY

by

Hassan Mazengera

November 2017

In memory of my loving mother

Sabina Mazengera

Abstract

Stochastic models for pricing financial securities are developed. First, we consider the Black Scholes model, which is a classic example of a complete market model and finally focus on Lévy driven models. Jumps may render the market incomplete and are induced in a model by inclusion of a Poisson process. Lévy driven models are more realistic in modelling of asset price dynamics than the Black Scholes model. Martingales are central in pricing, especially of derivatives and we give them the desired attention in the context of pricing. There are an increasing number of important pricing models where analytical solutions are not available hence computational methods come in handy, see Broadie and Glasserman (1997). It is also important to note that computational methods are also applicable to models with analytical solutions. We computationally value selected stochastic financial models using C++. Computational methods are also used to value or price complex financial instruments such as path dependent derivatives. This pricing procedure is applied in the computational valuation of a stochastic (revenue based) loan contract. Derivatives with simple pay off functions and models with analytical solutions are considered for illustrative purposes. The Black-Scholes P.D.E is complex to solve analytically and finite difference methods are widely used. Explicit finite difference scheme is considered in this thesis for computational valuation of derivatives that are modelled by the Black-Scholes P.D.E. Stochastic modelling of asset prices is important for the valuation of derivatives: Gaussian, exponential and gamma variates are simulated for the valuation purposes.

Keywords: Risk neutral, Martingale, Black-Scholes, Simulation, C++ computation, Finite difference, Lévy processes

Contents

Acknowledgments

I thank the Almighty our God for taking me this far. Great appreciation goes to my supervisor, Professor Irek Szyszkowski for his time, expertise and encouragement. I am truly grateful to his contribution in broadening and deepening my understanding of stochastic models and also the exposition to C++, which has become an indispensable tool in derivative pricing. It was a great time being around such a supportive supervisor. I really appreciate the support from the department of Statistics at large; members of the staff, our administrator and fellow students. I am thankful to big brother, Mr Melody Manyeruke, from the Chemistry department for his great contribution to my life. I am greatly indebted to Ms Bonelwa Sidumo for her support. Finally I am also thankful to my family for their moral and financial support, it was not easy but they have been very supportive.

Chapter 1

Introduction

Most resources discuss the theory around stochastic models and derivatives valuation in particular and neglect the computational part. For that reason our work bridges the gap that exist by giving a theoretical exposition of the models and then practical implementation in C++. The work will help in enriching the stochastic models literature with computational implementation.

In this thesis we consider stochastic models initially in a complete market set up. In a complete market portfolio replication for any contingent claim is possible, Davis (2004). With complete market models every claim can be perfectely hedged by the underlying asset. The Black-Scholes is a complete market model. In asset and contingent claim pricing, martingale theory plays an important role. It is necessary to find an equivalent martingale measure when pricing financial instruments. Chapter 2 and 3 deal with the review of martingales, change of measure and the risk neutral pricing. Derivatives are priced in a risk neutral environment. This risk neutral pricing is achieved when there is a change of probability measure. A risk neutral probability measure makes a pricing process a martingale.

Jumps may render the market incomplete thereby making the applicability of the Black-Scholes model difficult, see Staum (2007). In Chapter 5, we focus on the Lévy driven models and the Poisson process is very important in inducing jumps in any model. A model that incorporates the Brownian motion component and the Poisson process is called a Lévy Jump-Diffusion model. These models have been applied in a wide range of asset price modelling, a good example is in the electricity price modelling, see Cartea and Figueroa (2005) and Benth et al. (2007) among others.

For derivatives with simple payoff, the analytical solution for the price is available but for more complex derivatives where an early exercise is possible, efficient and accurate numerical algorithms are required, see Kadalbajoo et al. (2015). In this thesis our focus is on the European option and hence there is no possibility of early exercise.

Empirical implementation of these stochastic models require some computational effort. To start with, we have to approximate the integrals, differentials and boundary conditions for the equations. To achieve this goal we consider finite difference schemes (methods). In Chapter 4 we review the finite difference methods that are available especially in the finance literature. From those methods, the Crank-Nicolson scheme is the one that is widely used. In Chapter 6, we present computational work. The goal of the presented sections of chapter 6 is computational valuation of financial models. We implement basic financial models using the Dev C++ software and then generate uniform variates using the *standard C library.* The uniform variates are then used in the generation of normal and exponential variates.

We further use the exponential variates in generating the gamma variates. The exponential process is important especially when working with both the Poisson and Gamma processes. After simulation of the random variates in C++, we then simulate asset price processes. We consider the crude Monte Carlo and the antithetic variate methods. For discussion of Monte Carlo methods, see Boyle (1977), Korn et al. (2010) and London (2005) among other resources. The last section of Chapter 6 is devoted to the application of finite difference method in pricing using C++. The explicit finite difference method is the only method that we implement since the Implicit difference and Crank-Nicolson involve solving a system of equations for the unknown parameters and this is usually achieved using matrices in C++. We assume that the functions that we consider are smooth and differentiable up to order two.

The thesis is arranged as follows; Chapter 1 gives a brief insight into the work, Chapters 2, 3, 4 and 5 covers literature review. Chapter 6 focuses on computational implementation of stochastic models in C++.

Chapter 2

Review of Selected Financial Models

This chapter provides conceptual foundation for the next and subsequent chapters. Definitions and other preliminary concepts such as martingales are introduced. Chapters 3 and 5 depends heavily on the theory developed in this chapter.

2.1 Introduction to Martingales; Definitions

In this section, we define important terms. Some terms such as filtration and martingales are defined in both discrete and continuous setting. A very brief introduction to an essential concept of the conditional expectation and to its basic properties is given in Appendix A.

Definition 1. (Filtration) If $X_1, X_2, ...$ is a sequence of random variables, then the associated (discrete time) filtration is the collection $\{\mathscr{F}_n\}_{n\in\mathbb{N}}$ where $\mathscr{F}_n$ denotes the information in $X_1, X_2, ...X_n$. Sometimes the filtration is given starting at time $n = 1$ and other times at $n = 0$. If it starts at time $n = 1$, we define $\mathscr{F}_0$ to be "no information".

Definition 2. (Martingale) Let $(\Omega, \mathscr{F}, P)$ be a probability space where Ω, is the sample space and P is a probability measure. Suppose $X_1, X_2, ...$ is a sequence of random variables to which we associate the filtration $\{\mathscr{F}_n\}$ where $\mathscr{F}_n$ denotes the information in $X_1, X_2, ...X_n$, for each n. A sequence of random variables $M_0, M_1, ...$ is called a martingale with respect to the filtration $\{\mathscr{F}_n\}_{n\in\mathbb{N}}$ iff

1. for each integer n, M_n is an $\mathscr{F}_n$ measurable random variable and $E[\mid M_n \mid] < \infty$;
2. for any integers n and m, if $m < n$ then $E[M_n \mid \mathscr{F}_m] = M_m$ (or $E[M_n - M_m \mid \mathscr{F}_m] = 0$).

If M_n is a martingale, then

$$E[M_n] = E[E[M_n \mid \mathscr{F}_0]] = E[M_0].$$

If we replace conditions $E[M_n \mid \mathscr{F}_m] = M_m$ by $E[M_n \mid \mathscr{F}_m] \geq M_m$, then sequence of random variables $M_0, M_1, ...$ is called a ***sub-martingale***. If we replace conditions $E[M_n \mid \mathscr{F}_m] = M_m$ by $E[M_n \mid \mathscr{F}_m] \leq M_m$, then sequence of random variables $M_0, M_1, ...$ is called a ***super-martingale***.

Definition 3. (Adapted) Given a probability space $(\Omega, \mathscr{F}, P)$ and a filtration $\{\mathscr{F}_n\}_{n\in\mathbb{N}}$, that is a sequence of $\sigma-$fields, $\mathscr{F}_0 \subseteq \mathscr{F}_1 \subseteq ....$ A sequence of random variables X_n, $n \in \mathbb{N}$, is called adapted if for every integer n, X_n is $\mathscr{F}_n$ measurable.

Definition 4. (Square Integrable Martingale) A martingale $\{M_n\}_{n\in\mathbb{N}}$ is called square integrable martingale if additionally $E[M_n^2] < \infty$, for each n.

Definition 5. (Stationary Increments) A continuous time process $\{X(t),\, t \geq 0\}$ is said to have stationary increments if for any $0 \leq s \leq t$, the probability distribution functions of $X(t) - X(s)$ and $X(t-s) - X(0)$ are identical.

Definition 6. (Continuous version-Filtration) A collection $\{\mathscr{F}_t : t \in [0,\infty)\}$ of $\sigma-$algebras of sets in $\mathscr{F}$ is called a filtration if $\mathscr{F}_t \subseteq \mathscr{F}_{t+s}$ for any $t, s \in [0,\infty)$.

Definition 7. (Continuous version-Martingale) A stochastic process M_t is called a martingale with respect to measure P and filtration $\{\mathscr{F}_n\}_{n\in\mathbb{N}}$ if and only if

1. $E_P(|M_t|) < \infty$, $\forall t$;
2. $E_P(M_t|\mathscr{F}_s) = M_s$, $\forall s \leq t$.

Definition 8. (Usual Conditions) A filtration $(\mathscr{F}_t)_{t\geq 0}$ or (the filtered probability space $\left(\Omega, \mathscr{F}, (\mathscr{F}_t)_{t\geq 0}, P\right)$) is said to satisfy the usual conditions if it is right continuous and complete. $\mathscr{F}_t$ is right continuous if for each $t \geq 0$, $\mathscr{F}_t \equiv \bigcap_{\varepsilon>0} \mathscr{F}_{t+\varepsilon}$. We say that $(\mathscr{F}_t)_{t\geq 0}$ is complete if $(\Omega, \mathscr{F}, P)$ is complete (ie. all subsets of the P-null sets are measurable) and $\{A \in \mathscr{F} : P(A) = 0\} \subset \mathscr{F}_0$.

Definition 9. (Brownian Motion) A stochastic process $\{W_t,\, t \geq 0\}$ defined on a filtered probability space $(\Omega, \mathscr{F}, P)$ is said to be a Brownian motion process with parameter, σ if:

- $\{W_t,\, t \geq 0\}$ has stationary and independent increments;
- for every $t > 0$, W_t is normally distributed with mean zero and variance $\sigma^2 t$;
- $\{W_t,\, t \geq 0\}$ has continuous sample paths and $W_0 = 0$.

The Brownian motion process is sometimes called the Wiener process, see Chapter 10, Ross (2014) and Chapter 11, Kijima (2013).

Definition 10. The process $W^* = (W_t^* : t \geq 0)$ is a standard Brownian motion when $\sigma = 1$ in the above definition. Any Brownian motion can be converted to a standard Brownian motion by letting:

$$W_t^* = \frac{W_t}{\sigma}.$$

2.2 Examples of Martingales

Example 1. The constant process $S_t = c$, for all t, is a martingale with respect to any measure:

$$E[S_t \mid \mathscr{F}_s] = E[c \mid \mathscr{F}_s] = E[c] = c,$$

this holds for all $s \leq t$, since c is independent of $\mathscr{F}_t$.

Example 2. A P-Brownian motion $\{W_t,\, t \geq 0\}$ is always a P-martingale with respect to its natural filtration $(\mathscr{F}_t = \sigma(W_s, 0 \leq s \leq t))_{t\geq 0}$. Note that, for every $0 \leq s \leq t$, $W_t - W_s$ is independent of $\mathscr{F}_s$ (W_s is $\mathscr{F}_s$ measurable), and $E(W_t - W_s) = E(W_{t-s} - W_0) = E(W_{t-s}) = 0$ (because of stationary increments). Hence, due to $W_t - W_s$ also having expected value of zero, we obtain

$$E\left[W_t \mid \mathscr{F}_s\right] = E\left[W_s \mid \mathscr{F}_s\right] + E\left[W_t - W_s \mid \mathscr{F}_s\right] = W_s + E\left[W_t - W_s\right] = W_s.$$

2.3 Examples of Sub-martingales

Example 3. Assuming $\{(W_t, \mathscr{F}_t),\, t \geq 0\}$ is a Brownian motion, the process $X_t = \mu t + \sigma W_t$ with $\mu > 0$ is a sub-martingale. The integrability follows from the inequality $|X_t(\omega)| \leq \mu t + \sigma\,|W_t(\omega)|$ and the integrability of W_t. As X_t is adapted to $\mathscr{F}_t$, we obtain

$$\begin{aligned} E\left[X_{t+s} \mid \mathscr{F}_t\right] &= E\left[\mu(t+s) + \sigma W_{t+s} \mid \mathscr{F}_t\right] \\ &= \mu s + E\left[\mu t + \sigma W_{t+s} \mid \mathscr{F}_t\right]. \end{aligned}$$

But

$$\mu s + E\left[\mu t + \sigma W_{t+s} \mid \mathscr{F}_t\right] \;>\; E\left[\mu t + \sigma W_{t+s} \mid \mathscr{F}_t\right].$$

Finding the above expectation, in the square bracket;

$$\begin{aligned} E\left[\mu t + \sigma W_{t+s} \mid \mathscr{F}_t\right] &= \mu t + \sigma E\left[W_{t+s} \mid \mathscr{F}_t\right] \\ &= \mu t + \sigma W_t \\ &= X_t, \end{aligned}$$

where $E\left[W_{t+s} \mid \mathscr{F}_t\right] = W_t$, because $\{(W_t, \mathscr{F}_t),\, t \geq 0\}$ is also a martingale (refer to Example 2 above).

Example 4. We show here that the square of a Brownian motion, $(W_t^2)_{t\geq 0}$, is a sub-martingale. We first show that $(W_t^2 - t)_{t\geq 0}$ is a martingale (i.e. $E\left[W_t^2 - t \mid \mathscr{F}_s\right] = W_s^2 - s$ for $t \geq s$).

$$\begin{aligned} E\left[W_t^2 - t \mid \mathscr{F}_s\right] &= E\left[((W_t - W_s) + W_s)^2 - t \mid \mathscr{F}_s\right] \\ &= E\left[\left((W_t - W_s)^2 + 2W_s(W_t - W_s) + W_s^2\right) - t \mid \mathscr{F}_s\right] \\ &= E\left[(W_t - W_s)^2 \mid \mathscr{F}_s\right] + 2E\left[W_s(W_t - W_s) \mid \mathscr{F}_s\right] + E\left[W_s^2 - t \mid \mathscr{F}_s\right] \\ &= E\left[(W_t - W_s)^2 \mid \mathscr{F}_s\right] + 2W_s E\left[W_t - W_s \mid \mathscr{F}_s\right] + E\left[W_s^2 - s - (t-s) \mid \mathscr{F}_s\right] \\ &= E\left[(W_t - W_s)^2\right] + 2W_s E\left[W_t - W_s\right] + E\left[W_s^2 - s - (t-s) \mid \mathscr{F}_s\right] \\ &= E\left[(W_t - W_s)^2\right] + 2W_s E\left[W_t - W_s\right] + E\left[W_s^2 - s \mid \mathscr{F}_s\right] - (t-s) \\ &= E\left[(W_t - W_s)^2\right] + 0 + W_s^2 - s - (t-s) \\ &= t - s + W_s^2 - t.\ \textit{Hence} \\ E\left[W_t^2 - t \mid \mathscr{F}_s\right] &= W_s^2 - s \quad \forall t \geq s. \end{aligned}$$

Using this crucial result, we finally get

$$E\left[W_{t+s}^2 \mid \mathscr{F}_t\right] = E\left[W_{t+s}^2 - (t+s) \mid \mathscr{F}_t\right] + (t+s) = W_t^2 - t + (t+s) = W_t^2 + s \geq W_t^2.$$

2.4 Continuous Martingales

Definition 11. (Predictable Process) The predictable $\sigma-$algebra is the $\sigma-$algebra P generated on $[0,T]\times\Omega$ by all adapted left continuous processes. A mapping $X:[0,T]\times\Omega\to\mathbb{R}$ which is measurable with respect to P is called a predictable process. Any left continuous process is therefore predictable. However there are predictable processes which are not left continuous, see Tankov (2003).

Definition 12. (Exponential Process) Let $u\in L^2[0,T]$ be a deterministic function. Then a stochastic process

$$M_t = e^{\int_0^t u(s)dW_s - \frac{1}{2}\int_0^t u^2(s)ds},\ t\geq 0,$$

is called an exponential process induced by u.

Example 5. If $v(s)$ is a continuous function on $[0,T]$, then $X_t = \int_0^t v(s)dW_s$ is an $\mathscr{F}_t$ martingale.

Proof.

$$\begin{aligned} E\left[X_t \mid \mathscr{F}_s\right] &= E\left[\int_0^s v(\zeta)dW_\zeta + \int_s^t v(\zeta)dW_\zeta \mid \mathscr{F}_s\right] \\ &= E\left[\int_0^s v(\zeta)dW_\zeta \mid \mathscr{F}_s\right] + E\left[\int_s^t v(\zeta)dW_\zeta \mid \mathscr{F}_s\right] \\ &= X_s + E\left[\int_s^t v(\zeta)dW_\zeta\right] = X_s, \end{aligned}$$

because $\int_s^t v(\zeta)dW_\zeta$ is independent of $\mathscr{F}_s$ which implies that;

$$E\left[\int_s^t v(\zeta)dW_\zeta \mid \mathscr{F}_s\right] = E\left[\int_s^t v(\zeta)dW_\zeta\right] = 0.$$

As a special case, we observe that $M_t = e^{\sigma W_t - \frac{1}{2}\sigma^2 t}$ is an $\mathscr{F}_t$ martingale, for any positive constant σ.

Example 6. Let $u:[0,T]\to\mathbb{R}$ be a continuous function, then

$$M_t = e^{\int_0^t u(s)dW_s - \frac{1}{2}\int_0^t u^2(s)ds}$$

is an $\mathscr{F}_t$ martingale for $0\leq t\leq T$.

Proof. Let us consider the process $U_t = \int_0^t u(s)dW_s - \frac{1}{2}\int_0^t u^2(s)ds$. Note that $M_t = e^{U_t}$ and

$$dU_t = u(t)dW_t - \frac{1}{2}u^2(t)dt.$$

Using basic properties of Itô's differentials, we obtain$(dU_t)^2 = u^2(t)dt$, see Appendix B. Moreover, customising the Itô's Lemma, $df(U_t) = f'(U_t)dU_t + \frac{1}{2}f''(U_t)(dU_t)^2$, to the case of

$f(x) = \exp(x)$, we get;

$$\begin{aligned} dM_t &= M_t dU_t + \frac{1}{2} M_t (dU_t)^2 \\ &= e^{U_t} dU_t + \frac{1}{2} M_t (dU_t)^2 \\ &= e^{U_t} \left(u(t) dW_t - \frac{1}{2} u^2(t) dt \right) + \frac{1}{2} e^{U_t} (dU_t)^2 \\ &= e^{U_t} \left(u(t) dW_t - \frac{1}{2} u^2(t) dt \right) + \frac{1}{2} e^{U_t} u^2(t) dt \\ &= e^{U_t} (u(t) dW_t) \\ dM_t &= u(t) M_t dW_t. \end{aligned}$$

Now, integrating between s and t yields

$$M_t = M_s + \int_s^t M_\zeta u(\zeta) dW_\zeta$$

Note that $\int_s^t M_\zeta u(\zeta) dW_\zeta$ is independent of $\mathscr{F}_s$ and
$E\left[\int_s^t M_\zeta u(\zeta) dW_\zeta \mid \mathscr{F}_s\right] = E\left[\int_s^t M_\zeta u(\zeta) dW_\zeta\right] = 0$. Hence

$$E[M_t \mid \mathscr{F}_s] = E\left[M_s + \int_s^t M_\zeta u(\zeta) dW_\zeta \mid \mathscr{F}_s\right] = M_s,$$

which completes the proof.

2.5 Radon-Nikodym Theorem

Definition 13. (Absolute Continuity) Let P and $\tilde{P}$ be two probability measures on space $(\Omega, \mathscr{F})$. Assume for every $A \in \mathscr{F}$ satisfying $P(A) = 0$, we also have $\tilde{P}(A) = 0$. Then we say that $\tilde{P}$ is absolutely continuous with respect to P.

Definition 14. (Equivalence) If for any event A in the sample space one has that

$$P(A) > 0 \iff \tilde{P}(A) > 0,$$

then two measures P and $\tilde{P}$ are said to be equivalent. In other words, if A is possible under P then it is also possible under $\tilde{P}$ and if it is impossible under P then it is also impossible under $\tilde{P}$.

Theorem 1. ***(Radon-Nikodym)*** *Given that $\tilde{P}$ is absolutely continuous with respect to P then there is a non negative random variable Z such that*

$$\tilde{P}(A) = \int_A Z dP \quad \forall A \in \mathscr{F}$$

and Z is called the Radon-Nikodym derivative of $\tilde{P}$ with respect to P. For the proof of the theorem, see Miermont (2017).

Remark. The equation in the above theorem implies a stronger condition given below

$$\tilde{E}[X] = E[XZ]$$

for every random variable X for which $E[XZ] < \infty$. Here $\tilde{E}$ is expectation with respect to measure $\tilde{P}$ and E is expectation with respect to measure P. Given that $\tilde{P}$ and P are equivalent probability measures and Z is the Radon-Nikodym derivative of $\tilde{P}$ with respect to P then $\frac{1}{Z}$ is the Radon-Nikodym derivative of P with respect to $\tilde{P}$, that is:

$$E[Y] = \tilde{E}\left[Y\frac{1}{Z}\right] \quad \forall Y.$$

2.5.1 Radon-Nikodym Martingales

Let Ω be the sample space, P and $\tilde{P}$, be two equivalent probability measures. Let Z be the Radon-Nikodym derivative of $\tilde{P}$ with respect to P i.e.

$$Z(\omega) = \frac{d\tilde{P}(\omega)}{dP(\omega)}.$$

Let us also define the process

$$Z_k = E[Z \mid \mathscr{F}_k] \quad k = 0, 1, ..., n,$$

where$\mathscr{F}_k$ is a filtration. Z_k is actually a P martingale, because

$$\begin{aligned} E[Z_{k+1} \mid \mathscr{F}_k] &= E[E[Z \mid \mathscr{F}_{k+1}] \mid \mathscr{F}_k] \\ &= E[Z \mid \mathscr{F}_k] \\ &= Z_k. \end{aligned}$$

Lemma 1. *If X is $\mathscr{F}_k$ measurable, then*

$$\tilde{E}[X] = E[XZ_k].$$

Proof. Note that

$$\begin{aligned} \tilde{E}[X] &= E[XZ] \\ &= E[E[XZ \mid \mathscr{F}_k]] \\ &= E[XE[Z \mid \mathscr{F}_k]] \\ &= E[XZ_k]. \end{aligned}$$

as required.

If X is $\mathscr{F}_k$ measurable, then for any $A \in \mathscr{F}_k$

$$\tilde{E}[I_A X] = E[Z_k I_A X]$$

or equivalently

$$\int_A X d\tilde{P} = \int_A X Z_k dP.$$

□

Lemma 2. *If X is $\mathscr{F}_k$ measurable and $0 \leq j \leq k$, then*

$$\tilde{E}\left[X \mid \mathscr{F}_j\right] = \frac{1}{Z_j} E\left[Z_k X \mid \mathscr{F}_j\right].$$

Proof. We first note that $\frac{1}{Z_j} E\left[Z_k X \mid \mathscr{F}_j\right]$ is $\mathscr{F}_j$ measurable. For any $A \in \mathscr{F}_j$, we have:

$$\begin{aligned}
\int_A \frac{1}{Z_j} E\left[Z_k X \mid \mathscr{F}_j\right] d\tilde{P} &= \int_A E\left[Z_k X \mid \mathscr{F}_j\right] dP \\
&= \int_A X Z_k dP \\
&= \int_A X d\tilde{P}.
\end{aligned}$$

Therefore, we have shown that

$$\tilde{E}\left[X \mid \mathscr{F}_j\right] \;=\; \frac{1}{Z_j} E\left[Z_k X \mid \mathscr{F}_j\right].$$

Please note that the continuous versions of the Lemmas presented in this Section can easily be obtained by using a continuous filtration $(\mathscr{F}_t)$. □

Chapter 3

Black-Scholes Model

This chapter is dedicated to the Black Scholes model. We start by introducing Girsanov's Theorem and change of measure. Girsanov's Theorem is fundamental when we need to change the drift of a stochastic differential equation. Removing drift in the price process makes it a martingale. Risk neutral valuation is covered as the next aspect and the importance of a martingale measure is highlighted in this section. We end the chapter by looking at valuation of derivatives in a risk neutral environment.

3.1 Girsanov's Theorem and Change of measure

Let $(\Omega, \mathscr{F}, P)$ be a filtered probability space when dealing with $\mathscr{F}_t$ martingale on the mentioned probability space. The filtration $\mathscr{F}_t$ is considered to be the sigma-algebra generated by the given Brownian motion W_t, that is $\mathscr{F}_t = \sigma\{W_u : u \leq t\}$.

Theorem 2. ***(Cameron-Martin-Girsanov)*** *If $(W_t)_{0\leq t\leq T}$ is a P Brownian motion and $(u(t))_{0\leq t\leq T}$ (drift) is an $(\mathscr{F}_t)_{0\leq t\leq T}$ predictable process satisfying the boundedness condition $(E_P\left[exp\left(\int_0^T u^2(t)dt\right)\right] < \infty)$, then there exist a measure Q such that;*

1. Q is equivalent to measure P;
2. $$\frac{dQ}{dP} = e^{-\int_0^T u(s)dW_s - \frac{1}{2}\int_0^T u^2(s)ds};$$
3. $\tilde{W}_t$ is a Q Brownian motion where
$$\tilde{W}_t = W_t + \int_0^t u(s)ds,\ 0 \leq t \leq T.$$

(see Baxter and Rennie (1996))

Theorem 3. ***(Converse to the Cameron-Martin-Girsanov Theorem)*** *If $(W_t)_{0\leq t\leq T}$ is a P Brownian motion and Q is a probability measure equivalent to P, then there exists an $(\mathscr{F}_t)_{0\leq t\leq T}$ predictable process $u(t)$ such that*

$$\tilde{W}_t = W_t + \int_0^t u(s)ds,$$

is a Q Brownian motion. That is W_t plus $\int_0^t u(s)ds$ is a Q Brownian motion and the Radon-Nikodym derivative of Q with respect to P (at time t) is $e^{-\int_0^T u(s)dW_s - \frac{1}{2}\int_0^T u^2(s)ds}$, see Baxter and Rennie (1996).

3.1.1 Cameron-Martin-Girsanov and Stochastic Differentials

Cameron-Martin-Girsanov (C-M-G) theorem applies to Brownian motions and is a powerful tool for controlling the drift of any process. Let's consider a stochastic process X_t with increment;

$$dX_t = \mu(t)dt + \sigma(t)dW_t, \tag{3.1.1}$$

where; $\mu(t)$ is the drift term, $\sigma(t)$ is volatility and W_t is a P Brownian motion. Suppose we want to find if there is a measure Q such that the drift process under Q is $v(t)dt$ instead of $\mu(t)dt$. The stochastic differential equation (3.1.1) can be rewritten as

$$dX_t = \sigma(t)\left(dW_t + \frac{\mu(t)-v(t)}{\sigma(t)}dt\right) + v(t)dt.$$

If we set $u(t) = \frac{\mu(t)-v(t)}{\sigma(t)}$ and $u(t)$ satisfies growth condition ($E_P\left[exp\left(\frac{1}{2}\int_0^T u^2(t)dt\right)\right] < \infty$), then there is a new measure Q such that

$$\tilde{W}_t = W_t + \int_0^t \left(\frac{\mu(s)-v(s)}{\sigma(s)}\right) ds$$

is a Q Brownian motion. That is the process $(\tilde{W}_t)$ defined by

$$d\tilde{W}_t = \left(\frac{\mu(t)-v(t)}{\sigma(t)}\right) dt + dW_t,$$

is a Brownian motion under Q. The differential of X_t under Q is given by

$$dX_t = v(t)dt + \sigma(t)d\tilde{W}_t.$$

3.2 Risk Neutral Valuation

An $(\mathscr{F}_t)$ predictable stochastic process X_t on a probability space $(\Omega, \mathscr{F}, P)$ is not always a martingale. It might become a martingale with respect to another probability measure Q on $\mathscr{F}$. This is called a martingale measure and this play an important role in mathematical explanation of the risk neutral valuation.

If the stock prices (S_t) are assumed to follow the Black-Scholes model, i.e. satisfy the stochastic differential equation

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

then one can easily check that $S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}$ solves it. Here μ and $\sigma > 0$ are assumed to be constants, and (W_t) is a Brownian motion.

Proposition 1. *Let us consider a Black-Scholes stock price model (S_t) with μ being the rate of return of the stock S_t. Then*

$$E\left[e^{-\mu t}S_t \mid \mathscr{F}_u\right] = e^{-\mu u}S_u,$$

for all $u < t$, that is $(e^{-\mu t}S_t)$ is an $\mathscr{F}_t$ martingale.

Proof. The process $e^{-\mu t}S_t$ is non-explosive since $E\left[\mid e^{-\mu t}S_t \mid\right] = e^{-\mu t}E\left[S_t\right]{=}e^{-\mu t}S_0e^{\mu t}{=}S_0 < \infty$. Moreover S_t is $\mathscr{F}_t$ is predictable, so is $e^{-\mu t}S_t$. Now

$$\begin{aligned} E\left[S_t \mid \mathscr{F}_u\right] &= E\left[S_0e^{(\mu-\frac{1}{2}\sigma^2)t+\sigma W_t} \mid \mathscr{F}_u\right] \\ &= E\left[S_0e^{(\mu-\frac{1}{2}\sigma^2)u+\sigma W_u}e^{(\mu-\frac{1}{2}\sigma^2)(t-u)+\sigma(W_t-W_u)} \mid \mathscr{F}_u\right]. \end{aligned}$$

Here we are making use of the independent and stationary increments of a Brownian motion process, W_t;

$$\begin{aligned} E\left[S_t \mid \mathscr{F}_u\right] &= S_0e^{(\mu-\frac{1}{2}\sigma^2)u+\sigma W_u}E\left[e^{(\mu-\frac{1}{2}\sigma^2)(t-u)+\sigma(W_t-W_u)} \mid \mathscr{F}_u\right] \\ &= S_0e^{(\mu-\frac{1}{2}\sigma^2)u+\sigma W_u}\left[e^{(\mu-\frac{1}{2}\sigma^2)(t-u)}E\left(e^{\sigma(W_t-W_u)} \mid \mathscr{F}_u\right)\right] \end{aligned} \tag{3.2.1}$$

since $S_0e^{(\mu-\frac{1}{2}\sigma^2)u+\sigma W_u}$ is $\mathscr{F}_u$ measurable. W_t is a Brownian motion and $W_t \sim N(0,t)$ where $N(0,t)$ is a Gaussian random variable with mean 0 and variance t. The Brownian motion process, $W_t - W_s \sim N(0, t-s)$ where $s < t$, see Chapter 2, Lawler (2017) and Chapter 3, Baxter and Rennie (1996) among other resources. The above conditional expectation, (3.2.1) becomes (the condition was dropped because of independence):

$$E\left[S_t \mid \mathscr{F}_u\right] = S_u\left[e^{(\mu-\frac{1}{2}\sigma^2)(t-u)}E\left(e^{\sigma(W_t-W_u)}\right)\right],$$

where $S_u = S_0e^{(\mu-\frac{1}{2}\sigma^2)u+\sigma W_u}$. Making use of the moment generating function, we have

$$E\left[e^{\sigma(W_t-W_u)}\right] = e^{\frac{1}{2}\sigma^2(t-u)}.$$

Therefore

$$\begin{aligned} E\left[S_t \mid \mathscr{F}_u\right] &= S_ue^{(\mu-\frac{1}{2}\sigma^2)(t-u)}E\left[e^{\sigma(W_t-W_u)}\right] \\ &= S_ue^{(\mu-\frac{1}{2}\sigma^2)(t-u)}e^{\frac{1}{2}\sigma^2(t-u)} \\ E\left[S_t \mid \mathscr{F}_u\right] &= S_ue^{\mu(t-u)}. \end{aligned}$$

This is equivalent to

$$E\left[e^{-\mu t}S_t \mid \mathscr{F}_u\right] = e^{-\mu u}S_u.$$

□

3.3 Risk Neutral Valuation for Derivatives

A martingale measure, under which the discounted stock price, $M_t = e^{-rt}S_t$ becomes a martingale, is called a risk neutral martingale measure. The constant, r denotes here the risk free interest rate. If we assume that such a measure exists then

$$\hat{E}\left[e^{-rt}S_t\right] = \hat{E}\left[e^{-rt}S_t \mid \mathscr{F}_u\right] = e^{-ru}S_u,$$

where $\hat{E}$ denotes the expectation with respect to the risk neutral martingale measure. The previously stated relation can be then written as

$$e^{-r(t-u)}\hat{E}\left[S_t \mid \mathscr{F}_u\right] = S_u, \quad u < t.$$

The following theorem is of paramount importance in pricing financial derivatives. Its proof can be found in Chapter 14, Calin (2017) and many other advanced texts on stochastic calculus.

Theorem 4. *Assume the price process (S_t) follows a Black Scholes model. If $f(S_t, t)$ is the price of the derivative at time t, then $e^{-rt}f(S_t, t)$ is an $\mathscr{F}_t$ martingale under the uniquely determined risk neutral martingale measure, that is*

$$\hat{E}\left[e^{-rt}f(S_t, t) \mid \mathscr{F}_u\right] = e^{-ru}f(S_u, u), \quad u < t.$$

Chapter 4

Finite Difference Methods and Techniques

Numerical methods find great place in computational work and have wider applications in finance and derivatives valuation cannot be sparred. In this chapter we discuss the popular Black Scholes model under various finite difference schemes.

Numerical methods known as finite difference methods for pricing derivatives by approximating the diffusion process are presented. Finite difference methods are a means for generating numerical solutions to partial differential equations and linear complementary (free boundary) problems such as those used in pricing of American options. Finite difference schemes are useful for valuation of derivatives when closed form analytical solutions do not exist or for solutions to complicated multi-factor (multi-dimensional) models. By discretising the continuous time partial differential equation that the derivative security must follow, it is possible to approximate the evolution of the derivative and therefore the present value of the security, see chapter 5 London (2005).

4.1 Fundamentals of Numerical Differentiation

Lets consider a real valued function of a real variable as follows

$$y = f(x).$$

In general we are interested in finding approximations to the first and second order derivatives of the function f. This is needed because, in general the form of the function f is unknown and thus impossible to calculate its derivatives analytically and so we have to use numerical approximations. Lets consider approximating the first derivative of y at some point a and assume that h is a (small) positive number. We approximate the first derivative using the following three approaches:

1. Centered difference formula;

2. Forward difference formula;

3. Backward difference formula.

4.1.1 Centered Difference formula

To come up with the formula we consider the following points, $(a-h, f(a-h))$ and $(a+h, f(a+h))$, the approximation of the first derivative is given by:

$$\begin{aligned} f^{'}(a) &\approx \frac{f(a+h)-f(a-h)}{(a+h)-(a-h)} \\ &= \frac{f(a+h)-f(a-h)}{2h}. \end{aligned}$$

4.1.2 Forward difference formula

The forward difference method considers the following points, $(a, f(a))$ and $(a+h, f(a+h))$ and the first derivative approximation is given by:

$$\begin{aligned} f^{'}(a) &\approx \frac{f(a+h)-f(a)}{(a+h)-(a)} \\ &= \frac{f(a+h)-f(a)}{h}. \end{aligned}$$

4.1.3 Backward difference formula

The backward difference method considers the following points, $(a, f(a))$ and $(a-h, f(a-h))$ and the first derivative approximation is given by:

$$\begin{aligned} f^{'}(a) &\approx \frac{f(a)-f(a-h)}{(a)-(a-h)} \\ &= \frac{f(a)-f(a-h)}{h}. \end{aligned}$$

4.2 Comparison of the three formulas

How good are these approximations to the derivative of f at a point a and which one should we use? To answer these questions we have to examine the central difference case first. We use Taylor's expansion to show that:

$$f(a \pm h) = f(a) \pm h f^{'}(a) \pm \frac{h^2}{2!} f^{''}(a) \pm \frac{h^3}{3!} f^{'''}(\eta_{\pm}),$$

where; $\eta_{-} \in (a-h, a)$ and $\eta_{+} \in (a, a+h)$. To distinguish the formulas above we use the following notation, $D_0 f(a)$ denotes the centered difference, $D_{-} f(a)$ denotes the backward difference formula and $D_{+} f(a)$, the forward difference formula. The centered difference gives a second order approximation to the first derivative if h is small enough and if f has a continuous derivatives up to order 3, that is

$$D_0 f(a) = f^{'}(a) + \frac{h^2}{6}\left(\frac{f^{'''}(\eta_{+}) + f^{'''}(\eta_{-})}{2}\right).$$

This we can show from Taylor series expansion of the above expression:

$$f(a+h) = f(a) + h f^{'}(a) + \frac{h^2}{2!} f^{''}(a) + \frac{h^3}{3!} f^{'''}(\eta_{+}) \tag{4.2.1}$$

$$f(a-h) = f(a) - hf^{'}(a) + \frac{h^2}{2!}f^{''}(a) - \frac{h^3}{3!}f^{'''}(\eta_-). \tag{4.2.2}$$

Subtracting equation 4.2.2 from 4.2.1 we obtain:

$$f(a+h) - f(a-h) = 2hf^{'}(a) + \frac{h^3}{3!}\left[f^{'''}(\eta_+) + f^{'''}(\eta_-)\right]$$

dividing by $2h$ yields the following expression

$$\frac{f(a+h) - f(a-h)}{2h} = f^{'}(a) + \frac{h^2}{6}\left(\frac{f^{'''}(\eta_+) + f^{'''}(\eta_-)}{2}\right).$$

Some arithmetic shows that the forward and backward differencing give the first order approximation to the first derivative of f at point a that is:

$$D_+f(a) = f^{'}(a) + \frac{h}{2}f^{''}(\eta_+), \quad \eta_+ \in (a, a+h)\,.$$

This can also be verified the same way we did for the centered difference formula, we consider Taylor series expansion as follows:

$$f(a+h) = f(a) + hf^{'}(a) + \frac{h^2}{2!}f^{''}(\eta_+). \tag{4.2.3}$$

Rearranging equation 4.2.3 to get:

$$\frac{f(a+h) - f(a)}{h} = f^{'}(a) + \frac{h}{2}f^{''}(\eta_+).$$

The backward differencing is given by:

$$D_-f(a) = f^{'}(a) - \frac{h}{2}f^{''}(\eta_-), \quad \eta_- \in (a-h, a)\,.$$

We observe that these one sided schemes are first order accurate and they also place low continuity constraints on the function f. We only need to assume that the second derivative is continuous, see Duffy (2013b).

4.3 Difference formulas for the second derivative

We now consider the divided differences for the second derivative of f at some point a. The second derivative is denoted by $D_+D_-f(a)$ and is given by:

$$D_+D_-f(a) = \frac{f(a-h) - 2f(a) + f(a+h)}{h^2}.$$

The above expression is obtained from:

$$\begin{aligned} D_+D_-f(a) &= D_-\left(\frac{f(a+h) - f(a)}{h}\right) \\ &= \frac{1}{h}D_-\left(f(a+h) - f(a)\right). \end{aligned}$$

Taking the divided differences of $f(a+h)$ and $f(a)$ with respect to h gives:

$$\begin{aligned} &= \frac{1}{h}\left(\frac{f(a+h)-f(a)}{h} - \frac{f(a)-f(a-h)}{h}\right) \\ &= \frac{f(a+h)-2f(a)+f(a-h)}{h^2}. \end{aligned}$$

This divided difference is a second order approximation to the second derivative of f at point a and we assume that the function has continuous derivatives up to and including order 4. The discretization error is given by:

$$D_+D_-f(a) = f''(a) + \frac{h^2}{4!}\left(f^{(4)}(\eta_+) + f^{(4)}(\eta_-)\right).$$

This can be verified using Taylor series expansion

$$f(a+h) = f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f^{(4)}(\eta_+) \tag{4.3.1}$$

$$f(a-h) = f(a) - hf'(a) + \frac{h^2}{2!}f''(a) - \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f^{(4)}(\eta_-). \tag{4.3.2}$$

Adding 4.3.1 and 4.3.2 we obtain;

$$f(a+h) + f(a-h) = 2f(a) + h^2f''(a) + \frac{h^4}{4!}\left(f^{(4)}(\eta_+) + f^{(4)}(\eta_-)\right).$$

Rearranging the above equation, we get;

$$f(a+h) + f(a-h) - 2f(a) = h^2f''(a) + \frac{h^4}{4!}\left(f^{(4)}(\eta_+) + f^{(4)}(\eta_-)\right).$$

Dividing the above equation by h^2, gives the desired result;

$$\frac{f(a+h) + f(a-h) - 2f(a)}{h^2} = f''(a) + \frac{h^2}{4!}\left(f^{(4)}(\eta_+) + f^{(4)}(\eta_-)\right).$$

4.4 Explicit Difference Method

In this section we consider the explicit finite difference method with particular reference to the Black-Scholes partial differential equation. Finite difference methods are used to value more complex derivatives with non linear payoffs, such as exotic options. They are used to price derivatives by solving the differential equation in conjunction with the initial asset price condition and boundary value conditions (that is payoffs) that the derivative must also satisfy. The differential equation is converted into a system of difference equations that are solved iteratively. Let's consider the Black-Scholes PDE:

$$\frac{\partial f}{\partial t} + (r-q)S\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2S^2\frac{\partial^2 f}{\partial S^2} = rf, \tag{4.4.1}$$

subject to the payoff condition $f(S_T, T) = (S_T - X)^+$, see London (2005). When implementing finite difference methods, we divide space and time into discrete intervals, Δx and Δt, which generate the lattice. We add boundary conditions to the grid, which determines the

option prices as a function of the asset price for high and low values so that $\frac{\partial f}{\partial S} = 1$ for large S and $\frac{\partial f}{\partial S} = 0$ for S small. We can now discretize the PDE to develop a numerical finite difference scheme. First we simplify the PDE by letting $x = logS$ and $\frac{dx}{dS} = \frac{1}{S}$ thus $dS = Sdx$ so that:

$$\frac{\partial f}{\partial t} + \mu\frac{\partial f}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 f}{\partial x^2} = rf,$$

where; $\mu = r - q$. To get rid of the rf term on the right hand side, we let $u(x,t)$ be a new function:

$$u(x,t) = e^{r(T-t)}f(e^x, t),$$

since $x = logS$ and $S = e^x$, the term u is a forward price of the option f and satisfies the PDE;

$$\mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} = -\frac{\partial u}{\partial t}.$$

We will discretize this PDE by taking the central difference of the state variable x and the forward difference of the time variable t. We denote $u_j^n = u(x_j, t_i)$, $t_i = i\Delta t$ and $x_j = j\Delta x$, see London (2005) and Boyle and Tian (1998). From the difference formulas above the second derivative's finite difference is given by:

$$\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}.$$

The first derivative of the space variable using the centered difference is given by:

$$\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}$$

and for time we have;

$$\frac{u_j^{n+1} - u_j^n}{\Delta t}.$$

Substituting the finite differences into the the PDE we obtain:

$$\frac{1}{2}\sigma^2\left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}\right) + \mu\left(\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right) = -\left(\frac{u_j^n - u_j^{n+1}}{\Delta t}\right).$$

Rearranging the terms and making u_j^{n+1} the subject we get:

$$u_j^{n+1} = u_{j+1}^n\left(\frac{\Delta t\sigma^2}{2(\Delta x)^2} + \frac{\mu\Delta t}{2\Delta x}\right) + u_j^n\left(1 - \frac{\Delta t\sigma^2}{(\Delta x)^2}\right) + u_{j-1}^n\left(\frac{\Delta t\sigma^2}{2(\Delta x)^2} - \frac{\mu\Delta t}{2\Delta x}\right).$$

The recurrent relation for the forward option price is:

$$u_j^{n+1} = \tilde{p_u}u_{j+1}^n + \tilde{p_m}u_j^n + \tilde{p_d}u_{j-1}^n, \tag{4.4.2}$$

where:

$$\tilde{p_u} = \frac{\Delta t\sigma^2}{2(\Delta x)^2} + \frac{\mu\Delta t}{2\Delta x}, \qquad \tilde{p_m} = 1 - \frac{\Delta t\sigma^2}{(\Delta x)^2}, \qquad \tilde{p_d} = \frac{\Delta t\sigma^2}{2(\Delta x)^2} - \frac{\mu\Delta t}{2\Delta x}.$$

Lets note that

$$\tilde{p_u} + \tilde{p_m} + \tilde{p_d} = 1.$$

Thus $\tilde{p_u}$, $\tilde{p_m}$ and $\tilde{p_d}$ can be interpreted as probabilities and a sufficient condition for convergence is that $\tilde{p_u}$, $\tilde{p_m}$ and $\tilde{p_d}$ must all be non negative for sufficiently small Δt, see Boyle and Tian (1998). If we substitute the present value of the option $f_j^{n+1} = e^{-r(T-t)}u_j^{n+1}$ into the equation 4.4.2, we arrive at the backward relationship

$$f_j^{n+1} = e^{-r\Delta t}\left(\tilde{p_u}f_{j+1}^n + \tilde{p_m}f_j^n + \tilde{p_d}f_{j-1}^n\right).$$

This is equivalent to the discounted expectation of the forward option price (under a risk-neutral measure), see London (2005).

4.5 Implicit Finite Difference

In this section we focus on the implicit schemes where the solution of the finite difference equation at time level $n+1$ is calculated from the solution at the level n by solving a system of linear equations. The difference between these methods lies in how they calculate values at time level $n+1$ terms of the solution at level n. Thus the values of the solution are known at level n and we march to time $n+1$ in order to compute a solution. In financial literature, we march from a "terminal" value to at level $n+1$ to the solution at level n, see Duffy (2013a). We consider the fully implicit method for the Black-Scholes PDE;

$$\mu\frac{\partial u}{\partial x}+\frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2}=-\frac{\partial u}{\partial t}.$$

It is approximated by:

$$\frac{1}{2}\sigma^2\left(\frac{u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1}}{(\Delta x)^2}\right)+\mu\left(\frac{u_{j+1}^{n+1}-u_{j-1}^{n+1}}{2\Delta x}\right)=-\left(\frac{u_j^{n+1}-u_j^n}{\Delta t}\right).$$

The above relation reduces to;

$$u_j^n=\tilde{p_u}u_{j+1}^{n+1}+\tilde{p_m}u_j^{n+1}+\tilde{p_d}u_{j-1}^{n+1},$$

where:

$$\tilde{p_u}=\frac{\Delta t\sigma^2}{2(\Delta x)^2}+\frac{\mu\Delta t}{2\Delta x},\qquad \tilde{p_m}=1-\frac{\Delta t\sigma^2}{(\Delta x)^2},\qquad \tilde{p_d}=\frac{\Delta t\sigma^2}{2(\Delta x)^2}-\frac{\mu\Delta t}{2\Delta x}.$$

In this scheme we have one known solution and three unknowns namely $\tilde{p_u}$, $\tilde{p_m}$ and $\tilde{p_d}$, see London (2005). This scheme is more difficult than the explicit scheme because we have to solve a system of equations. We now have to solve tridiagonal matrix system;

$$\boldsymbol{A}\boldsymbol{U}^{n+1}=U^n,$$

where; $\boldsymbol{U}^{n+1}$ is a vector of the option values and

$$\boldsymbol{A} \;=\; \begin{pmatrix} \tilde{p_{m_1}} & \tilde{p_{d_1}} & . & . & 0 \\ \tilde{p_{u_2}} & \tilde{p_{m_2}} & \tilde{p_{d_2}} & & . \\ . & \tilde{p_{u_3}} & . & . & . \\ . & & . & . & \tilde{p_{d_{n-1}}} \\ 0 & . & . & \tilde{p_{u_n}} & \tilde{p_{m_n}} \end{pmatrix}$$

, see Duffy (2013a).

Definition 15. (Spectral Radius) Let $A=(a_{ij})$ be an $n\times n$ real matrix with eigenvalues λ_j , $j=1,...,n$. The spectral radius $\rho(A)$ is given by;

$$\rho(A) \;=\; max|\lambda_j|\ , j=1,...,n.$$

Definition 16. (Unconditional Stability) The time-dependent matrix $T(t)$ is stable for $0\leq t\leq T$ if;

$$\rho[T(t)] \;\leq\; 1.$$

It is unconditionally stable if;

$$\rho[T(t)] \;<\; 1,\ \forall t:\ 0\leq t\leq \infty,$$

see Duffy (2013b).

Let us consider a one step finite difference scheme for an initial value problem in the vector form;

$$\boldsymbol{u^{n+1}} = \boldsymbol{Qu^n}, n \geq 0 \tag{4.5.1}$$

where: $\boldsymbol{u^n} = \left(..., u_{-1}^n, u_0^n, ...\right)'$ and $\boldsymbol{Q}$ is an operator.

Definition 17. (Stability) The difference scheme (4.5.1) is said to be stable with respect to the norm, $\|.\|$, if there exist positive constants k_0 and h_0 and two non-negative constants K and β such that;

$$\left\|\boldsymbol{u^{n+1}}\right\| \leq Ke^{\beta t} \left\|\boldsymbol{u^n}\right\| \; for \; 0 \leq t = t_n + 1, \; 0 < h \leq h_0 \; and \; 0 < k \leq k_0.$$

We now generalise (4.5.1) to include an inhomogeneous term;

$$\boldsymbol{u^{n+1}} = \boldsymbol{Q\,u^n} + k\boldsymbol{G^n}, \tag{4.5.2}$$

where; $\boldsymbol{G^n} = \left(..., G_{-1}^n, G_0^n, G_1^n, ...\right)'$ is the inhomogeneous term.

Definition 18. (Accuracy of order (p, q)) The difference scheme (4.5.2) is said to be accurate of order (p, q) to the given partial differential equation if;

$$\|\boldsymbol{\tau}^n\| = O(h^p) + O(k^q),$$

where; h and k are positive constants and $\|\boldsymbol{\tau}^n\|$, is the truncation error, Duffy (2013b).

Definition 19. (Two-factor Black-Scholes equation) The general two-dimensional Black–

Scholes equation is given by;

$$-\frac{\partial u}{\partial t} + L_1 u + L_2 u + L_3 u = f,$$

where;

$$L_j u = \sigma_j \frac{\partial^2 u}{\partial x^2} + \mu_j \frac{\partial u}{\partial x_j} + b_J u, \; j = 1, 2$$

and

$$L_3 u = \rho_{12} \frac{\partial^2 u}{\partial x_1 \partial x_2} \; (cross \; term).$$

There are three main operators, namely two convection–diffusion operators and a cross-term. These correspond to single-factor models and coupling respectively, chapter 21, Duffy (2013a).

4.6 Crank-Nicolson

Both the Crank-Nicolson and the implicit methods use the centered difference to approximate the first and second derivatives of the option price with respect to the space variable, x, in this case while they employ one step differencing in time, t. We can view the Crank-Nicolson

as the averaged scheme by adding the fully explicit and implicit schemes together. We define the quantity

$$u_j^{n+\frac{1}{2}} = \frac{1}{2}\left(u_j^{n+1} + u_j^n\right).$$

The Crank-Nicolson method has gained wide acceptance in the financial literature and seems to be standard finite difference scheme for one and two factor Black-Scholes equations. It has second order accuracy in the time parameter and is stable, see Duffy (2013a). It has been known for some considerable time that the centered differencing in space combined with averaging in time leads to spurious oscillations in the approximate solution and in the divided differences approximating its derivatives. In this section we use the Crank-Nicolson scheme and implement the Black-Scholes partial differential equation below:

$$\mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} = -\frac{\partial u}{\partial t}$$

and this becomes;

$$\mu\left(\frac{u_{j+1}^{n+\frac{1}{2}} - u_{j-1}^{n+\frac{1}{2}}}{2\Delta x}\right) + \frac{1}{2}\sigma^2\left(\frac{u_{j+1}^{n+\frac{1}{2}} - 2u_j^{n+\frac{1}{2}} + u_{j-1}^{n+\frac{1}{2}}}{(\Delta x)^2}\right) = -\frac{u_j^{n+1} - u_j^n}{\Delta t}.$$

Making use of $u_j^{n+\frac{1}{2}} = \frac{1}{2}\left(u_j^{n+1} + u_j^n\right)$ and simplifying we get:

$$-\alpha^0 u_{j+1}^n + \beta^0 u_j^n - \gamma^0 u_{j-1}^n = \alpha^0 u_{j+1}^{n+1} + \beta^{0*} u_j^{n+1} + \gamma^0 u_{j+1}^n,$$

where:

$$\begin{aligned}
\alpha^0 &= \left(\frac{\mu\Delta t}{4\Delta x} + \frac{\sigma^2\Delta t}{4(\Delta x)^2}\right),\\
\beta^0 &= \left(1 + \frac{\sigma^2\Delta t}{2(\Delta x)^2}\right),\\
\beta^{0*} &= \left(1 - \frac{\sigma^2\Delta t}{2(\Delta x)^2}\right),\\
\gamma^0 &= \left(\frac{\sigma^2\Delta t}{4(\Delta x)^2} - \frac{\mu\Delta t}{4\Delta x}\right).
\end{aligned}$$

There are variations in notation and arithmetic manipulations of our finite schemes to the referenced sources. For further discussion see Duffy (2013b), Duffy (2013a) and London (2005).

4.6.1 Criticism of the Crank-Nicolson scheme

The Crank-Nicolson method has become a very popular finite difference scheme for solving the Black-Scholes equation, which is an example of a *convection-diffusion* equation, see Duffy (2013a). The main problem is that the traditional finite difference schemes start to oscillate when the coefficient of the second derivative (the diffusion term) is very small or when the coefficient of the first derivative (the convection term) is large (or both). In this case the mesh size Δx, in the space direction must be smaller than a certain value, if we wish to avoid these oscillations, see Duffy (2013a).

Chapter 5

Generalisations: Lévy Processes

Chapter 5 is developed as an extension or generalisation of the Black Scholes model. This chapter has strong theoretical foundations in chapters 2 and 3. Lévy processes play a central role in several fields of science, such as physics; in the study of turbulence, laser cooling and in quantum field theory; in engineering for the study of networks, queues and dams; in economics for continuous time series models; in actuarial science for calculation of insurance and reinsurance risk and mathematical finance cannot be spared. In mathematical finance, Lévy processes are becoming extremely fashionable because they describe the observed reality of the financial markets in a more accurate way than the models based on Brownian motion Papapantoleon (2008).

The Lévy stable model was introduced to finance by Mandelbrot (1963); however, in its general form it cannot be used for pricing, since its exponential moments do not exist, therefore, should one use an exponential Lévy stable model to price a stock, its expected future price would be infinite. The variance gamma (VG) model, introduced to finance by Madan and Seneta (1990), was the first non-Gaussian model used in derivative pricing. In recent years, however, the VG model has become less relevant in practical applications, since empirical studies have shown that, for calibrations to liquid option prices, VG seldom provides a best fit within a wider class of Lévy processes. Later, a host of other models were introduced which were also based on Lévy processes with exponentially decaying tails (semi-heavy tails), but without the undesirable property of the PDFs of VG processes, which decay rather slowly at infinity.

5.1 Stylized statistical properties of asset returns

Stylized statistical facts which are common to a wide set of financial assets are presented below;

5.1.1 Absence of Autocorrelations:

(Linear) autocorrelations of asset returns are often insignificant, except for very small intraday time scales (20 minutes) for which microstructure effects come into play.

5.1.2 Heavy tails

The (unconditional) distribution of returns seems to display a power-law or Pareto-like tail, with a tail index which is finite, higher than two and less than five for most data sets studied. In particular this excludes stable laws with infinite variance and the normal distribution. However the precise form of the tails is difficult to determine.

5.1.3 Gain/loss asymmetry

One observes large drawdowns in stock prices and stock index values but not equally large upward movements.

5.1.4 Aggregational Gaussianity

As one increases the time scale, Δt over which returns are calculated, their distribution looks more and more like a normal distribution. In particular, the shape of the distribution is not the same at different time scales.

5.1.5 Intermittency

Returns display, at any time scale, a high degree of variability. This is quantified by the presence of irregular bursts in time series of a wide variety of volatility estimators.

5.1.6 Volatility clustering

Different measures of volatility display a positive autocorrelation over several days, which quantifies the fact that high-volatility events tend to cluster in time.

5.1.7 Conditional heavy tails

Even after correcting returns for volatility clustering (e.g. via GARCH-type models), the residual time series still exhibit heavy tails. However, the tails are less heavy than in the unconditional distribution of returns.

5.2 Some Stylized statistical properties in brief

In this section we briefly describe some of the stylized statistical properties. As in Cont (2001), let $S(t)$ be an asset price, say a stock, an exchange rate or a market index. We are considering the log returns at time t, given by;

$$r(t, \Delta t) \quad = \quad ln(\frac{S(t + \Delta t)}{S(t)}),$$

where; Δt is a time scale which can range from a few seconds to a month.

5.2.1 Leptokurtic distributions

The financial data displays a high peak and asymmetric heavy tails. The name leptokurtic is given, which means the kurtosis of the distribution is large. More precisely, the kurtosis and skewness are defined as:

$$k = E\left(\frac{(X-\mu)^4}{\sigma^4}\right),$$

where; k is the kurtosis, σ is the standard deviation and μ is the mean

$$s = E\left(\frac{(X-\mu)^3}{\sigma^3}\right),$$

where: s is the skewness, σ is the standard deviation and μ is the mean. For a standard normal distribution $k = 3$. If $k > 3$ the distribution is called leptokurtic and the distribution will have a higher peak and two heavier tails than those of a normal distribution. Examples of leptokurtic distributions include the double exponential distribution with the density;

$$f(x) = p\theta_1 e^{-x\theta_1} 1_{\{x>0\}} + (1-p)\theta_2 e^{x\theta_2} 1_{\{x<0\}}.$$

The leptokurtic feature has been observed since 1950s. However classical finance models simply ignore this feature for example in Black Scholes Brownian motion model, the stock price is modelled as a geometric Brownian motion

$$S_t = S_0 e^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)},$$

where; $W(t)$ has a normal distribution with mean zero and variance t. Here μ is called the drift, which measures the average return and σ is the standard deviation of the return distribution. In this model, the continuously compounded return, $ln\left(\frac{S(t)}{S(0)}\right)$ has a normal distribution which is not consistent with leptokurtic feature. Many alternative models for example models with jumps or stochastic volatility have been proposed to incorporate the feature, Staum (2007).

5.2.2 Heavy Tails

Serious interest in the functional form of the distribution of prices began with Mandelbrot's 1963 study of cotton prices, in which he showed that logarithmic price returns are far from normal and suggested that they might be drawn from a Lévy distribution. There have been many studies since then, most of which indicate that the cumulative distribution of logarithmic price changes has tails that asymptotically scale for large $|r|$ as a power law of the form $|r|^{-\alpha_r}$, see Mike and Farmer (2008). The exponent, α_r which takes on typical values in the range $2 < \alpha_r < 4$ is called a tail exponent. It is important because it characterizes the risk of extreme price movements and corresponds to the threshold above which the moments of the distribution become infinite. Having a good characterization of the the price returns has important practical consequences for risk control and option pricing.

From theoretical point of view, the heavy tails of the price returns excite interest among physicists because they suggest non-equilibrium behaviour. A fundamental result in Statistical mechanics is that except for unusual situations such as phase transitions, equilibrium distributions are either exponential or normal distributions. The fact that price returns have tails that are heavier than this suggests that markets are not in equilibrium. Many models have been proposed that attempt to explain the heavy tails of the price returns, see Mike and Farmer (2008). An exposition of how heavy the returns distributions are, is well documented in Cont (2001).

5.2.3 Power tails and exponential tails

The returns of stocks have two tail distributions heavier than the normal distribution. However how heavy the stock tail distributions are is debatable. Two main classes proposed in literature are the power-type tails and the exponential-type tails. We say that the right tail of the random variable X has a power-type tail if $P(X > x) \approx \frac{c}{x^\alpha}$, $x > 0$ as $x \to \infty$ and the left tail of X has a power-type tail if $P(X < -x) \approx \frac{c}{x^\alpha}$, $x < 0$ as $x \to \infty$. X has a right exponential-type tail if $P(X > x) \approx ce^{-\alpha x}$, $x > 0$ as $x \to \infty$ and a left exponential type tail if $P(X < -x) \approx ce^{-\alpha x}$, $x < 0$ as $x \to \infty$, see Staum (2007).

5.2.4 Volatility Clustering

One of the best known stylized features of financial asset returns is volatility clustering. That is, high volatility movements (represented by large fluctuations in returns) tend to be followed by high volatility movements (characterized by a period of relative turbulence in the market), while low volatility movements (indicated by small fluctuations in returns) tend to be followed by low volatility movements (characterized by a period of a relative tranquility in the market). The former is indicative of the clusters of high volatility while the later is suggestive of the clusters of low volatility. This pattern of volatility clustering appears to describe the dynamics of volatility fluctuations in financial asset returns. Modeling this type of volatility clustering in the financial market is important since volatility of asset returns can directly impact prices of options and risks of stocks and portfolios. Volatility clustering has mainly been analyzed within the class of Generalized Autoregressive Conditional Heteroscedastic (GARCH) and Stochastic Volatility (SV) models. Each of these classes of models is capable of capturing the volatility clustering in asset returns in a symmetric way, see Ning et al. (2015). Volatility clustering and non linear dependence is well covered in Cont (2001).

5.2.5 Implied Volatility Smile

Volatility measures how much prices move. The direction of the move, whether up or down, is irrelevant. The magnitude and the speed of the change are most important. In the futures and options markets, the implied volatility is often used, which refers how much the market anticipates price changes and is reflected on the prices of contracts. Implied volatility is used independently or together with historical volatility and other different methodologies for instance Stochastic volatility, GARCH and EWMA (Exponential Weighted Moving Average) models to estimate the future volatility, see García-Machado and Rybczyński (2015).

5.3 Some alternative models to the Black-Scholes

Many studies have been conducted to modify the Black-Scholes model to explain the above empirical stylized facts; leptokurtic feature, volatility clustering and implied volatility smile among others. The following subsections consider a list of the proposed models:

5.3.1 Chaos Theory and fractal Brownian Motion

Chaos Theory and Fractal Brownian Motion, have dependent increments rather than independent, for example Mandelbrot (1963). According to Staum (2007), Rogers pointed out

that these models may lead to arbitrage opportunities. Fractal Brownian motion (FBM) is characterized by its Hurst exponent, $H \in (0,1)$. Given any $H \in (0,1)$, the associated FBM, W^H is a Gaussian stochastic process that can be defined on an appropriate probability space, $\left(\Omega, \mathscr{F}, P^H\right)$ satisfying the following properties;

1. the process W^H has continuous sample paths with $W_0^H = 0$;
2. W_t^H is a zero mean Gaussian random variable $\forall t \geq 0$;
3. $E\left[W_t^H W_s^H\right] = \frac{1}{2}\left[t^{2H} + s^{2H} - |t-s|^{2H}\right]$ $\forall t, s \geq 0$, where E denotes the expectation with respect to measure P^H, see Brody et al. (2002).

Definition 20. (Fractal Brownian motion) If $H \in (0,1)$, the fractal Brownian motion with Hurst parameter, H is a zero mean centered Gaussian process W^H with continuous paths such that $\forall s, t \geq 0$

$$E\left[\left(W_t^H - W_s^H\right)^2\right] = |t-s|^{2H}.$$

If $H = \frac{1}{2}$, the correlation between increments is zero and we recover a standard Brownian motion. Positive correlation exists between the increments when $H > \frac{1}{2}$ and negative correlation exists for $H < \frac{1}{2}$, see Lawler (2017).

5.3.2 Stochastic Volatility and GARCH models

Stochastic Volatility and GARCH models, for example the 1987 model by Hull and White, Engle's GARCH model of 1995 and Heston's Stochastic Volatility model of 1993, have mainly been designed to capture the volatility clustering effect. A typical example of these models is:

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma(t) dW_1(t)$$

$$d\sigma(t) = -\alpha(\sigma(t) - \beta)dt + \gamma\sqrt{\sigma(t)}dW_2(t),$$

where; $W_1(t)$ and $W_2(t)$ are correlated Brownian motions and γ, is the correlation coefficient. The generalised autoregressive conditional heteroscedastic, GARCH(p,q) model is given by:

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2,$$

where: w is the mean, α_i, the error coefficient, measures the degree of volatility reactivity to the unexpected news and β_j, the lag coefficient, measures the degree of volatility persistence from previous shocks to the conditional variance.

5.4 Characteristic Functions

The characteristic function of a random variable is the Fourier transform of its distribution. Many probabilistic properties of random variables correspond to analytical properties of their characteristic functions, making this concept very useful for studying random variables.

Definition 21. (Characteristic function) The characteristic function of the $\mathbb{R}$ valued random variable X is the function $\varphi_X : \mathbb{R} \to \mathbb{R}$ defined by

$$\forall s \in \mathbb{R},\ \varphi_X(s) = E\left[e^{isX}\right] = \int_{\mathbb{R}} e^{isx} d\mu(x)$$

where; X has distribution (law) μ.

The characteristic function of a random variable completely characterizes its law; two variables with the same characteristic function are identically distributed. A characteristic function is always continuous and verifies $\varphi_X(0) = 1$.

Proposition 2. ***(Characteristic function and moments)***

1. If $E\left[|X|^n\right] < \infty$, then φ_X has n continuous derivatives at $s = 0$ and $\forall k = 1, 2.., n$,

$$m_k \equiv E\left[X^k\right] = \varphi_X^{(k)}(0) = \frac{1}{i^k} \frac{\mathrm{d}^k \varphi_X(0)}{\mathrm{d}s^k} \mid_{s=0} . \tag{5.4.1}$$

2. If φ_X has $2n$ continuous derivatives at $s = 0$ then $E\left[|X|^{2n}\right] < \infty$ and $\forall k = 1, 2.., 2n$,

$$m_k \equiv E\left[X^k\right] = \varphi_X^{(k)}(0) = \frac{1}{i^k} \frac{\mathrm{d}^k \varphi_X(0)}{\mathrm{d}s^k} \mid_{s=0},$$

see Tankov (2003).

5.5 Jump-Processes

The assumptions that led to the Brownian motion were:

1. Independent increments;
2. Stationary increments and
3. Continuous paths.

To consider more general processes we need to give up at least one of the assumptions. Here we will not assume continuity of the paths, Lawler (2017). The simplest Lévy process is a linear drift, a deterministic process. Brownian motion is the only non-deterministic Lévy process with continuous sample paths. Other examples of Lévy processes are the Poisson and Compound Poisson, Papapantoleon (2008). More examples of Lévy processes include the Gamma model and Variance Gamma model, see Pflug and Thoma (2016). The sum of linear drift, a Brownian motion and a compound Poisson process is again a Lévy process, it is often called a "jump-diffusion process". We shall call it a "Lévy jump-diffusion" process since there exist other jump-diffusion processes which are not Lévy processes, Papapantoleon (2008). Let $(\Omega, \mathscr{F}, P)$ be a filtered probability space where $\mathscr{F} = (\mathscr{F}_t)_{0 \le t \le T}$ satisfies the usual conditions. A filtered probability space is said to satisfy the usual conditions if it is complete (that is contains all sets of measure zero) and right continuous (for all times).

Definition 22. (Cadlag) We call a function a cadlag, if the paths are right continuous everywhere and have left limits.

The paths of the usual Poisson and the compound Poisson processes are cadlags, Lawler (2017).

Definition 23. (Lévy process) A cadlag, adapted, real valued stochastic process $X = (X_t)_{0\leq t\leq T}$ with $X_0 = 0$ is called a Lévy process if the following conditions are satisfied:

1. has independent increments that is $X_t - X_s$ is independent of $\mathscr{F}_s$ for any $0 \leq s < t \leq T$;
2. X has stationary increments that is for any $t \leq T$ the distribution of $X_{t+s} - X_t$ does not depend on t;
3. X is stochastically continuous that is for every $0 \leq t \leq T$ and $\varepsilon > 0$

$$\lim_{s\to t} P\left(|X_s - X_t| > \varepsilon\right) = 0,$$

Papapantoleon (2008). An adapted process is defined in definition (3).

5.6 Poisson processes

The Poisson process is the basic jump Lévy process. It is obtained by taking jumps of size one at a particular rate λ.

Definition 24. (Poisson process) A cadlag, adapted stochastic process $N = (N_t)_{0\leq t\leq T}$ with $N_t : \Omega \times \mathbb{R} \to \mathbb{N}\bigcup\{0\}$ is called a Poisson process if:

1. $N_0 = 0$;
2. $(N_t)_{0\leq t\leq T}$ has independent increments, $N_t - N_s$ is independent of $\mathscr{F}_s$ for any $0 \leq s < t < T$;
3. $N_t - N_s$ is Poisson distributed with parameter $\lambda(t - s)$ for any $0 \leq s < t < T$. Then λ is also called the intensity of the Poisson process.

Suppose we have such a process and let $p(s)$ denote the probability that there is at least one jump in the interval $[t,\, t + s]$. If increments are to be stationary, this quantity must not depend on t. Rate λ means that the probability that there is a jump sometime during the time interval [t, t+Δt] is about $\lambda\Delta$t; more precisely, $p(\Delta\text{t}) = \lambda\Delta\text{t} + o(\Delta\text{t})$ as $\Delta\text{t} \downarrow 0$. We can use this observation to construct the Poisson process. We consider the waiting times between the jumps, let X_t denote the number of jumps that have occurred by time t and let $T = inf\{t :\ X_t = 1\}$ denote the amount of time until the first jump. Using the assumption of independent stationary increments, we have that:

$$\begin{aligned} P\{T > t\} &= \prod_{j=1}^{n} P\left(no\ jump\ during \left[\frac{(j-1)t}{n},\ \frac{jt}{n}\right]\right) \\ &= \left[1 - p(\frac{t}{n})\right]^n. \end{aligned}$$

Therefore:

$$\begin{aligned} P[T > t] &= \lim_{n\to\infty} \left[1 - p(\frac{t}{n})\right]^n \\ &= \lim_{n\to\infty} \left[1 - \frac{\lambda t}{n} + o\left(\frac{\lambda t}{n}\right)\right]^n \\ &= e^{-\lambda t}. \end{aligned}$$

We have used that; $\lim_{n\to\infty} o\left(\frac{\lambda t}{n}\right) \to 0$ and $\lim_{n\to\infty} \left(1 - \frac{x}{n}\right)^n \to e^{-x}$. A random variable T has an exponential distribution with rate λ if it has density

$$f(t) = \lambda e^{-\lambda t}, \quad 0 < t < \infty$$

and hence

$$P[T > t] = \int_t^\infty f(s)ds = e^{-\lambda t}.$$

Our assumptions imply that the waiting times of a Poisson distribution with parameter λ must be exponential with rate λ.

$$E[T] = \int_0^\infty t f(t)dt = \frac{1}{\lambda}.$$

This observation gives a way to construct a Poisson process, this construction also gives a good way to simulate Poisson processes. Let $T_1, T_2, ...$ be independent random variables each exponential with rate λ, let $\xi_0 = 0$ for positive integer n,

$$\xi_n = T_1 + T_2 + ... + T_n$$

in other words, ξ_n is the time at which the n^{th} jump occurs. ξ_n is gamma distributed with parameters n and λ, see Lawler (2017), Papapantoleon (2008) and Tankov (2003).

5.6.1 Characteristic Exponent

Definition 25. (Characteristic exponent) If X is a random variable, then its characteristic exponent, $\psi(s) = \psi_X(s)$ is defined to be the continuous function satisfying

$$E[e^{isX}] = e^{\psi(s)}, \ \psi(0) = 0.$$

If X_t is a Lévy process, then the characteristic exponent of the process is the characteristic of $X_1 - X_0$. If X and Y are independent then:

$$\psi_{X+Y} = \psi_X + \psi_Y. \tag{5.6.1}$$

If $X \sim N(\mu, \sigma^2)$ then:

$$\psi(s) = i\mu s - \frac{\sigma^2 s^2}{2}.$$

If X is Poisson with mean λ then:

$$\psi(s) = \lambda\left[e^{is} - 1\right].$$

This can be seen from the computation:

$$\begin{aligned} E\left[e^{isX}\right] &= \sum_{n=0}^{\infty} e^{isn} P\left[X=n\right] \\ &= \sum_{n=0}^{\infty} \frac{(\lambda e^{is})^n e^{-\lambda}}{n!} \\ &= e^{\lambda(e^{is}-1)}. \end{aligned}$$

If X_t is a Lévy process starting at the origin with the characteristic exponent φ then independent stationary increments and relation 5.6.1 imply that

$$\psi_{X_t} = t\psi_{X_1} = t\psi.$$

An important property of the Lévy process is the following: Suppose that X_t^1 and X_t^2 are independent Lévy processes with $\psi_{X_1} = \psi_1$, $\psi_{X_2} = \psi_2$ then $X_t = X_t^1 + X_t^2$ is a Lévy process with $\psi_{X_t} = \psi_1 + \psi_2$. For example if X_t^1 is a Brownian motion with drift μ and variance σ^2 and X_t^2 is an independent Poisson process with rate λ then $X_t = X_t^1 + X_t^2$ is a Lévy process with

$$\psi(s) = i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left[e^{is} - 1\right].$$

5.6.2 Generator for a Poisson process

In this subsection, we start by defining a generator of a stochastic process, X_t. The definition considers a smooth function and is extended to a Poisson process that has jumps.

Definition 26. (Generator of Itô Diffusion) The generator of the stochastic process X_t , is the second order partial differential operator L defined by;

$$L\left[f(x)\right] = \lim_{\Delta t \to 0} \frac{E[f(X_t) \mid X_0 = x] - f(x)}{\Delta t},$$

for any smooth function with compact support $f : \mathbb{R}^n \to \mathbb{R}$. E, stands for the expectation operator.

The generator for a Brownian motion, W_t with drift μ and variance σ^2, is;

$$L\left[f(x)\right] = \mu f^{'}(x) + \frac{\sigma^2}{2} f^{''}(x).$$

We now compute the generator for the Poisson process. Although the Poisson process takes integer values, the definition can be extended so that $X_0 = x$. In this case, in a small time interval Δt, the values taken by the process are $x, x+1, x+2, ...$

$$P\left[X_{\Delta t} = X_0 + 1\right] = 1 - P\left[X_{\Delta t} = X_0\right] = \lambda \Delta t + o(\Delta t).$$

The expectation is given by;

$$E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] = \lambda \Delta t f(x+1) + \left[1 - \lambda \Delta t\right] f\left(x\right) + o(\Delta t),$$

thus

$$E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] - f\left(x\right) = \lambda \Delta t f(x+1) - \lambda \Delta t f\left(x\right) + o(\Delta t).$$

Taking the limit as $\Delta t \to 0$,

$$\lim_{\Delta t \to 0} \frac{E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] - f\left(x\right)}{\Delta t} = \lambda f(x+1) - \lambda f\left(x\right) + \lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t},$$

but $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} \to 0$**.** Now we have;

$$\lim_{\Delta t \to 0} \frac{E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] - f\left(x\right)}{\Delta t} = \lambda f(x+1) - \lambda f\left(x\right).$$

Therefore;

$$L\left[f(x)\right] = \lambda\left[f(x+1) - f(x)\right].$$

5.7 Compensated Poisson process

Definition 27. (Compensated Poisson process) Let N be a Poisson process with parameter λ. The process $\bar{N} = (\bar{N}_t)_{0 \le t \le T}$ with $N_t : \Omega \times \mathbb{R} \to \mathbb{R}$ where;

$$\bar{N} = N_t - \lambda t$$

is called a compensated Poisson process.

Proposition 3. *The compensated Poisson process is a martingale.*

Proof. We have that: □

1. The process $\bar{N}$ is adapted to filtration because N is adapted (by definition);
2. $E[\bar{N}_t] < \infty$ because $E[N_t] < \infty$ for all $0 < t < T$.

Finally, let $0 \le s < t < T$ then:

$$\begin{aligned} E\left[\bar{N}_t \mid \mathscr{F}_s\right] &= E\left[N_t - \lambda t \mid \mathscr{F}_s\right] \\ &= E\left[(N_t - N_s) + N_s \mid \mathscr{F}_s\right] - \lambda((t-s)+s) \\ &= E\left[N_s \mid \mathscr{F}_s\right] + E\left[N_t - N_s \mid \mathscr{F}_s\right] - \lambda\left((t-s)+s\right) \\ &= N_s + E\left[N_t - N_s\right] - \lambda(t-s) - \lambda s \\ &= N_s + \lambda(t-s) - \lambda(t-s) - \lambda s \\ &= N_s - \lambda s = \bar{N}_s. \end{aligned}$$

The characteristic function of the compensated Poisson random variable is given by:

$$\begin{aligned} E\left[e^{is\bar{N}_t}\right] &= E\left[e^{is(N_t - \lambda t)}\right] \\ &= e^{-\lambda t i s} E\left[e^{isN_t}\right] \\ &= e^{-\lambda t i s} e^{\lambda t\left(e^{is}-1\right)} \\ &= exp\left\{\lambda t\left(e^{is} - 1 - is\right)\right\}. \end{aligned}$$

5.8 Compound Poisson process

A stochastic process $\{Y_t,\, t \geq 0\}$ is said to be a compound Poisson process, if it can be represented as:

$$Y_t = \sum_{k=1}^{N(t)} J_k, \quad t \geq 0,$$

where; $\{N(t),\, t \geq 0\}$ is a Poisson process with intensity λ, $\{J_k,\, k \geq 1\}$ is a family of independent and identically random variables that are also independent of $\{N(t),\, t \geq 0\}$. When $J_1 = 1$ then $Y_t = N(t)$ and we have the usual Poisson process.

5.8.1 Properties of the Compound Poisson process

1. The sample paths of Y_t are cadlag piece-wise constant functions.

2. The jump times $(T_i)_{i\geq 1}$ have the same distribution (law) as the jump times of a Poisson process, $\{N(t),\, t \geq 0\}$; they can be expressed as partial sums of independent exponential random variables with parameter λ.

3. The jump sizes $\{J_k,\, k \geq 1\}$ are independent and identically distributed with distribution (law) $\mu^{\#}$.

Definition 28. (Lévy measure) Let X_t be a Lévy process, the measure ν on $\mathbb{R}$ defined by;

$$\nu(A) \;=\; E\left[\#\left\{t \in [0,1] : \Delta X_t \neq 0, \Delta X_t \in A\right\}\right],\; A \in \mathbb{R}$$

is called a Lévy measure of X_t ; $\nu(A)$ is the expected number, per unit of time, of jumps whose size belong to A. $\nu(A)$ is finite for any set A such that $0 \notin A$. If this is not true, the process would have an infinite number of jumps of finite size on $[0,T]$, see Tankov (2003).

Proposition 4. *Suppose X_t is a Compound Poisson process (CPP) with Lévy measure ν and $X_0 = 0$ then*

$$\begin{aligned} \psi(s) &= \int_{-\infty}^{\infty} \left[e^{isx} - 1\right] d\nu(x) \\ L\left[f(x)\right] &= \int_{-\infty}^{\infty} \left[f(x+y) - f(x)\right] d\nu(y). \end{aligned}$$

Proof. Let $\mu^{\#}$ denote the distribution of random variable J_k that is if $A \subset \mathbb{R}$ then $\nu(A) = P\{J_1 \in A\}$, we define $\nu = \lambda\mu^{\#}$, which is a measure on $\nu(\mathbb{R}) = \lambda$. Let

$$\phi(s) = E\left[e^{isJ_k}\right] = \int_{-\infty}^{\infty} e^{isx} d\mu^{\#}(x),$$

denote the characteristic function of a random variable with distribution F, then

$$E\left[e^{isY_t}\right] = \sum_{n=0}^{\infty} P\left[N(t) = n\right] E\left[e^{isY_t} \mid N_t = n\right].$$

Conditioning on $N(t) = n$, the distribution of Y_t is that of $J_1 + J_2 + ... + J_n$. Hence

$$\begin{aligned} E\left[e^{isY_t} \mid N_t = n\right] &= E\left[e^{is\left(J_1+J_2+....+J_{N_t}\right)} \mid N_t = n\right] \\ &= E\left[e^{is(J_1+J_2+....+J_n)}\right] \\ &= \prod_{j=1}^{n} E\left[e^{isJ_k}\right] \\ &= \left(E\left[e^{isJ_1}\right]\right)^n = \phi(s)^n. \end{aligned}$$

Since N_t is Poisson with mean rate λt;

$$\begin{aligned} E\left[e^{isY_t}\right] &= \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!}\phi(s)^n \\ &= \sum_{n=0}^{\infty} e^{-\lambda t}\frac{(\lambda t\phi(s))^n}{n!} \\ &= e^{-\lambda t}\sum_{n=0}^{\infty} \frac{(\lambda t\phi(s))^n}{n!} \\ &= e^{-\lambda t}e^{\lambda t\phi(s)} = e^{\lambda t(\phi(s)-1)}, \end{aligned}$$

but $\phi(s) = \int_{-\infty}^{\infty} e^{isx} d\mu^{\#}(x)$ then

$$\begin{aligned} E\left[e^{isY_t}\right] &= exp\left\{\lambda t\left(\phi(s) - 1\right)\right\} \\ &= exp\left\{\lambda t\left(\int_{-\infty}^{\infty}\left[e^{isx} - 1\right] d\mu^{\#}(x)\right)\right\}. \end{aligned}$$

The second last equality uses the fact that $\mu^{\#}$ is a probability measure. Finally we get;

$$\begin{aligned} E\left[e^{isY_t}\right] &= exp\left\{t\left(\int_{-\infty}^{\infty}\left[e^{isx} - 1\right] \lambda d\mu^{\#}(x)\right)\right\} \\ &= exp\left\{t\left(\int_{-\infty}^{\infty}\left[e^{isx} - 1\right] d\nu(x)\right)\right\}, \end{aligned} \tag{5.8.1}$$

since $\lambda\mu^{\#} = \nu$. The measure ν is called the Lévy *measure* for the process $(X_t)_{t\geq 0}$, ν is a positive measure on $\mathbb{R}$ but not a probability measure since $\int_{\mathbb{R}} d\nu(x) = \lambda \neq 1$.
We now consider the computation for the generator, L, which follows from the definition of the compound Poisson process. In a small time Δt, the probability that there is a jump is $\lambda\Delta t + o(\Delta t)$. Given that there is a jump the amount jumped is given by the distribution $\mu^{\#}$. In other words the probability that there is a jump in time Δt whose size is in (a, b) is given up to an error $o(\Delta t)$ by;

$$\lambda\Delta t\mu^{\#}(a, b) = \Delta t\nu(a, b),$$

since $\lambda\mu^{\#} = \nu$. Therefore;

$$\begin{aligned} E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] &= \left[1 - \lambda\Delta t\right] f(x) + \Delta t\int_{-\infty}^{\infty} f(x + y)d\nu(y) + o(\Delta t) \\ &= f(x) + \Delta t\int_{-\infty}^{\infty}\left[f(x + y) - f(x)\right] d\nu(y) + o(\Delta t). \end{aligned}$$

Taking the limit as $\Delta t \to 0$,

$$\lim_{\Delta t \to 0} \frac{E\left[f\left(X_{\Delta t}\right) \mid X_0 = x\right] - f\left(x\right)}{\Delta t} = \int_{-\infty}^{\infty} \left[f(x+y) - f\left(x\right)\right] d\nu(y) + \lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t},$$

where $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} \to 0$. Therefore;

$$L\left[f\left(x\right)\right] = \int_{-\infty}^{\infty} \left[f(x+y) - f\left(x\right)\right] d\nu(y).$$

□

5.9 The Gamma model

The $Gamma(a, b)$ process is a Lévy process with increment distribution:

$$X(t+1) - X(t) \sim Gamma(a, b)$$

that is has a distribution with density:

$$f(x) = \frac{1}{b^a \Gamma(a)} x^{a-1} exp\left(-\frac{x}{b}\right) \mathbf{1}_{\{x \geq 0\}}.$$

The expectation of the gamma density is given by $E\left[X\right] = ab$ and the characteristic function is, $\varphi_X(t) = (1 - ibt)^{-a}$.

5.10 Popular Models

In this section we present some popular models in mathematical finance from point of view of the Lévy processes, which we denote by X_t. We describe their characteristic functions and Lévy triplets.

Definition 29. (Lévy triplet) For every Lévy process, X_t there exist a vector of drift parameters, γ, a positive definite matrix of Brownian motion, $\boldsymbol{A}$ and a positive measure, ν that uniquely determines its distribution, the triplet $(\gamma, \boldsymbol{A}, \nu)$ is called a characteristic triplet or Lévy triplet of the process.

5.10.1 Black-Scholes

The most famous asset price model based on the Lévy process is that of Samuelson, Black and Scholes and Merton, see Papapantoleon (2008). The log returns are normally distributed with mean μ and variance σ^2 that is, the random variable $X_1 \sim N(\mu, \sigma^2)$ and its density is:

$$f_{X_1}(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}.$$

The characteristic function is

$$\varphi_{X_1}(s) = exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\}.$$

Using 5.4.1, the first moment is given by

$$\begin{aligned} E[X_1] &= \frac{1}{i}\left(\frac{\mathrm{d}\varphi_{X_1}(s)}{\mathrm{d}s} \mid_{s=0}\right) \\ &= \frac{1}{i}\frac{\mathrm{d}}{\mathrm{d}s}\left(exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\}\right) \mid_{s=0} \\ &= \frac{1}{i}\left(i\mu - \sigma^2 s\right) exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\} \mid_{s=0} \\ &= \mu. \end{aligned}$$

Using 5.4.1, the second moment is given by

$$\begin{aligned} E\left[X_1^2\right] &= \frac{1}{i^2}\left(\frac{\mathrm{d}^2\varphi_{X_1}(s)}{\mathrm{d}s^2} \mid_{s=0}\right) \\ &= -\frac{\mathrm{d}^2}{\mathrm{d}s^2}\left(exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\}\right) \mid_{s=0} \\ &= -\frac{\mathrm{d}}{\mathrm{d}s}\left(\left(i\mu - \sigma^2 s\right) exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\}\right) \mid_{s=0} \\ &= -\left(\left(i\mu - \sigma^2 s\right)^2 exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\} - \sigma^2 exp\left\{i\mu s - \frac{\sigma^2 s^2}{2}\right\}\right) \mid_{s=0} \\ &= -\left(i\mu\right)^2 + \sigma^2 \\ &= \mu^2 + \sigma^2. \end{aligned}$$

$E[X_1] = \mu$ and $var[X_1] = \sigma^2$,

$$\begin{aligned} var\left[X_1\right] &= E\left[X_1^2\right] - E^2\left[X_1\right] \\ &= \mu^2 + \sigma^2 - \mu^2 \\ &= \mu^2 + \sigma^2 - \mu^2 \\ &= \sigma^2 \end{aligned}$$

while kurtosis and skewness are: $skew[X_1] = 0$, $kurt[X_1] = 3$. The canonical decomposition of X is :

$$X_t = \mu t + \sigma W_t$$

and the Lévy triplet is $(\mu, \sigma^2, 0)$

5.10.2 Merton

Merton was one of the first to use a discontinuous price process to model asset returns, Papapantoleon (2008). The canonical decomposition of the driving process is:

$$X_t = \mu t + \sigma W_t + \sum_{k=1}^{N_t} J_k,$$

where $J_k \sim N(\mu_j, \sigma_j^2)$ $k = 1, 2, ...$hence the distribution of the jump size has density:

$$f_j(x) = \frac{1}{\sigma_j\sqrt{2\pi}} exp\left\{-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right\}.$$

The characteristic function of X_1 is

$$\varphi_{X_1}(s) = exp\left\{i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(e^{i\mu_j s - \frac{\sigma_j^2 s^2}{2}} - 1\right)\right\}.$$

The Lévy triplet is $(\mu, \sigma^2, \lambda f_j)$. The density of X_1 is not known in closed form, whilst the first two moments are: $E[X_1] = \mu + \lambda\mu_j$ which is computed as follows;

$$\begin{aligned}
E[X_1] &= \frac{1}{i}\left(\frac{\mathrm{d}\varphi_{X_1}(s)}{\mathrm{d}s} \mid_{s=0}\right) \\
&= \frac{1}{i}\frac{\mathrm{d}}{\mathrm{d}s}\left(exp\left\{i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(e^{i\mu_j s - \frac{\sigma_j^2 s^2}{2}} - 1\right)\right\}\right) \mid_{s=0} \\
&= \frac{1}{i}\left(i\mu - \sigma^2 s + \lambda\left[i\mu_j - \sigma_j^2 s\right] e^{i\mu_j s - \frac{\sigma_j^2 s^2}{2}}\right) exp\left\{i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(e^{i\mu_j s - \frac{\sigma_j^2 s^2}{2}} - 1\right)\right\} \mid_{s=0} \\
&= \mu + \lambda\mu_j
\end{aligned}$$

and $var[X_1] = \sigma^2 + \lambda\mu_j^2 + \lambda\sigma_j^2$.

5.10.3 Kou

Kou proposed a jump diffusion model similar to Merton's where the size of the jump is double exponentially distributed, Papapantoleon (2008). The canonical decomposition of the driving process is;

$$X_t = \mu t + \sigma W_t + \sum_{k=1}^{N_t} J_k,$$

where $J_k \sim Db\left(p, \theta_1, \theta_2\right) \quad k = 1, 2, ...$ hence the distribution of the jump size has density:

$$f_j(x) = p\theta_1 e^{-\theta_1 x} 1_{\{x>0\}} + (1-p)\,\theta_2 e^{\theta_2 x} 1_{\{x<0\}}.$$

The characteristic function of X_1 is :

$$\varphi_{X_1}(s) = exp\left\{i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(\frac{p\theta_1}{\theta_1 - is} + \frac{(1-p)\theta_2}{\theta_2 + is} - 1\right)\right\}.$$

The Lévy triplet is $(\mu, \sigma^2, \lambda f_j)$. The density of X_1 is not known in closed form and the first moment is given by:

$$\begin{aligned}
E[X_1] &= \frac{1}{i}\left(\frac{\mathrm{d}\varphi_{X_1}(s)}{\mathrm{d}s} \mid_{s=0}\right) \\
&= \frac{1}{i}\frac{\mathrm{d}}{\mathrm{d}s}\left(exp\left\{i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(\frac{p\theta_1}{\theta_1 - is} + \frac{(1-p)\theta_2}{\theta_2 + is} - 1\right)\right\}\right) \mid_{s=0} \\
&= \frac{1}{i}\left[i\mu - \sigma^2 s + \lambda i\left[\frac{p\theta_1}{(\theta_1 - is)^2} - \frac{(1-p)\,\theta_2}{(\theta_2 + is)^2}\right]\right] \\
&\times \quad exp\left[i\mu s - \frac{\sigma^2 s^2}{2} + \lambda\left(\frac{p\theta_1}{\theta_1 - is} + \frac{(1-p)\theta_2}{\theta_2 + is} - 1\right)\right] \mid_{s=0} \\
&= \frac{1}{i}\left(i\mu + \lambda i\left(\frac{p\theta_1}{\theta_1^2} - \frac{(1-p)\,\theta_2}{\theta_2^2}\right)\right) \\
&= \mu + \frac{\lambda p}{\theta_1} - \frac{\lambda\,(1-p)}{\theta_2}.
\end{aligned}$$

Thus

$$E[X_1] = \mu + \frac{\lambda p}{\theta_1} - \frac{\lambda(1-p)}{\theta_2}$$

and

$$Var[X_1] = \sigma^2 + \frac{\lambda p}{\theta_1^2} + \frac{\lambda(1-p)}{\theta_2^2},$$

also see Kou and Wang (2004).

Chapter 6

Some C++ Implementations

In the following sections we present computational work in C++. Implementation of basic financial models is followed by simulations. Finally derivative securities valuation is considered. A brief introduction to C++ is given in Appendix C.

6.1 Basic financial models

In this section we introduce basic financial models such as the accumulated and present value.

6.1.1 Accumulated Amount

In this section, we compute the accumulated amount, given the initial amount and an annual effective interest rate.

$$A(t) = A(0)(1+r)^t$$

where $A(t)$ is the accumulated amount, $A(0)$ is the initial amount, t is the time period and r is the annual effective interest rate, see chapter 6 Finan (2008).

```
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

// A function for accumulated amount
double AM(double const&t,double const &A_i, double
const&r,double const&i) {
double ACM=A_i*pow(1+r,t);
return ACM;
}
int main() {
double A_i,r,t;
for(int i=0;i<10;i++) {
cout<<"Accummulated Amount="<<AM(t,A_i,r,i)<<endl;
cout<<"Initial Amount:";
```

```
cin>>A_i;
cout<<"Interest rate:";
cin>>r;
cout<<"Time in years:";
cin>>t;};
}
```

Output for the Accumulated amount:

The code computes the accumulated amount or the future value of an investment given, the initial amount (present value) and time.

6.1.2 Present value function

In this section, we present a code for computing the present value of a given amount where we assume continuous compounding in the rate.

$$PV = Ae^{-rt}$$

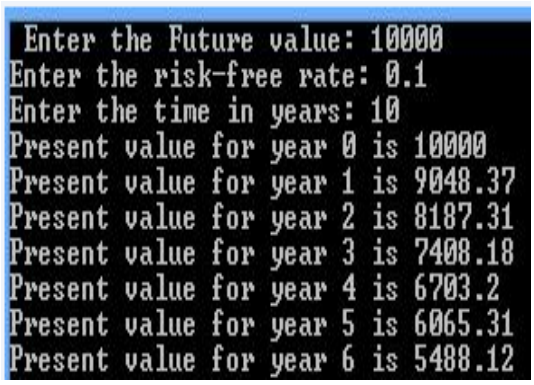
where A is a cash flow at time t, A is usually referred to as the future value, FV in the literature and r is the interest rate, see section 7 Finan (2008).

```
#include<iostream>//input and output
#include<cmath>// mathematics library
using namespace std;

double PV(double A,double r,int T){
  double PV_=A*exp(-r*T);
  return PV_;
}
int main(){
  double r,A, P;
int T;
  std::cout<<" Enter the Future value:";
```

```
  std::cin>>A;
  std::cout<<"Enter the risk-free rate:";
  std::cin>>r;
  std::cout<<"Enter the time in years:";
  std::cin>>T;
    for(int i=0;i<=T;i++){
  P=PV(A,r,i);
std::cout<<"Present Value "<<"for year"<<i<<":"<<P<<
   endl;}
    }
```

Output for computing the present value

An amount A is continuously discounted for a period of T years at a constant interest rate of r. The above code computes the present values for a given amount, A, interest rate, r and time $t = 0$ to T.



6.2 Random Number generation

In this section, we generate random numbers which we use in the subsequent sections.

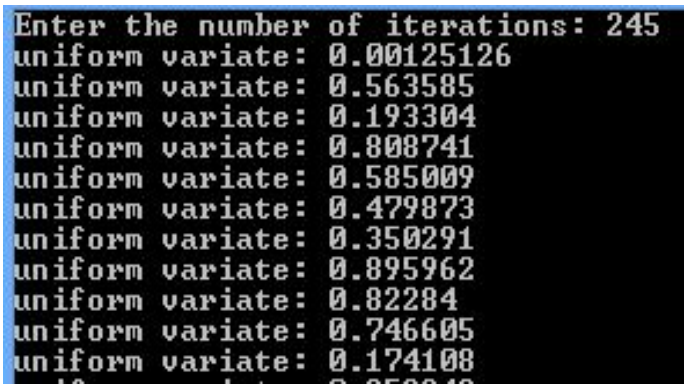
6.2.1 Generation of uniform random variates

We generate uniform $(0, 1)$ numbers from the code below using *Cstdlib*, see Halterman (2016). In this section, we generate uniform random number using the *rand*() function, see Ødegaard (2017).

```
#include<iostream>// input and output
#include<cstdlib>// standard c library and is
//important for random number generation
#include<cmath>// mathematics library
using namespace std;

double uniform(){
```

```
  double randmax=RAND_MAX;
  double x=rand()/randmax;
  return x;
  };

int main(){
  int n;
  double U;
  std::cout<<"Enter the number of iterations: ";
  std::cin>>n;
  for(int i=0;i<=n;i++){
    U=uniform();
    std::cout<<"uniform variate:"<<U<<endl;
    }
  }
```

Output: Uniform random variates



From the above algorithm, the interface will enable the user to specify the number of iterations so that the above output is produced.

6.3 Simulation

In this section we present codes for simulating the variables that are used in pricing of financial instruments.

6.3.1 Simulation: Exponential variates

We implement a code for generating exponential variates, which are important in the simulation of Poisson and gamma processes. We considered Haugh (2004) for the algorithm that we implemented in our code.

```
#include<iostream>//input and output
#include<cstdlib>//random number generation
```

```
#include<cmath>//mathematics library
using namespace std;

double uni(){double randmax=RAND_MAX;
double u=rand()/randmax;
return u;
}
double Exp(){double lambda;
double u1= uni();
double x=-(1/lambda)*log(u1);
return x;
}

int main(){
  int n;
double lambda,u1,y;
std::cout<<"Enter lambda:";
std::cin>>lambda;
std::cout<<"Enter the number of iterations:";
std::cin>>n;
for(int i=1;i<=n;i++) {
  u1=uni();
  y=Exp();
  //std::cout<<"Uniform variate: "<<uni()<<endl;
  std::cout<<"Exponential random variate:"<<y<<endl;
}
  }
```

Output: Exponential variates

```
Enter lambda:0.5
Enter the number of iterations:100
Exponential random variate:458.288
Exponential random variate:1.09815
Exponential random variate:1.25508
Exponential random variate:0.313616
Exponential random variate:0.355135
Exponential random variate:0.873321
Exponential random variate:0.937981
Exponential random variate:13.8184
Exponential random variate:11.043
```

6.3.2 Simulation: Gamma variates

In implementation of the Lévy processes or Lévy driven models such as the Variance Gamma model, we need gamma variates. Our Gamma simulation code is based on the algorithm from Haugh (2004).

```
#include<iostream>//input and output
#include<cmath>//mathematics library
#include<cstdlib>//random number generation
#include<vector>//vectors
using namespace std;

double unif(){
  double randmax=RAND_MAX;
  double y=rand()/randmax;
  return y;
}
 double Exp(){
  double lambda;
  double u=unif();
  double x=-(1/(lambda))*log(u);
  return x;
 }

 int main(){
  double lambda,u,v,x1;
  int n;
  double Gamma;
  std::cout<<"Number of iterations: ";
  std::cin>>n;
  std::cout<<"Enter  the value of Lambda:";
  std::cin>>lambda;
    for(int i=1;i<=n;i++){
    u=unif();
    x1=Exp();
    Gamma +=Exp();
  std::cout<<"Uniform variate is :"<<u<<endl;
    std::cout<<"Expo variate is :"<<x1<<endl;
    std::cout<<"Gamma variate is:"<<Gamma<<endl;}
  }
```

Output: Gamma variates

```
Number of iterations: 250
Enter  the value of Lambda:2
Gamma variate is :1313.47
Gamma variate is :1314.38
Gamma variate is :1314.93
Gamma variate is :1315.14
Gamma variate is :1316.81
Gamma variate is :1384.17
Gamma variate is :1384.25
Gamma variate is :1396.29
Gamma variate is :1467.18
Gamma variate is :1468.06
```

6.3.3 Simulation: Random Normal variates (Box-Muller)

In this section, we present a code for generating random normal variates from uniform variates that have been generated using a *rand*() function. The approximations used for the normal variates are found in Chapter 2, Glasserman (2013) and Chapter 2, London (2005). In this code the Box-Muller method has been used in the generation of a normal random variates.

```
#include<iostream>// input and output
#include<cstdlib>//random number generation
#include<cmath>//mathematics library
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264643;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}

int main(){double x;
int n;
std::cout<<"Please enter the number of iterations:";
  std::cin>>n;
for(int i=1;i<=n;i++){
  std::cout<<"Random uniform number is:";
  std::cout<<unif()<<endl;
  std::cout<<"Random Normal Variates:";
  std::cout<<rnorm()<<endl;
  }
}
```

Output: Random normal and uniform variates

```
Please enter the number of iterations:50
Random uniform number is:0.00125126
Random Normal Variates:1.00369
Random uniform number is:0.808741
Random Normal Variates:0.130602
Random uniform number is:0.350291
Random Normal Variates:-0.420497
Random uniform number is:0.746605
Random Normal Variates:-1.44859
Random uniform number is:0.710501
Random Normal Variates:1.0887
Random uniform number is:0.0149846
Random Normal Variates:1.6458
```

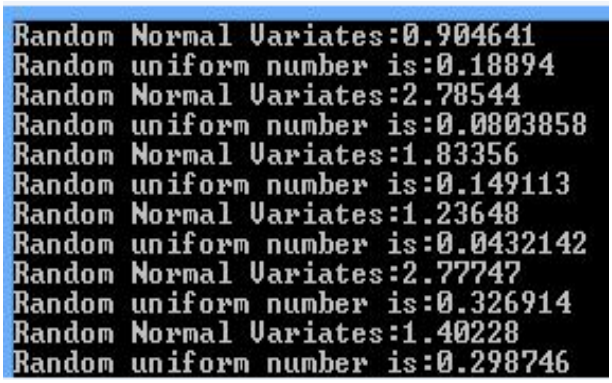
6.3.4 Simulation: Random Normal variates

Here we consider an improved Box-Muller method. Marsagalia and Bray modified the Box-Muller method with a goal of reducing computing time by avoiding the evaluation of the sine and cosine functions, see Chapter 2, Glasserman (2013).

```
#include<iostream>// input and output
#include<cstdlib>//random number generation
#include<cmath>//mathematics library
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double X,U1,U2,U,V,Y,Z1,Z2;
X=2;
while(X>1){
U1=unif();
 U2=unif();
 U=2*U1-1.0;
 V=2*U2-1.0;
 X=pow(U,2)+pow(V,2);}
 Y=sqrt(-2*log(X)/X);
 Z1=-U1*Y;
 Z2=U2*Y;
return Z1,Z2;
}

int main(){double x;
int n;
std::cout<<"Enter the number of iterations:";
  std::cin>>n;
for(int i=1;i<=n;i++){
```

```
  unif();
  rnorm();
  std::cout<<"Random uniform number is:";
  std::cout<<unif()<<endl;
  std::cout<<"Random Normal Variates:";
  std::cout<<rnorm()<<endl;
};
}
```

Output: Random Normal variates



6.4 Monte Carlo Valuation of Options

This section deals on stochastic modelling of asset prices and valuation of financial securities.

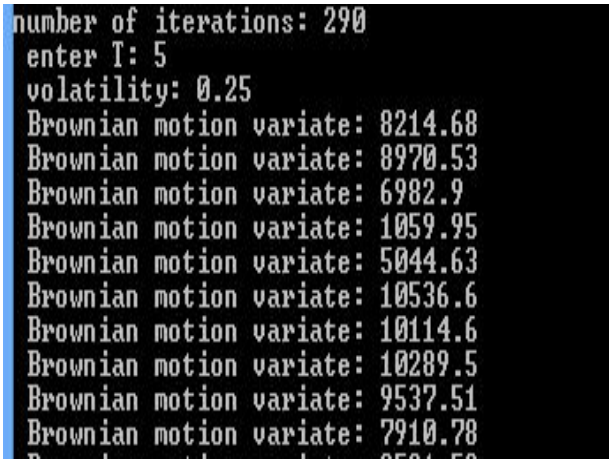
6.4.1 Simulation: Brownian Motion

Korn et al. (2010), provides an algorithm for simulation of a Brownian motion in chapter 4.

```
#include<iostream>//input and output
#include<cstdlib>//random number generation
#include<cmath>//mathematics
#include<vector>//vector

using namespace std;

double un(){
  double randmax=RAND_MAX;
  double p=rand()/randmax;
  return p;
}

double rnorm(){double un();
```

```
double U1,U2,PI,R,V,Z1,Z2;
 U1=un();
 U2=un();
 PI=3.14159265358979323846264643;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}



double Brownian(int N,double z,double sigma){
  int T;
  double Brow=sigma*sqrt(T/N)*z;
  return Brow;
}

int main(){
  double z,w,v,b,T,sigma;
     int N=10000;
  vector<double>V(N);
  vector<double>Z(N);
  vector<double>W(N);
  vector<double>t_i(N);
  vector<double>B(N);
  W[0]=0;
  std::cout<<"number of iterations: ";
  std::cin>>N;
  std::cout<<" enter T: ";
  std::cin>>T;
    std::cout<<" volatility: ";
  std::cin>>sigma;
  for(int i=1;i<=N;i++){
    V[i]=un();
    v=V[i];
    Z[i]=rnorm();
    z=Z[i];
    B[i]=Brownian(i,z,sigma);
    b=B[i];
    W[i]=W[i-1]+B[i];
  std::cout<<" Brownian motion variate: "<<W[i]<<endl;

  }}
```

Output: Simulation of Brownian Motion



6.4.2 Simulating stock price path in the Black Scholes model

We present a log normal asset model, in this case we consider a European option for illustrative purposes. The log normally distributed asset price is given by:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}$$

and

$$S_t = S_0 e^{\left(r - \frac{1}{2}\sigma^2\right)t + \sigma W_t}$$

where μ is the drift, r is the risk free rate, σ is the asset return volatility, t is the time to maturity and W_t is a Brownian motion, see Chapters 53-55, Finan (2011), Chapter 2, London (2005) and Broadie and Glasserman (1997). The code for simulating risk neutral asset prices is given below. The risk neutral concept has been covered in the literature, see from section 3.2 up to the end of the chapter 3:

```
#include<iostream>// input and output
#include<cstdlib>// standard c library and is
//important for random number generation
#include<cmath>// mathematics library
#include<vector>//vectors
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
```

```
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}
double Simulation(int i, double r, double b,double
   sigma){
  double  S_i,Z=rnorm();
  double t;
   S_i=exp((r-b-0.5*sigma*sigma)*t+sigma*Z*sqrt(t));
  return S_i;
  }

  int main(){
  double t=0.5;
  int n;
double b,r,sigma,X;
double c,p, Z,Pt,Cal, CalOpt,PutOpt;
double U1,U2;
vector<double>a(n);
vector<double>Call(n);
vector<double>Put(n);
std::cout<<" Dividend yield:";
      std::cin>>b;
      std::cout<<" Volatility:";
      std::cin>>sigma;
      std::cout<<"Risk-free rate:";
      std::cin>>r;
        std::cout<<"Number of iterations:";
          std::cin>>n;
        std::cout<<"Strike price:";
        std::cin>>X;
      std::cout<<"Current asset price: ";
      std::cin>>a[0];
  for(int i=1;i<=n;i++){
    U1=unif();
    U2=unif();
  Z=rnorm();
  a[i]=a[0]*Simulation(i,r,b,sigma);
    std::cout<<"Sim value: "<<a[i]<<endl;}
    }
```

Output: Simulated asset prices

The independently simulated asset prices below are used in the valuation of options. We used an initial asset price, $S_0 = 50$ to get the output below;

```
 Dividend yield:0.02
 Volatility:0.34
Risk-free rate:0.05
Number of iterations:200
Strike price:50
Current asset price: 50
Sim value: 50.8802
Sim value: 31.9988
Sim value: 53.5226
Sim value: 50.0673
Sim value: 41.8731
Sim value: 36.5941
Sim value: 45.0833
Sim value: 58.3688
Sim value: 48.8386
Sim value: 50.4015
Sim value: 44.2259
```

6.4.3 Option Valuation

In this subsection, we present a code for computing the options (Call and Put) payoffs from the simulated asset prices. We subsequently compute the option values based on these payoffs. The code for valuing call and put options is presented below:

```
#include<iostream>// input and output
#include<cstdlib>// standard c library and is
//important for random number generation
#include<cmath>// mathematics library
#include<vector>//vectors
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}
 double Simulation(int i, double r, double b,double
    sigma){
  double Z=rnorm();
  double t;
double S_i=exp((r-b-0.5*sigma*sigma)*t+sigma*Z*sqrt(t))
   ;
  return S_i;
```

```
}

  int main(){
  double t;
  int n;
double S_i;
double b;
double r;
double sigma;
double c,v,op,Z,Pt,Cal;
double U1,U2;
double X;
vector<double>a(n);
vector<double>Call(n);
vector<double>Put(n);
std::cout<<"Enter the number of iterations:";
std::cin>>n;
std::cout<<"Enter the continuous dividend yield:";
std::cin>>b;
std::cout<<"Enter the time to maturity (in years):";
std::cin>>t;
std::cout<<"Enter the risk free interest rate:";
std::cin>>r;
std::cout<<"Enter the volatility of asset retuns:";
std::cin>>sigma;
std::cout<<"Enter the Strike price:";
std::cin>>X;
std::cout<<"Enter the current asset price(Stock):";
std::cin>>a[0];
for(int i=1;i<=n;i++){
    U1=unif();
    U2=unif();
  Z=rnorm();
a[i]=a[0]*Simulation(i,r,b,sigma);
    Call[i]=max(0.0,a[i]-X);
    Put[i]=max(0.0,X-a[i]);
    Cal +=Call[i];
    Pt +=Put[i];
  std::cout<<"Simulated value:"<<a[i]<<endl;
  }
v=Cal*exp(-r*t)/n;
op=Pt*exp(-r*t)/n;
std::cout<<"The Call Option price is: "<<v<<endl;
std::cout<<"Put Option price is:"<<op<<endl;
}
```

Output: Call and Put Options valuation

The output above shows the call and put option values among other data.

6.4.4 Monte Carlo Option Valuation: Antithetic Variate

A number of techniques have been proposed in finance literature to improve the efficiency of crude Monte Carlo method. Control and Antithetic variate methods are among the procedures that improve the crude Monte Carlo method. For discussions on these methods, see Boyle (1977), Chapter 2, London (2005) and Chapter 57, Finan (2011) among other resources. Here we present a code for computing option values, both call and put.

```
#include<iostream>// input and output
#include<cstdlib>// standard c library and is
//important for random number generation
#include<cmath>// mathematics library
#include<vector>//vectors
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}
    double Simulation(int i, double r, double b,
double sigma){
```

```
  double Z=rnorm();
  double t;
double S_i=exp((r-b-0.5*sigma*sigma)*t+sigma*Z*sqrt(t))
   ;
  return S_i;
  }
    double Simulation1(int i, double r, double b,
double sigma){
  double Z1=-rnorm();
  double t;
double S1_i=exp((r-b-0.5*sigma*sigma)*t+sigma*Z1*sqrt(t
   ));
  return S1_i;
  }

int main(){
  double t=0.5;
  int n;
double S_i;
double S1_i;
double b;
double r;
double sigma;
double c,v,v1,op,op1,Z,Z1,Pt,Pt1,Cal, Cal1;
double U1,U2;
double X;
vector<double>a(n);
vector<double>a1(n);
vector<double>Call(n);
vector<double>Call1(n);
vector<double>Put(n);
vector<double>Put1(n);
std::cout<<"Enter the number of iterations:";
std::cin>>n;
std::cout<<"Enter the continuous dividend yield:";
std::cin>>b;
std::cout<<"Enter the risk free interest rate:";
std::cin>>r;
std::cout<<"Enter the volatility of asset retuns:";
std::cin>>sigma;
std::cout<<"Enter the Strike price:";
std::cin>>X;
std::cout<<"Enter the current asset price(Stock):";
std::cin>>a[0];
  for(int i=1;i<=n;i++){
    U1=unif();
    U2=unif();
  Z=rnorm();
  Z1=-rnorm();
```

```
    a[i]=a[0]*Simulation(i,r,b,sigma);
      a1[i]=a[0]*Simulation1(i,r,b,sigma);
      Call[i]=max(0.0,a[i]-X);
        Call1[i]=max(0.0,a1[i]-X);
      Put[i]=max(0.0,X-a[i]);
        Put1[i]=max(0.0,X-a1[i]);
      Cal +=Call[i];
      Pt +=Put[i];
      Cal1 +=Call1[i];
      Pt1 +=Put1[i];
    std::cout<<"Simulated value:"<<a[i]<<endl;
    //std::cout<<"Sim Antithetic value:"<<a1[i]<<endl;
    }
  v1=(Cal+Cal1)*exp(-r*t)/(2*n);
  op1=(Pt+Pt1)*exp(-r*t)/(2*n);
  v=Cal*exp(-r*t)/n;
  op=Pt*exp(-r*t)/n;
  std::cout<<"Antithetic Call Option price is: "<<v1<<
     endl;
  std::cout<<"Antithetic Put Option price is:"<<op1<<endl
     ;
  std::cout<<"The Call Option price is: "<<v<<endl;
  std::cout<<"Put Option price is:"<<op<<endl;
    }
```

Output: Option Valuation (Antithetic variate)

```
Enter the number of iterations: 11000
Enter time to maturity: 1
Enter the continuously compounded dividend yield: 0.025
Enter the risk-free interest rate: 0.045
Enter the volatility of asset returns: 0.27
Enter the current asset (Stock) price: 50
The Antithetic Call Option price is: 3.71276
The Antithetic Put Option price is:3.19246
The Call Option price is: 3.71062
Put Option price is:3.25378
```

The output above shows the option values for call and put options under the Antithetic variate and crude Monte Carlo methods.

6.4.5 Simulating Path Dependent prices

We present a log normal distribution for valuing path dependent options. The log normally distributed asset price is given by:

$$S_{i+1} = S_i e^{\mu\Delta t+\sigma\sqrt{\Delta t}z_{i+1}}$$

where: $\Delta t = \frac{T}{N}$, N is the number of time points on a given sample path and T is the time to maturity, see Chapter 2, London (2005), Chapters 4-5, Korn et al. (2010) and Chapters 53-55, Finan (2011).

```
#include<iostream>// input and output
#include<cstdlib>// standard c library and is
//important for random number generation
#include<cmath>// mathematics library
#include<vector>//vectors
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}
  double SimPath(int t,int n, double r, double b,
    double sigma){
  double S_t_n,Z=rnorm();
S_t_n=exp((r-b-0.5*sigma*sigma)*t/12+sigma*Z*sqrt(t/12)
   );
return S_t_n;
}

int main(){
  int t=12;// is the time to maturity of the option
  int n=5000;
double r,b,sigma,X;
double w,v,Z,Cal, Pt;
double U1,U2;
vector<double>a(n);
vector<double>g(n);
std::cout<<" Dividend yield:";
```

```
        std::cin>>b;
          std::cout<<"Volatility:";
          std::cin>>sigma;
          std::cout<<"Risk-free rate:";
          std::cin>>r;
        std::cout<<"Strike price:";
        std::cin>>X;
        std::cout<<"Current asset price: ";
            std::cin>>a[0];
            for ( int j=1;j<=n;j++){
    for(int i=1;i<=t;i++){
    U1=unif();
      U2=unif();
      Z=rnorm();
        a[i]=a[i-1]*SimPath(i,n,r,b,sigma);
    }
    g[j]=a[t-1]*SimPath(t,j,r,b,sigma);
    std::cout<<"Path dep Sim Terminal values:"<<g[j]<<endl;

        }
        }
```

Output: Simulating Dependent asset prices

```
Path dep Simulated value:51.7282
Path dep Simulated value:84.5362
Path dep Simulated value:50.9905
Path dep Simulated value:54.232
Path dep Simulated value:60.4272
```

6.5 Revenue based (Stochastic) loan

In this section, we present a code for generating and predicting future revenues that are used in computation of a stochastic loan. A stochastic loan enables the borrower to repay a random amount that is a function of the generated revenue on regular basis. That is the stochastic loan repayment amount, $h(t) = f(t, R(t))$, where $R(t)$ is revenue at time, t. This is in contrast to traditional loans where borrowers are expected to make fixed regular payments. For further discussion on the stochastic loan and model derivation, see Mazengera (2017b) and Mazengera (2017a).

6.5.1 Revenue Prediction

The empirical revenue model is assumed to be path dependent and is given by:

$$R_{t+1} = R_t e^{(\beta - \frac{1}{2}\sigma^2)t + \sigma Z(t)}$$

where β is the drift, σ is volatility and $Z(t)$ is a random normal variate, see Mazengera (2017a). In our pricing we consider a risk neutral revenue model and is given by:

$$R_{t+1} = R_t e^{(r-\frac{1}{2}\sigma^2)t+\sigma Z(t)}.$$

We present the code for the computation of R_{t+1};

```
#include<iostream>// input and output
#include<cstdlib>// standard c library random number
   generation
#include<cmath>// mathematics library
#include<vector>
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}

double Revenue(int t, double r,double z,double sigma){
  double R_t;
R_t=exp((r-0.5*sigma*sigma)*t/12+sigma*z*sqrt(t/12));
return R_t;
}

    int main(){
  int t=50;
double U1,U2,R_t,r,p,z,sigma;
vector<double>R(t);
vector<double>Z(t);
    std::cout<<"Revenue returns volatility:";
    std::cin>>sigma;
    std::cout<<"Risk-free rate:";
    std::cin>>r;
      std::cout<<"Initial Revenue: ";
      std::cin>>R[0];
for(int i=1;i<=t;i++){
U1=unif();
```

```
    U2=unif();
    Z[i]=rnorm();
z=Z[i];
     R[i]=R[i-1]*Revenue(i,r,z,sigma);
    std::cout<<"Simulated Revenue: "<<i<<"is"<<R[i]<<endl
       ;
}
    }
```

Output: Revenue Prediction

```
Revenue returns volatility: 0.55
Risk-free rate: 0.045
Initial Revenue: 50
Simulated Revenue: 1 is: 49.5592
Simulated Revenue: 2 is: 48.6894
Simulated Revenue: 3 is: 47.4131
Simulated Revenue: 4 is: 45.7632
Simulated Revenue: 5 is: 43.7815
Simulated Revenue: 6 is: 41.5163
Simulated Revenue: 7 is: 39.0212
```

The other extract of the predicted revenue is given below;

```
Simulated Revenue: 20 is: 47.783
Simulated Revenue: 21 is: 30.1986
Simulated Revenue: 22 is: 71.4057
Simulated Revenue: 23 is: 55.982
Simulated Revenue: 24 is: 13.2976
Simulated Revenue: 25 is: 10.4477
Simulated Revenue: 26 is: 3.49955
Simulated Revenue: 27 is: 4.25482
```

6.5.2 Computational valuation of a stochastic loan

In this section we present a code that is used in the computation of loan amount that can be advanced to a borrowing firm. The computation is based on the generated revenues, see Mazengera (2017b) and Mazengera (2017a). We solve the problem as a derivative pricing problem, where we compute the amount that a firm affords as a payoff function of the repayment amount. In this case we consider

$$Payoff = max\{0, f(t, R(t))\}$$

for all $t > 0$ where: $h(t) = f(t, R(t))$, is the loan repayment amount at time t. After computing the payoffs, we discount and sum them up. Finally we perform thousands of iterations and find the expected value. The code is presented below:

```
#include<iostream>// input and output
#include<cstdlib>// standard c library random number
   generation
```

```
#include<cmath>// mathematics library
#include<vector>
using namespace std;

double unif(){double randmax= RAND_MAX;
double x=rand()/randmax;
return x;
}
double rnorm(){double unif();
double U1,U2,PI,R,V,Z1,Z2;
 U1=unif();
 U2=unif();
 PI=3.14159265358979323846264 3;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}

double Revenue(int t,int n, double r,double z,double
   sigma){
  double R_t_n;
R_t_n=exp((r-0.5*sigma*sigma)*t/12+sigma*z*sqrt(t/12));
return R_t_n;
}

    int main(){
  int n=1000,t=24;
double U1,U2,H,H2,R_t,Rev0,Rev1,r,p,z,sigma;
vector<double>Rev(n);
vector<double>h(n);
vector<double>h1(n);
vector<double>H1(n);
vector<double>R(n);
vector<double>Z(n);
double a=0.15;//proportion of revenue used in loan
//repayment
    std::cout<<"Revenue returns volatility:";
    std::cin>>sigma;
    std::cout<<"Risk-free rate:";
    std::cin>>r;
      std::cout<<"Initial Revenue: ";
      std::cin>>R[0];
        for(int j=1;j<=n;j++){
for(int i=1;i<=t;i++){
U1=unif();
  U2=unif();
  Z[i]=rnorm();
```

```
z=Z[i];
  R[i]=R[i-1]*Revenue(i,n,r,z,sigma);
    h[i]=max(0.0,a*R[i]);
    h1[i]=exp(-r*i/12)*h[i];
std::cout<<"Simulated Revenue: "<<i<<"is"<<R[i]<<endl;
std::cout<<"Stochastic repay amount: "<<i<<"is"<<h[i]<<
   endl;
std::cout<<"Discounted repay amount: "<<i<<"is"<<h1[i
   ]<<endl;
  Rev[j]+=R[i];
  Rev1+=Rev[j];
  H1[j]+=h1[i];
  H2+=H1[j];
  }
  H=H2/n;
  Rev0=Rev1/n;
  std::cout<<" Loan amount: "<<j<<"is"<<H1[j]<<endl;
  std::cout<<"Total Revenue: "<<"is"<<Rev1<<endl;
  std::cout<<"Avarage  Revenue: "<<"is"<<Rev0<<endl;
  std::cout<<"Avarage Loan Amount: "<<"is"<<H<<endl;
}
  }
```

Input Data: Stochastic loan amount

```
Revenue returns volatility: 0.45
Risk-free rate: 0.10
Initial Revenue: 50
```

The above data has been entered in addition to $n = 2000$ iterations and $t = 24$ path dependent iterations.

Output: Stochastic loan amount

```
Total Revenue: is3.66196e+007
Avarage  Revenue: is18309.8
Avarage Loan Amount: is2520.5
 Loan amount: 1999is244.792
Total Revenue: is3.66375e+007
Avarage  Revenue: is18318.8
Avarage Loan Amount: is2521.73
 Loan amount: 2000is183.959
Total Revenue: is3.66545e+007
Avarage  Revenue: is18327.2
Avarage Loan Amount: is2522.9
```

We performed $n = 2000$ iterations and $t = 24$ path dependent iterations. The changes in average loan amount were no longer significant starting from 1986 th iteration right up to

the 2000^{th}. The loan amount is 2500 and after 14 additional iterations the loan amount is 2522.9. This shows that our iterations have converged to the average loan amount. Therefore given revenue returns volatility of 45%, risk free interest rate of 10%, initial revenue of 50 and the proportion of revenue that goes towards loan repayment, $\alpha = 0.15$, based on 24 revenue periods the loan amount that can be advanced to a firm is 2500.

6.6 Lévy Processes Simulation

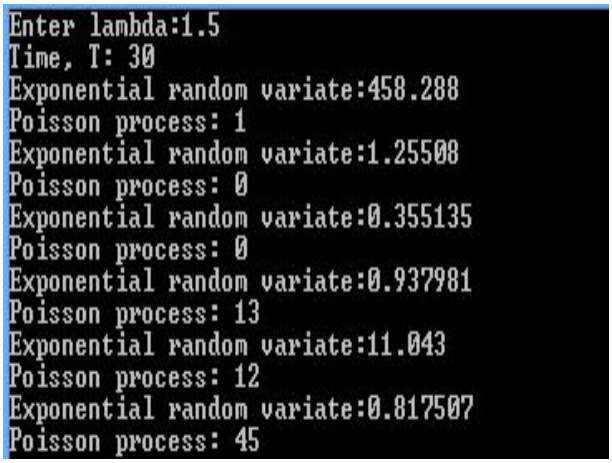
We consider the simulation of finite Lévy processes, see Korn et al. (2010) for the general algorithms.

6.6.1 Simulation: Poisson processes

```
#include<iostream>//input and ouput
#include<cstdlib>//random number generation
#include<cmath>//mathematics library
#include<vector>
using namespace std;

double uni(){double randmax=RAND_MAX;
double u=rand()/randmax;
return u;
}

double Exp(){double lambda;
double u= uni();
double x=-(1/lambda)*log(u);
return x;
}



int Pois(){
    int T;
    double R,r,s,u,uni(),Exp();
 vector<double>X(T);

  X[0]=0;
  R=0;
  r=0;
  while (R<T){
      u=uni();
      s=Exp();
      r=R;
  return  R=R+s;
  }
  if (R>T){
```

```
      R=T;
      X[R]=X[r];
      return X[r];
      }
    else {
        u=uni();
      X[R]=X[r]+1;
    return X[R];
    }
}


int main(){
  int T;
  vector<double>X(T);
  vector<double>s(T);
  vector<double>u(T);
  double r,R,lambda,unif();

std::cout<<"Enter lambda:";
std::cin>>lambda;
std::cout<<"Time, T: ";
std::cin>>T;

for(int t=1;t<=T;t++){
  u[t]=uni();
  s[t]=Exp();

std::cout<<"Exponential random variate:"<<s[t]<<endl;
std::cout<<"u: "<<u[t]<<endl;
std::cout<<"Poisson process: "<<Pois()<<endl;
}

}
```

Output: Poisson simulation



6.6.2 Simulation: Compound Poisson processes

In this section, we simulate a compound Poisson process. The algorithm used is found in Korn et al. (2010).

```
#include<iostream>//input and output
#include<cstdlib>//random number generation
#include<cmath>//mathematics library
#include<vector>// vector

using namespace std;

double unif(){double randmax=RAND_MAX;
double u=rand()/randmax;
return u;
}

double Exp(){double lambda;
double u= unif();
double a=-(1/lambda)*log(u);
return a;
}

double Comp_Pois(){
    int T;
    double R,r,s,u,unif(),Exp();
 vector<double>X(T);

  X[0]=0;
  R=0;
  r=0;
  while (R<T){
      u=unif();
```

```
        s=Exp();
        r=R;
    return  R=R+s;
    }
    if (R>T){
      R=T;
      X[R]=X[r];
      return X[r];
      }
    else {
        u=unif();
        X[R]=X[r]+u;
      return X[R];
    }
}

int main(){
  int T;
  vector<double>X(T);
  double u,s,r,R,lambda,unif();
  std::cout<<"Time, T: ";
  std::cin>>T;
  std::cout<<"lambda: ";
  std::cin>>lambda;
  for (int t=1;t<=T;t++){
      u=unif();
      s=Exp();
  std::cout<<"R : "<<R<<endl;
//std::cout<<"r: "<<r<<endl;
std::cout<<"s: "<<Exp()<<endl;
//std::cout<<"u: "<<unif()<<endl;
std::cout<<"Compound Poisson process: "<<Comp_Pois()<<
   endl;
}
}
```

Output: Compound Poisson process

```
Time, T: 50
lambda: 1.5
Compound Poisson process: 2.91613
Compound Poisson process: 1.25508
Compound Poisson process: 0.217636
Compound Poisson process: 0.873321
Compound Poisson process: 2.31872
Compound Poisson process: 11.043
Compound Poisson process: 0.0695684
Compound Poisson process: 45.0671
Compound Poisson process: 1.67181
Compound Poisson process: 0.829151
Compound Poisson process: 1.20166
Compound Poisson process: 8.7326
Compound Poisson process: 0.815035
Compound Poisson process: 0.36448
```

6.6.3 Simulation: Lévy-Jump process (Finite process)

```
#include<iostream>//input and output
#include<cstdlib>//random number generation
#include<cmath>//mathematics
#include<vector>//vector

using namespace std;

double un(){
  double randmax=RAND_MAX;
  double p=rand()/randmax;
  return p;
}

double rnorm(){double un();
double U1,U2,PI,R,V,Z1,Z2;
 U1=un();
 U2=un();
 PI=3.14159265358979323846264643;
 R=-2*log(U1);
 V=2*PI*U2;
 Z1=sqrt(R)*cos(V);
 Z2=sqrt(R)*sin(V);
return Z1,Z2;
}



double Brownian(int N,double z,double sigma){
  int T;
  double Brow=sigma*sqrt(T/N)*z;
  return Brow;
}
```

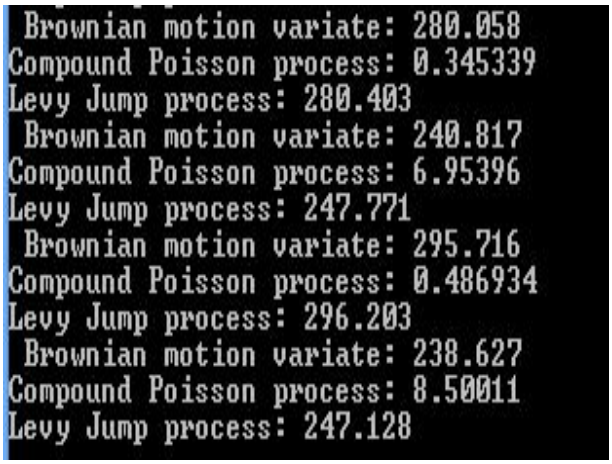
```
double Exp(){double lambda;
double u= un();
double x=-(1/lambda)*log(u);
return x;
};

double Comp_Pois(int T,double R ){
    double r,s,u,un(),Exp();
 vector<double>X(T);

  X[0]=0;
  R=0;
  r=0;
  while (R<T){
      u=un();
      s=Exp();
      r=R;
  return  R=R+s;
  }
  if (R>T){
    R=T;
    X[R]=X[r];
    return X[r];
    }
  else {
      u=un();
      X[R]=X[r]+u;
    return X[R];
  }
}



int main(){
  double z,v,b,T,sigma,R;
     int N;
  vector<double>Z(N);
  vector<double>W(N);
  vector<double>B(N);
  vector<double>u(N);
  vector<double>s(N);
  vector<double>cmp(N);
  vector<double>L(N);
  W[0]=0;
  std::cout<<"number of iterations: ";
  std::cin>>N;
  std::cout<<" enter T: ";
```

```
    std::cin>>T;
      std::cout<<" volatility: ";
    std::cin>>sigma;

        for(int i=1;i<=N;i++){
        u[i]=un();
        s[i]=Exp();
      v=u[i];
      Z[i]=rnorm();
      z=Z[i];
      cmp[i]=Comp_Pois(T,R);
      B[i]=Brownian(i,z,sigma);
      W[i]=W[i-1]+B[i];
      L[i]=W[i]+cmp[i];

  std::cout<<"normal variate: "<<z<<endl;
  std::cout<<" Brownian motion variate: "<<W[i]<<endl;
  std::cout<<"Exponential random variate:"<<s[i]<<endl;
  std::cout<<"u: "<<u[i]<<endl;
  std::cout<<"Compound Poisson process: "<<cmp[i]<<endl;
  std::cout<<"Lévy Jump process: "<<L[i]<<endl;
  }
  };
```

Output 1: Lévy Jump process (finite)

```
number of iterations: 80
 enter T: 2
 volatility: 0.25
 Brownian motion variate: -649.263
Compound Poisson process: 1.25508
Levy Jump process: -648.008
 Brownian motion variate: -818.779
Compound Poisson process: 0.873321
Levy Jump process: -817.906
 Brownian motion variate: -786.617
Compound Poisson process: 11.043
Levy Jump process: -775.574
 Brownian motion variate: -776.856
Compound Poisson process: 45.0671
Levy Jump process: -731.789
```

Output 2: Lévy Jump process (finite)



6.7 Finite Difference Computations

In this section we compute value of an option using the Black-Scholes PDE. We implemented the code from Ødegaard (2017), by breaking down the code into several codes of several functions, which is somehow different from the approach taken by Ødegaard (2017). We come up with small code fragments as a build up for the implementation. At each stage we program some useful functions and then merge all the codes together to evaluate the option price given the terminal value conditions. In this section we computationally value f, the forward option price:

$$u_{i,j} = \tilde{p_u} u_{i+1,j+1} + \tilde{p_m} u_{i+1,j} + \tilde{p_d} u_{i+1,j-1}, \tag{6.7.1}$$

where:

$$\begin{aligned} \tilde{p_u} &= \frac{\Delta t \sigma^2}{2\left(\Delta x\right)^2} + \frac{\mu \Delta t}{2\Delta x}, \\ \tilde{p_m} &= 1 - \frac{\Delta t \sigma^2}{\left(\Delta x\right)^2}, \\ \tilde{p_d} &= \frac{\Delta t \sigma^2}{2\left(\Delta x\right)^2} - \frac{\mu \Delta t}{2\Delta x}. \end{aligned}$$

Lets note that;

$$\tilde{p_u} + \tilde{p_m} + \tilde{p_d} = 1.$$

If we substitute the present value of the option $f_{ij} = e^{-r(T-t)} u_{ij}$ into the equation 6.7.1, we arrive at the backward relationship;

$$f_{ij} = e^{-r\Delta t} \left(\tilde{p_u} f_{i+1,j+1} + \tilde{p_m} f_{i+1,j} + \tilde{p_d} f_{i+1,j-1} \right),$$

see section 4.4 of the literature.

6.7.1 Computing values of the underlying(S) from Delta

In this section we compute the values of delta and the underlying asset, S. Delta S is given by the following approximation:

$$\Delta S = 2.0 * \frac{S}{M}$$

where: S, is the price of the underlying asset and M is the number of steps

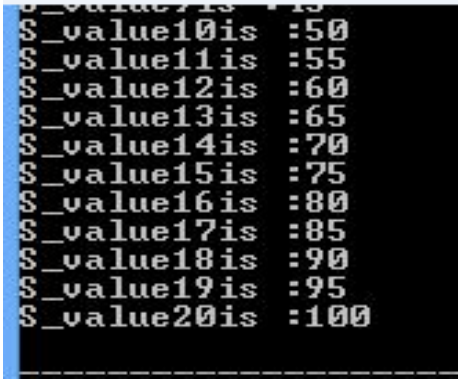
$$S_m = m * \Delta S = m * Delta_S$$

where: $0 \leq m \leq M$.

```
#include<cmath>//mathematics library
#include<vector>//vectors
#include<iostream>// input and output
using namespace std;// the above is part of

double delta(double S, int M){
double delta_S=2.0*S/M;
return delta_S;
};

int main(){
int M=10;
double S=50;
vector<double>b(M);
for (int m=1;m<=M;m++){
b[m]=2.0*S/m;
cout<<"value of b"<<m<<"=:"<<b[m]<<endl;
cout<<" delta_S:"<<m<< "is: " <<delta(S,m)<<endl;
};
}
```

Output for values of S

The above code gives the output below:



6.7.2 Computing Delta(t)

The change in time for the PDE is approximated by $delta(t)$ and is given by:

$$\Delta t = \frac{time}{N}$$

where N is the number of steps. The code for computing the above function is given below:

```
#include<iostream>// input and output
#include<cmath>//mathematics library
#include<vector>// vectors
#include<algorithm>//max function
using namespace std;


  double Delta(double time,int N){
    double delta=time/N;
    return delta;
    }

int main(){
  int N=11;
  double time=0.4167;
for(int i=N-1;i>0;i--){
Delta(time,i);
cout<<"value of delta_"<<i<<"is:"<<Delta(time,i)<<endl;
      }
       }
```

Output (Delta(t) computation)

We have assumed 11 steps that is $N = 11$, in our computation.

```
value of delta_10is :0.04167
value of delta_9is :0.0463
value of delta_8is :0.0520875
value of delta_7is :0.0595286
value of delta_6is :0.06945
value of delta_5is :0.08334
value of delta_4is :0.104175
value of delta_3is :0.1389
value of delta_2is :0.20835
value of delta_1is :0.4167
```

6.7.3 Computing the weights (probabilities) of the PDE

In this section, we compute the probabilities or weights of the PDE equation and we have denoted them $\tilde{p_u}$, $\tilde{p_m}$ and $\tilde{p_d}$ above. In our code we will denote them by the letters, $a[j]$, $b[j]$ and $c[j]$ respectively. We present a code for computing these weights below:

```
#include<iostream>// input and output
#include<cmath>//mathematics library
#include<vector>// vectors
#include<algorithm>//max function
using namespace std;

double A(double r,double sigma, double time,
int M,int N){
  double Delta=time/N;
  double sigma_sqr=pow(sigma,2);
  double r2=Delta/(1.0+r*Delta);
double  z=r2*0.5*M*(-r+sigma_sqr*M);
  return z;
  }
 double B(double r1,double sigma,double time,double r,
int N,int M ){
  double  Delta=time/N;
    double sigma_sqr=pow(sigma,2);
    r1=1.0/(1.0+r*Delta);
    double B_=r1*(1.0-sigma_sqr*M*M*Delta);
    return B_;
  }

  double C(double r, double time,double sigma,int N,
int M){
    double Delta=time/N;
      double sigma_sqr=pow(sigma,2);
double  r2=Delta/(1.0+r*Delta);
  double C_=r2*0.5*M*(r+sigma_sqr*M);
  return C_;
  }


 int main(){
  int N=11;
  double X=50;
  double time=0.4167;
  double S=50;
  int M=10;
   double r=0.1;
   double r1;
   double sigma=0.4;
    double r2;
    vector<double>a(M);
    vector<double>b(M);
    vector<double>c(M);
    for(int j=1;j<=M;j++){
      a[j]=A(r,sigma,time,j,N);
      b[j]=B(r1,sigma,time,r,N,j);
```

```
        c[j]=C(r,time,sigma,N,j);
cout<<"value of a "<<j<<"is :"<<a[j]<<endl;
cout<<"value of b"<<j<<"is :"<<b[j]<<endl;
cout<<"value of c"<<j<<"is :"<<c[j]<<endl;
        }
}
```

Output for the weights (Probabilities)

For the computed weights to be equated to probabilities, their sum must equal to one and they must be non-negative. At each j^{th} level all the computed weights should fulfill these conditions. This can be verified from the output below;

```
value of c2is :0.0158503
value of a 3is :0.0215111
value of b3is :0.941882
value of c3is :0.0328328
value of a 4is :0.040758
value of b4is :0.899615
value of c4is :0.0558535
value of a 5is :0.066043
value of b5is :0.845271
value of c5is :0.0849124
value of a 6is :0.0973662
value of b6is :0.77885
value of c6is :0.12001
value of a 7is :0.134728
value of b7is :0.700353
value of c7is :0.161145
value of a 8is :0.178127
value of b8is :0.60978
value of c8is :0.208318
value of a 9is :0.227565
value of b9is :0.507131
value of c9is :0.26153
```

6.7.4 Calculation of the Option payoff

In this section, we present a code for computing the payoffs, both the call and put payoff.

$$Payoff_{Call} = max\left(0.0, S_j - X\right),$$

$$Payoff_{Put} = max\left(0.0, X - S_j\right),$$

where X is the strike price and S_j is the asset price at time j.

```
#include<iostream>// input and output
#include<cmath>//mathematics library
#include<vector>// vectors
#include<algorithm>//max function
using namespace std;

double Delta(double time,int N){
    double delta=time/N;
    return delta;
    }
```

```
        double delta_S(double S,int M){
          double delta_s=2.0*S/M;
          return delta_s;
        }

      int main(){
       double X=50;
       double S=50;
       int M=10;
         vector<double>S_values(M);
      vector<double>p_next(M);
         vector<double>f_next(M);
       for(int j=1;j<=M;j++){
         S_values[j]=j*delta_S(S,M);
           f_next[j]=max(0.0,S_values[j]-X);
           p_next[j]=max(0.0,X-S_values[j]);
    cout<<"S_value"<<j<<"is :"<<S_values[j]<<endl;
    cout<<"call option pay-off"<<"is :"<<f_next[j]<<endl;
    cout<<"put option pay-off"<<"is :"<<p_next[j]<<endl;
        }
            }
```

Output for the payoff functions

In our code we assumed a strike price, $X = 50$.

```
put option pay-offis :30
S_value3is :30
call option pay-offis :0
put option pay-offis :20
S_value4is :40
call option pay-offis :0
put option pay-offis :10
S_value5is :50
call option pay-offis :0
put option pay-offis :0
S_value6is :60
call option pay-offis :10
put option pay-offis :0
S_value7is :70
call option pay-offis :20
put option pay-offis :0
```

6.7.5 Computing the option value

In this section, we merge all the code fragments and compute the option value, f and is given by:

$$f_{ij} = e^{-r\Delta t}\left(\tilde{p_u} f_{i+1,j+1} + \tilde{p_m} f_{i+1,j} + \tilde{p_d} f_{i+1,j-1}\right).$$

```
#include<iostream>// input and output
#include<cmath>//mathematics library
#include<vector>// vectors
#include<algorithm>//max function
using namespace std;

double A(double r,double sigma, double time, int M,
int N){
  double Delta=time/N;
  double sigma_sqr=pow(sigma,2);
  double r2=Delta/(1.0+r*Delta);
double  z=r2*0.5*M*(-r+sigma_sqr*M);
  return z;
  }
  double B(double r1,double sigma,double time,double r,
int N,int M ){
  double  Delta=time/N;
    double sigma_sqr=pow(sigma,2);
    r1=1.0/(1.0+r*Delta);
    double B_=r1*(1.0-sigma_sqr*M*M*Delta);
    return B_;
  }
  double C(double r, double time,double sigma,int N,
int M){
    double Delta=time/N;
      double sigma_sqr=pow(sigma,2);
double  r2=Delta/(1.0+r*Delta);
  double C_=r2*0.5*M*(r+sigma_sqr*M);
  return C_;
  }
  double Delta(double time,int N){
    double delta=time/N;
    return delta;
    }
    double delta_S(double S,int M){
      double delta_s=2.0*S/M;
      return delta_s;
    }

 int main(){
  int N=11;
  double X=50;
  double time=0.4167;
  double S=50;
  int M=10;
   double r=0.1;
   double r1;
   double sigma=0.4;
    double r2;
```

```
      vector<double>a(M);
      vector<double>S_values(M);
      vector<double>b(M);
      vector<double>c(M);
      vector<double>p_next(M);
      vector<double>f_next(M);
      vector<double>f(M);
      vector<double>f1(M);
        f[M]=0;
      for(int j=1;j<=M;j++){
        a[j]=A(r,sigma,time,j,N);
        S_values[j]=j*delta_S(S,M);
        b[j]=B(r1,sigma,time,r,N,j);
        c[j]=C(r,time,sigma,N,j);
        f_next[j]=max(0.0,S_values[j]-X);
        p_next[j]=max(0.0,X-S_values[j]);
   f[j]=a[j]*f_next[j-1]+b[j]*f_next[j]+c[j]*f_next[j
      +1];
   f1[j]=a[j]*p_next[j-1]+b[j]*p_next[j]+c[j]*p_next[j
      +1];
        f_next[j]=f[j];
cout<<"value of a "<<j<<"is :"<<a[j]<<endl;
cout<<"value of b"<<j<<"is :"<<b[j]<<endl;
cout<<"value of c"<<j<<"is :"<<c[j]<<endl;
cout<<"call option pay-off"<<"is :"<<f_next[j]<<endl;
cout<<"put option pay-off"<<"is :"<<p_next[j]<<endl;
cout<<"value of the call option"<<j<<"is :"<<f1[j]<<
   endl;
cout<<"value of the put option"<<j<<"is :"<<f[j]<<endl;

}
for(int i=N-1;i>0;i--){Delta(time,i);
   b[i]=B(r1,sigma,time,r,i,M);
   a[i]=A(r,sigma,time,M,i);
    c[i]=C(r,time,sigma,i,M);
cout<<"value of delta_"<<i<<"is :"<<Delta(time,i)<<endl
   ;
       cout<<"value of b"<<i<<"is :"<<b[i]<<endl;
       cout<<"value of c"<<i<<"is :"<<c[i]<<endl;
        }




}
```

Chapter 7

Conclusion

In most theoretical and practical applications closed form solutions to stochastic models do not exist. Computational methods come in handy and in such processes software capability is at test. Financial institutions have relied heavily on computational methods for asset pricing. As more stochastic models are developed for pricing purposes so should be the computational effort. We computationally valued assets and derivatives using selected stochastic models. In computational work codes were developed based on algorithms from referenced sources. Lévy-driven models have also been computationally implemented. Computational work accompanied theoretical exposition of these models.

The work will help in enriching the stochastic models literature, with emphasis on pricing where martingales and risk neutral pricing have been given an attention. The work links practitioners and academics since practitioners are mostly interested in the practical aspects of pricing which we have addressed through computational work after theoretical presentation of the models.

In future, it will be interesting to dig deeper theoretically into Lévy-driven models. Considering computational implementation of more complicated finite difference methods such as the Implicit and Crank-Nicolson for pricing purposes especially for advanced Lévy-driven models will also be a great thing.

Appendix A

Conditional Expectation

Let $X_1, X_2, X_3, ...$ be random variable which we think of as a time series with data arriving one at a time. At time n, we have viewed values of $X_1, X_2, X_3, ...X_n$. If Y is another variable, then $E(Y \mid X_1, X_2, X_3, ...X_n)$ is the best guess of Y given $X_1, X_2, X_3, ...X_n$. We assume that Y is an integrable random variable which means that $E[Y] < \infty$, we let $\mathscr{F}_n$ denote the information contained in $X_1, X_2, X_3, ...X_n$ and we write $E[Y \mid \mathscr{F}_n]$ in place of $E[Y \mid X_1, X_2, X_3, ...X_n]$. We view $\mathscr{F}_0$ as no information, if we have no information, the best guess is the expected value. In other words

$$E[Y \mid \mathscr{F}_0] \;=\; E[Y].$$

The conditional expectation $E[Y \mid \mathscr{F}_n]$ should only use the information available at time n, in other words it should be a function of $X_1, X_2, X_3, ...X_n$

$$E[Y \mid \mathscr{F}_n] \;=\; \psi(X_1, X_2, X_3, ...X_n)$$

we say that $E[Y \mid \mathscr{F}_n]$ is $\mathscr{F}_n$ measurable. We assume that the random variables $Y, X_1, X_2, X_3, ...$ are defined on a probability space $(\Omega, \mathscr{F}, P)$. Here $\mathscr{F}$ is a $\sigma - field$ of subsets of Ω that is a collection of subsets satisfying

- $\phi \in \mathscr{F}$;
- $A \in \mathscr{F}$ implies $\Omega \backslash A \in \mathscr{F}$;
- $A_1, A_2, ... \in \mathscr{F}$ implies $\bigcup_{n=1}^{\infty} A_n \in \mathscr{F}$.

The information $\mathscr{F}_n$ is the smallest sub $\sigma - field$ G of $\mathscr{F}$ such that $X_1, X_2, X_3, ...X_n$ are G measurable, this means that for all $t \in \mathbb{R}$, the event $\{X_j \leq t\} \in \mathscr{F}$. The "no information " $\sigma - algebra$ $\mathscr{F}_0$ is the trivial $\sigma - algebra$ containing only ϕ and Ω.

Definition 30. The conditional expectation $E[Y \mid \mathscr{F}_n]$ is a unique random variable satisfying the following:

- $E[Y \mid \mathscr{F}_n]$ is $\mathscr{F}_n$ measurable;
- For every $\mathscr{F}_n$ measurable event A, $E[E[Y \mid \mathscr{F}_n] 1_A] = E[Y 1_A]$.

Suppose $(\Omega, \mathscr{F}, P)$ is a probability space and Y is an integrable random variable and further suppose that G is a sub $\sigma-$ algebra of $\mathscr{F}$, then $E[Y \mid G]$ is defined to be the unique G measurable random variable such that if $A \in G$, $E[Y 1_A] = E[E[Y \mid G] 1_A]$.

Proposition 5. *Suppose $X_1, X_2, ...$ is a sequence of random variables and $\mathscr{F}_n$ denotes the information at time n. The conditional expectation $E[Y \mid \mathscr{F}_n]$ satisfies the following properties*

- If Y is $\mathscr{F}_n$ measurable then $E[Y \mid \mathscr{F}_n] = Y$;
- If A is an $\mathscr{F}_n$ measurable event then $E[E[Y \mid \mathscr{F}_n] 1_A] = E[Y 1_A]$ and in particular we have $E[E[Y \mid \mathscr{F}_n]] = E[Y]$.

Properties of Conditional Expectation

- Suppose $X_1, X_2, ...$ are independent of Y , then $\mathscr{F}_n$ contains no useful information about Y and hence

$$E[Y \mid \mathscr{F}_n] \quad = \quad E[Y]. \tag{A.0.1}$$

- Linearity, if Y and Z are random variables and a, b are constants, then

$$E[aY + bZ \mid \mathscr{F}_n] \quad = \quad aE[Y \mid \mathscr{F}_n] + bE[Z \mid \mathscr{F}_n].$$

- Projection or Tower Property, If $m < n$, then

$$E[E[Y \mid \mathscr{F}_n] \mid \mathscr{F}_m] \quad = \quad E[Y \mid \mathscr{F}_m]. \tag{A.0.2}$$

- If Z is an $\mathscr{F}_n$ measurable random variable, then when conditioning with respect to $\mathscr{F}_n$, Z acts like a constant

$$E[YZ \mid \mathscr{F}_n] \quad = \quad ZE[Y \mid \mathscr{F}_n].$$

Linearity Property

In this section, we prove the linearity property, we want to show that

$$aE[Y \mid \mathscr{F}_n] + bE[Z \mid \mathscr{F}_n] \quad = \quad E[aY + bZ \mid \mathscr{F}_n]$$

we know that $aE[Y \mid \mathscr{F}_n]+bE[Z \mid \mathscr{F}_n]$ is an $\mathscr{F}_n$ measurable random variable. Also if $A \in \mathscr{F}_n$, then the linearity of the expectations implies that

$$\begin{aligned}
&= E[1_A (aE[Y \mid \mathscr{F}_n] + bE[Z \mid \mathscr{F}_n])] \\
&= aE[1_A E[Y \mid \mathscr{F}_n]] + bE[1_A E[Z \mid \mathscr{F}_n]] \\
&= aE[1_A Y] + bE[1_A Z] \\
&= E[1_A (aY + bZ)] \\
&= E[(aY + bZ) \mid \mathscr{F}_n]
\end{aligned}$$

Lawler (2017).

Appendix B

Itô's Formula

Itô's formula is the analog of the Chain rule from elementary Calculus. Let f be a differentiable function of a real variable x , let x_0 be fixed and consider the changes $\Delta x = x - x_0$ and $\Delta f(x) = f(x) - f(x_0)$. It is known from Calculus that the following second order Taylor approximation holds.

$$\Delta f(x) = f^{'}(x)\Delta x + f^{''}(x)\frac{(\Delta x)^2}{2} + O(\Delta x)^3.$$

When x is infinitesimally close to x_0 we replace Δx by the differential dx and obtain

$$df(x) = f^{'}(x)dx + f^{''}(x)\frac{(dx)^2}{2} + O(dx)^3.$$

In elementary Calculus all terms involving terms equal or higher order to $(dx)^2$ are neglected then the aforementioned formula becomes:

$$df(x) = f^{'}(x)dx.$$

Now if we consider $x = x(t)$ to be a differentiable function of t substituting into the previous formula we obtain the differentiable form the well known Chain rule.

$$df(x(t)) = f^{'}(x(t))dx(t) = f^{'}(x(t))x^{'}(t)dt.$$

We shall present a similar formula for the stochastic environment. In this case the deterministic $x(t)$ is replaced by a stochastic process X_t. The composition between the differentiable function f and the process X_t is denoted by $F_t = f(\ X_t)$. Neglecting the increment powers higher than or equal to $(dX_t)^3$, we get:

$$dF_t = f^{'}(X_t)dX_t + \frac{1}{2}f^{''}(X_t)(dX_t)^2. \tag{B.0.1}$$

In the computation of the dX_t, we may take into account the stochastic relations such as $(dW_t)^2 = dt$ or $dtdW_t = 0$.

Appendix C

Brief Introduction to C++ Programming

All the computational work is implemented in C++ and hence need for review of some basic concepts.

Short history of C++

The object oriented way of thinking and programming (we call this paradigm) is almost fifty years old and it has its origins in the programming language Simula that was developed in Norway. Simula was the first, see page 7-8, Duffy (2013c) language to support the concept of class as we know it in current form. C++ has origins in the early 1980s, when its inventor Dr Bjarne Stroustrup was working at AT & T and it was invented in 1997. The original name of the language was "C with classes", because the language was developed as an object oriented extension to the programming language C, while remaining compatible with it. C++ is compatible with C and was called a "better C" in those early days. The late 1980s can be seen as the period when C++ came out of the laboratories and began to manifest itself in mainstream applications, Duffy (2013c).

C++ A Multi Paradigm language

This paradigm is based on the concept of class. Classes have their origins in philosophy, logic and cognitive psychology. In particular the theory of concepts has been an important influence on the development of the object paradigm. For a class in object oriented programming languages, some features of the defining attribute view are:

1. The meaning of a concept is captured by its defining attributes;

2. Attributes are atomic building blocks for concepts;

3. Attributes are necessary and sufficient for defining members of a concept;

4. All members of the concept are equally representative of the concept or we cannot say that one member is more typical of the concept than another member, see Duffy (2013c).

The Compilation process

C++ is a programming language and it is possible to write a code in this language using some kind of text editor. The sentences must conform to the rules as described by the specification of the language.

Definition 31. (**A compiler**) A compiler is an executable program that accepts a text file containing C++ code. It translates this code (in a series of steps) to a form that can eventually be executed in a computer. Basically it translates human readable text into machine readable code. This is a simplistic explanation but it is reasonably accurate.

When writing programs, we try to split the problem into independent pieces or modules. Each module will be implemented by valid C++ code for example :

- A C/C++ function;
- A C++ Class.

In general we create two files, one containing the declaration of all relevant functions and data while the other file contains the actual code of each function. These two files are called header and code files. In short a header file contains declarations of all functions (sometimes called function prototypes) and data. A code file contains the actual body of all functions and the initialised data from the header file, see Duffy (2013c). In the case of functions we define the signature of a function as consisting of:

- Its name;
- Its return type;
- Its input arguments (also known as parameters).

In the case of data we need:

- The name of the data (called the variable name);
- The type of the data (this could be built in type or user-defined type).

Lets consider an example where we wish to find the maximum or minimum of two numbers. These functions are useful when we define the pay-off functions for one factor and two factor options. To this end we create two files called:

- $Inequalities \cdot hpp$ contains the function declarations;
- $Inequalities \cdot cpp$ contains function code.

The header file is included by use of a special preprocessor command, see Duffy (2013c).

Structure of C++

C++ is a strongly typed language, Ødegaard (2017). Everything must be declared before it is used, both variables and functions. C++ has few building blocks, which can be grouped into types, operations and functions.

Types

The types that we will work with are: bool (boolean), int (integers), long, double and string. Here are some example definitions
bool this_is_true=true;
int i=0;
long j=123456789;
double pi=3.14152653897932384626 43;
string S ("this is a string");
The most important part of C++ comes from the fact that these basic types can be expanded by use of classes.

Operations

To these basic types the common mathematical operations can be applied, such as addition, subtraction, multiplication and division.
int i=100+50;
int j=100-50;
int n=100*2;
int m=100/2;
The operations are defined for all the data types with the exception of the string type.

Increment and Decrement

In addition to basic operators there are some additional operations with their own short hand, for example incrementing or decrementing a variable. When we want to increase the value of one item by one, in most languages this is written:
int i=0;
i=i+1;
i=i-1.
In C++ this operation has its own shorthand;
int i=0;
i++;
i–.

Functions and Libraries

In addition to basic mathematical operations there is a large number of additional operations that can be performed on any type. However those are not parts of the core language, they are implemented as stand alone functions (most of which are actually written in C or C++). The functions are included in the library that comes in any C++ installation. Since they are not part of the core language they must be defined to the compiler before they can be used. Such definitions are performed by means of the include statement. For example mathematical operation taking powers and performing exponentiation:

```
#include<cmath>//mathematics library
double a=pow(2,2);
double b=exp(1);
```

,see Ødegaard (2017).

Vectors

A vector in C++ is an object that manages a block of memory that can hold multiple values simultaneously. A vector therefore represents a collection of values. A vector has a name and we may access the values it contains through their position within the block of the memory managed by the vector. A vector stores a sequence of values and the values must be of the same type. A collection of values all of the same type is said to be homogeneous.

Declaring and Using Vectors

In order to use a vector in C++ we must add the preprocessor directive:

```
#include<vector>
```

The vector type is part of the standard *(std) namespace*, so its full name is:

```
std::vector,
```

just like the full name of *cout* is:

```
std::cout.
```

If we include the statement:

```
using namespace std;
using std::vector;
```

at the top of the of our source file, we can use a shorter vector name within our code. We may declare a vector object that can hold integers as simply as:

```
vector<int>vec_a;
```

The type within the angle brackets <> may be any valid C++ data type. When declared this way, the vector vec_a initially is empty. We can declare a vector with a particular initial size as follows:

```
vector<int>vec_b(10);
```

Here vec_b initially holds 10 integers and all elements are zero by default. The vector size appears in the parentheses following the vector's name. We may declare a vector object of a given size and specify the initial value of all elements:

```
vector<int>vec_c(10,8);
```

vec_c initially hold 10 integers, all having a value 8 and lets note that the first number within the parentheses following the vector's name indicates the number of elements and the second number specifies the initial value of all the elements, see Halterman (2016).

Transversing Vector

The action of moving through a vector visiting each element is known as transversal, *for* loops are ideal for vector transversals. If a is an integer vector containing 10 elements, the following loop prints each element in a :

```
for(int i=0;i<10,i++)
cout<<a[i]<<endl;
```

The loop control variable, i step through each valid index of vec_a. Variable, i's values start at 0 and ends at 9, the last valid position in a vector a. The following loop prints contents of a vector a in a reverse order:

```
for(int i=9,i>=9,i--)
cout<<a[i]<<endl;
```

The following code produces a vector named *Set* containing an integer sequence 0,1,2,...,999:

```
vector<int>Set(1000);
for(int i=0;i<100;i++)
Set[i]=i;
```

Standard C++ Classes

In hardware arena, a desktop computer is built by assembling a motherboard (a circuit board containing sockets for a processor and assorted supporting cards), a processor, memory, a video card, an input or output card (USB ports, parallel port and a mouse port), a disk controller, a disk drive, a case, keyboard, a mouse and a monitor. Some of these components like the *Input/Output*, the disk controller and video may be integrated with the motherboard. Software development today is increasingly component based. Software components are used like hardware components. A software system can be built largely by assembling pre-existing software building blocks. C++ supports various kinds of software building blocks. The simplest of these is the function. A more powerful technique uses built in and user designed software objects. C++ is object-oriented, it was not the first object oriented programming language, but it was the first object oriented programming language that gained widespread use in a variety of application areas. An object oriented programming language allows the programmer to define, create and manipulate objects. Variables representing objects can have considerable functionalities compared to primitive numeric variables like *int* and *double*. Like a normal variable, every C++ object has a type. We say an object is an instance of a particular class and class means the same as type. An object's type is its class. We have been using *cout* and *cin*, *cout* is an instance of *ostream* class, which is to say that *cout* is of type *ostream*, *cin* is an instance of the *istream* class. Code that uses an object is a client of the object for example.

```
cout<<"Hello"<<endl;
```

uses the *cout* object, there is a client of *cout*. Many of the functions we have seen so far have been clients of *cout* and or *cin* objects. Objects provide service to clients, Halterman (2016).

String Object

A *string* is a sequence of characters, most often used to represent words and names. The standard C++ library provides the class String which specifies string objects. In order to use string objects, we must provide the pre-processor directive.

```
#include<string>
```

The *string* class is part of the of the standard *namespace*, which means its full type name is *std::string.* We can declare a *string* object like any other variable: *string name;* We must assign a literal character sequence to a *string* object via a familiar string syntax:

```
string name="Joe";
cout<<name<<endl;
name="Jane";
cout<<name<<endl;
```

Like the vector class, the string class provides a number of methods, some string methods include; operator[ ] provides access to the values stored at a given index within the string. *Operator=* assigns one string to another, *operator +* appends a string or a single character to the end of a string, *at* provides bounds-checking access to the character stored at a given index. *Length*-returns the number of characters that make up the string, *size*-same as length. *Find* locates the index of sub-string within a string object, *substr* returns a new string object made of the substring, *empty* returns true if the string contains no characters; returns false if the string contains one or more characters, *clear* makes the vector empty. Here is a code fragment:

```
string word="computer";
cout<<"\"<<word<<"\"contains"<<word.length()
<<"letters"<<endl;
```

the code above prints "a computer" contains 8 letters. A string class defines a method named *operator[ ]* that allows a programmer to access a character within a string as in the code fragment below:

```
cout<<"The letter at index 3 is"<<word.[3]<<endl;
```

Custom Objects

Named variables and functions allow programmers to abstract away such machine level details and concentrate on concepts that transcend computers such as integers and characters. Objects provide a level of abstraction above that of variables. Objects allow programmers to go beyond simple values, developers can thus focus on more complex things like geometric shapes, bank accounts and aircraft wings. Programming objects that represent these real world things can possess capabilities that go beyond the simple variables, we have studied to this point. A C++ object typically consist of data and code, by bundling data and code together, objects store information and provide service to other parts of the software system. An object forms a computational unit that makes up part of the overall computer application. A programming object can model a real world object more naturally than a collection of simple variables since it encapsulate considerable complexity. Objects make it easier for developers to build complex software systems. C++ is classified as an object oriented language, most modern languages have some degree of object orientation. In this section we will be illustrating how a programmer can define, create and use custom objects, Halterman (2016).

Object Basics

Mathematicians represent a single point as an ordered pair of real numbers, usually expressed as (x, y). In C++, the double type serves to approximate a subset of mathematical real numbers. A point with coordinates within the range of a double precision floating point numbers, therefore can be modelled by two double variables in C++. Something that conceptually is one thing requires two variables in C++. As a consequence a function that computes distance between two points requires four parameters; x_1, y_1, x_2, y_2 rather than two points (x_1, y_1) and (x_2, y_2). Ideally we should be able to use one variable to represent a point. One approach to represent a point could use a two element vector, for example:

```
vector<double>pt{3.2,0.0};
```

This approach has several problems:

1. We must use numeric indices instead of names to distinguish between the two components of a point object. We may agree that *pt[0]* means x-coordinate of *pt* and *pt[1]* means the y-coordinate of point *pt* but the compiler is powerless to detect the error if the programmer uses an expression like *pt[19]* or *pt[-3];*

2. We cannot restrict the vector's size to two. A programmer may accidentally push extra items into the back of a vector representing a point object;

3. We cannot use a vector to represent objects in general. Lets consider a bank account object, a bank account object could include among other things, an account number (an integer), a customer name (a string), an interest rate (a double precision floating). A vector implementation of such an object is impossible because elements in a vector must all be of the same type. In addition to storing data, we want our objects to be active agents that do computational tasks. We need to be able to associate code with a

> class of objects. We need a fundamentally different programming construct to represent objects. An object programmer using the objects needs to know only what the object can do without needing to know how it works. An object provides an interface to any client code that wishes to use the object. A typical object selectively exposes some parts of itself to the clients and keeps other parts hidden to the client. The object's designer on the other hand must know the complete details of the object's implementation and must be an expert on both what the object does and how it works.

C++ uses the class keyword to define the structure of an object. A class serves as a pattern or template from which the programmer may produce objects. We will concentrate on five things facilitating object oriented programming with C++ classes:

1. A class can specify the data that constitute an object's state;
2. A class can define code to be executed on an object's behalf that provides services to clients that use the object;
3. A class may define code that automatically initialise a newly created object ensuring that it begins its life in a well defined state;
4. A class may define code that automatically cleans up any resources an object may be using when its life is finished;
5. The C++ language provides a way for the developer of a class to specify which parts of its objects are visible to clients and which parts are hidden from the client.

A class is a programmer defined type. An object is an instance of a class. The term instance and object may be used interchangeably, Halterman (2016).

Classes: Private and Public members

Classes and structures are almost the same. In fact the only difference is that by default members of a structure are accessible from the outside via $\cdot$ and $\rightarrow$ operators. In a class, on other hand, the members are by default hidden within the class and cannot be accessed from the outside. We say that members of the class are declared private by default. However if we want we can declare members of the class to be public. Nevertheless, its bad programming practice to declare member data to public. For safe and easy to update code, we should always declare these as private. If the private members of the class are not accessible outside the class, we have to find a way to change and inspect their values. This can be done with the help of functions that have access to member data and are themselves declared inside the class, see Nürnberg (2016).

Fields

The simplest kind of objects stores only data. We can define a point type as follows:

```
class  Point{
public:
double x;
double y;
};
```

The semi colon that follows the close curly bracket of the class definition is required but it is easy to overlook. By convention class names begin with capital letters, but class names are just identifiers like variable names and function names. Here class name is point, the body of the class appears in the curly braces. The class Point specifies two *double* precision floating point data components named x and y. These are called fields or data members. They both appear within the class body after the *public* label. We say that x and y are public fields of the Point class; this means that the client code has full access to x and y fields. Any client may examine and modify the x and y components of a Point object, Halterman (2016).

Methods

Member functions of a class are also referred to as *methods*. If we make the member data of our *Student* class private, we should provide functions to create and print the data. The declaration of the class would the look as follows:

```
class Student{
string name,firstname;
int year;
public:
void create(const string&n,const string&fn,int y);
void print();
};
```

The member data is declared private, since this is the default setting for a class. Before we can use them we have to give their definitions. We have to take note of the usage of the scope operator :: in the definition of member functions. An example below illustrates how we use the newly defined methods, see Nürnberg (2016).

Simulating random normal numbers

Generation of random numbers is a large topic and the generated numbers can never be truly random, only "pseudo"-random. They will be generated according to some reproducible algorithm and after a (large) number of random number generations the sequence will start repeating itself. The number of iterations before replication starts is a measure of the quality of a random number generator. For anybody requiring high-quality random numbers, generators like the *rand()* function provided by the standard C++ library should be avoided, but for not getting into involved discussion of random number generations we use this function as a basis for the generation of uniformly distributed numbers in the interval $[0; 1)$ These uniformly distributed distributed random variates are used as a basis for the polar method for normal densities, see Ødegaard (2017).

c

Bibliography

Baxter, M. and Rennie, A. (1996). *Financial Calculus: An Introduction to Derivative Pricing*. Cambridge University Press.

Benth, F. E., Kallsen, J., and Meyer-Brandis, T. (2007). A Non-Gaussian Ornstein–Uhlenbeck Process for Electricity Spot Price Modeling and Derivatives Pricing. *Applied Mathematical Finance*, 14(2):153–169.

Boyle, P. P. (1977). Options: A Monte Carlo Approach. *Journal of Financial Economics*, 4(3):323–338.

Boyle, P. P. and Tian, Y. (1998). An Explicit Finite Difference Approach to the Pricing of Barrier Options. *Applied Mathematical Finance*, 5(1):17–43.

Broadie, M. and Glasserman, P. (1997). Pricing American-Style Securities Using Simulation. *Journal of Economic Dynamics and Control*, 21(8):1323–1352.

Brody, D. C., Syroka, J., Zervos, M., et al. (2002). Dynamical Pricing of Weather Derivatives. *Quantitative Finance*, 2(3):189–198.

Calin, O. (2017). An Introduction to Stochastic Calculus with Applications to Finance,. https://people.emich.edu/ocalin/Teaching files/D18N.pdf. Accessed: 2017-02-10.

Cartea, A. and Figueroa, M. G. (2005). Pricing in Electricity Markets: A Mean Reverting Jump Diffusion Models with Seasonality. *Applied Mathematical Finance*, 12(4):313–335.

Cont, R. (2001). Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues. *Quantitative Finance*, 1(1):223–236.

Davis, M. H. (2004). Valuation, Hedging and Investment in Incomplete Financial Markets. *Applied Mathematics Entering the 21st Century, eds. JM Hill and R. Moore, Society for Industrial and Applied Mathematics*, pages 49–70.

Duffy, D. J. (2013a). *Financial Instrument Pricing using C++*. John Wiley & Sons.

Duffy, D. J. (2013b). *Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach*. John Wiley & Sons.

Duffy, D. J. (2013c). *Introduction to C++ for Financial Engineers: An Object-Oriented Approach*. John Wiley & Sons.

Finan, M. B. (2008). *A Basic Course in the Theory of Interest and Derivatives Markets: A Preparation for the Actuarial Exam FM/2*. Arkansas University.

Finan, M. B. (2011). *A Discussion of Financial Economics in Actuarial Models A Preparation for the Actuarial Exam MFE/3F*. Arkansas University.

García-Machado, J. J. and Rybczyński, J. (2015). Three-point Volatility Smile Classification: Evidence from the Warsow Stock Exchange during Volatile Summer 2011. *Investigaciones Europeas de Dirección y Economía de la Empresa*, 21(1):17–25.

Glasserman, P. (2013). *Monte Carlo Methods in Financial Engineering*, volume 53. Springer Science & Business Media.

Halterman, R. L. (2016). *Fundamentals of C++ Programming*. Richard Halterman.

Haugh, M. (2004). Generating Random Variables and Stochastic Processes. *Monte Carlo Simulation: IEOR EA703*.

Kadalbajoo, M. K., Kumar, A., and Tripathi, L. P. (2015). Application of the Local Radial Basis Function-Based Finite Difference Method for Pricing American Options. *International Journal of Computer Mathematics*, 92(8):1608–1624.

Kijima, M. (2013). *Stochastic Processes with Applications to Finance*. CRC Press.

Korn, R., Korn, E., and Kroisandt, G. (2010). *Monte Carlo Methods and Models in Finance and Insurance*. CRC press.

Kou, S. G. and Wang, H. (2004). Option Pricing under a Double Exponential Jump Diffusion Model. *Management Science*, 50(9):1178–1192.

Lawler, G. F. (2017). Stochastic Calculus: An Introduction with Applications. https://www.math.uchicago.edu/ lawler/finbook2.pdf. Accessed: 2017-04-06.

London, J. (2005). *Modeling Derivatives in C++*, volume 263. John Wiley & Sons.

Madan, D. B. and Seneta, E. (1990). The Variance Gamma (VG) Model for Share Market returns. *Journal of Business*, pages 511–524.

Mandelbrot, B. (1963). The Stable Paretian Income Distribution When the Apparent Exponent is Near two. *International Economic Review*, 4(1):111–115.

Mazengera, H. (2017a). Derivation of a Stochastic Loan Repayment Model for Valuing a Revenue-Based Loan Contract. *Annals of Financial Economics*, 12(3):1750013.

Mazengera, H. (2017b). Revenue-Based Lending for SMEs. *International Journal of Financial Engineering*, 4(2 & 3):1750035.

Miermont, G. (2017). Advanced Probability. http://perso.ens-lyon.fr/gregory.miermont/AdPr2006.pdf. Accessed: 2017-03-21.

Mike, S. and Farmer, J. D. (2008). An Empirical Behavioral Model of Liquidity and Volatility. *Journal of Economic Dynamics and Control*, 32(1):200–234.

Ning, C., Xu, D., and Wirjanto, T. S. (2015). Is Volatility Clustering of Asset Returns Asymmetric? *Journal of Banking & Finance*, 52:62–76.

Nürnberg, R. (2016). Computing in C++ Part I : An Introduction to C. https://wwwf.imperial.ac.uk/ rn/teaching/ccourse.pdf. =Accessed: 2017-03-10.

Ødegaard, B. A. (2017). Financial Numerical Recipes in C++. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.212.3734&rep=rep1&type=pdf. Accessed: 2017-04-03.

Papapantoleon, A. (2008). An Introduction to Levy Processes with Applications in Finance. *arXiv preprint arXiv:0804.0482*.

Pflug, G. C. and Thoma, P. (2016). Efficient Calculation of the Greeks for Exponential Lévy processes: An application of Measure valued Differentiation. *Quantitative Finance*, 16(2):247–257.

Ross, S. M. (2014). *Introduction to Probability Models*. Academic Press.

Staum, J. (2007). Incomplete Markets. *Handbooks in Operations Research and Management Science*, 15:511–563.

Tankov, P. (2003). *Financial Modelling with Jump Processes*, volume 2. CRC Press.