JP Morgan Stock Analysis

Daniel Cotto

Catholic Univeristy of the Sacred Heart

MSc Banking and Finance, 4805453

September 4, 2019

Contents

1 Introduction **3**

2 JP Morgan Stock **4**
2.1 Company Overview 4
2.2 Time Series Analysis 4
2.3 Bollinger Bands 5
2.4 Log Returns 5
2.5 Change Point Detection 6
2.6 Log Returns Distribution 7

3 Option Pricing **8**
3.1 Geometric Brownian motion 8
3.1.1 Failure of gBM Assumptions 8
3.2 Parameters estimation 8
3.2.1 Efficient and unbiased estimators 8
3.2.2 Maximum Likelihood Estimation 9
3.2.3 Quasi Maximum Likelihood Estimation 10
3.3 European Options 12
3.3.1 Black & Scholes Formula 12
3.3.2 Put-Call Parity 13
3.3.3 Monte Carlo Method 14
3.3.4 Fast Fourier Transfors 15
3.4 Greeks - European Option 17
3.4.1 Delta 17
3.4.2 Gamma 18
3.4.3 Vega 19
3.4.4 Theta 20
3.4.5 Rho 21
3.4.6 Volga 22

3.4.7 Vanna . . . 22
3.5 GARCH Models . . . 24
3.5.1 GARCH(p,q) . . . 25
3.5.2 EGarch . . . 26
3.5.3 Threshold Garch . . . 27
3.6 Lévy Processes . . . 29
3.6.1 Introduction to Lévy processes . . . 29
3.6.2 Variance Gamma model . . . 29
3.6.3 Parameters Estimation . . . 30
3.6.4 Pricing FFT and Monte Carlo . . . 30
3.7 American Options . . . 31
3.7.1 Explicit finite-difference Method . . . 31
3.7.2 Implicit finite-difference Method . . . 32
3.7.3 Broadies and Glasserman simulation method . . . 33
3.7.4 Longstaff and Schwartz least squares method . . . 33
3.8 Multi-Assets Options . . . 35
3.8.1 Multi-dimensional Geometric Brownian Motion . . . 35
3.8.2 Copula Method . . . 36
3.9 Neural Networks for Option Pricing . . . 40
3.9.1 Artificial Neural Networks . . . 40
3.9.2 Methodology . . . 41
3.9.3 Results . . . 41
3.10 Conclusions . . . 44

4 Bibliography **45**

5 Appendix **47**

1 Introduction

This paper aims to analyse different pricing methods for some financial derivatives: European Option, American Options, Multi-Asset Options. All of these financial instruments have been built upon the JP Morgan Stock as the main underlying. For this reason, a brief view of the time series of this stock and its returns is provided.

This paper is composed by three parts: the first one is based on time series analysis; the second one is mainly focused on option pricing methods, using both geometric Brownian motion assumptions and Levy processes; the last part is an attempt to apply deep learning techniques to option pricing.

The time series analysis results to be necessary to check volatility through the Bollinger Bands and Change Point Detection. Multiple QQ tests have been implemented to verify the normality assumptions on log-returns. Further analysis using GARCH models is conducted to verify the assumption of constant volatility.

In the second part, the JP Morgan stock (JPM) is used as underlying to price different types of options. To do so, different methods have been used to describe the dynamics of the assets and to calibrate the models using different parameters. The initial model used relied on the gBM assumptions. In order to correctly calibrate the model, the MoM, MLE and QMLE estimation techniques have been used to get efficient unbiased estimator. Once got the parameters, the pricing is conducted using BS formula, Monte Carlo simulations and FFT. The Greeks letters are also computed. After that, the Levy process, which has more realistic assumptions, is carried out: the ones used here is the Variance-Gamma.

In the third part, JP Morgan stock is used to price multi-asset options using both linear correlation and Copula method to model the relatioship between two assets.

In the last part, a brief overview on artificial neural networks is provided and then two models are estimated on a simulated dataset. Then, the results on predicted values are discussed.

2 JP Morgan Stock

2.1 Company Overview

JPMorgan Chase, in its current structure, is the result of the combination of several large U.S. banking companies since 1996. It operates in four segments: Consumer Community Banking (CCB), Corporate Investment Bank (CIB), Commercial Banking (CB), and Asset Wealth Management (AWM). The company was founded in 1799 and is headquartered in New York, New York. JPM stock is also a component of DJIA, SP 100 and SP 500 indexes.

2.2 Time Series Analysis

To conduct this analysis, the starting period considered is January 3rd,2000 up to August 22nd, 2019. The graph reports also the Simple Moving Average at 50 and 200 days. A simple moving average smooths out volatility and makes easier to view the price trend of a security. SMA is frequently used in technical analysis to identify the medium and long trend.

Analyzing stock price trend, from the 1st January 2005 to the 1st January 2015, it is possible to notice the effect of the financial crisis 2008 on the time series and after a stabilizing period, the stock price started rising rapidly from January 2013 up to today.

Figure 1: Stock price history

2.3 Bollinger Bands

Bollinger bands are frequently used to monitor the volatility of an asset. They have three components: a 20-days simple moving average, an upper band that is usually equal 2x sd above the SMA, and the lower bound that is the opposite of the upper band. The Bollinger Bands are used as volatility indicator as they widen with more volatility and narrow during lower volatility.

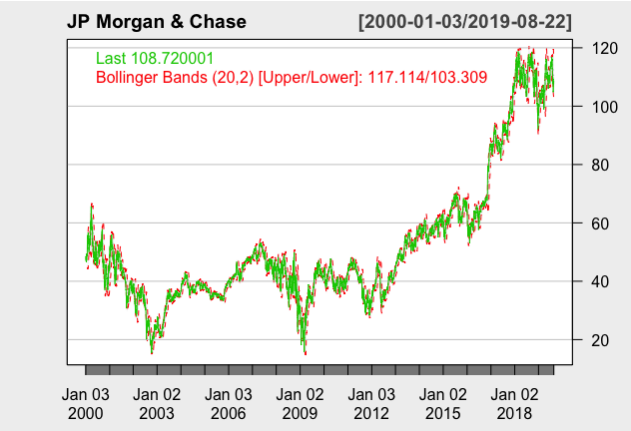

Figure 2: Bollinger Bands 2000-2019

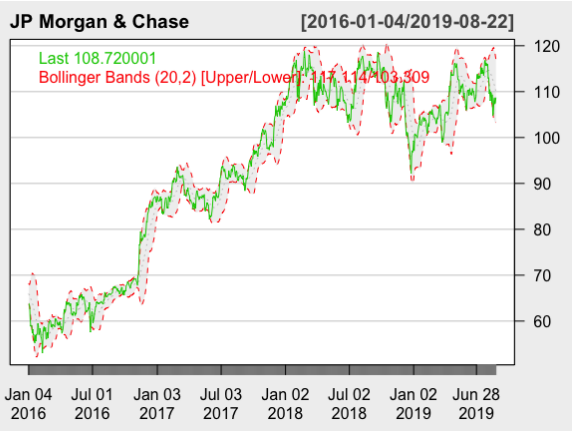

Figure 3: Bollinger Bands 2016-2019

2.4 Log Returns

From here, it will be used log-returns of the assets as they allow easier comparison between different assets and help to evaluate the log normality. Moreover, log returns permit time-additivity, mathematical ease and numerical stability.

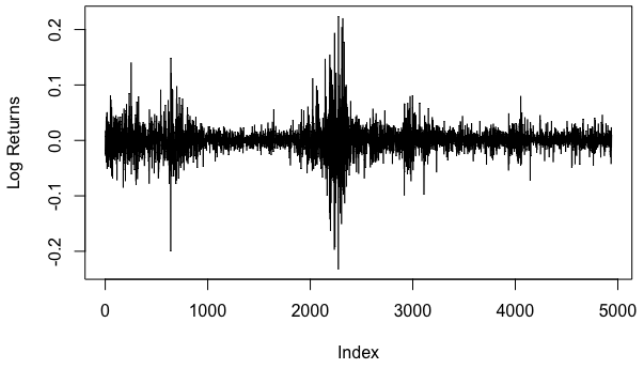

Figure 4: Log-returns of the JPM

2.5 Change Point Detection

To correctly price options, it is necessary to remove anomalous behaviour from the dataset. Indeed, these behaviours can change the probability distribution of a time series and the Change Point Detection allows to identify the moment in which probability distribution of the time series has changed. In the case of the JPM time series from the 1st January 2000 to the 23rd August 2019, such change has occurred on the 31st October 2007. In Figure 5 this change is identified by the vertical line. To provide a better visualization on the change point, in Figure 6 it is plotted only the relevant time period. Therefore, from here the series will be considered only from the 31st October 2007 to the 22nd August 2019.

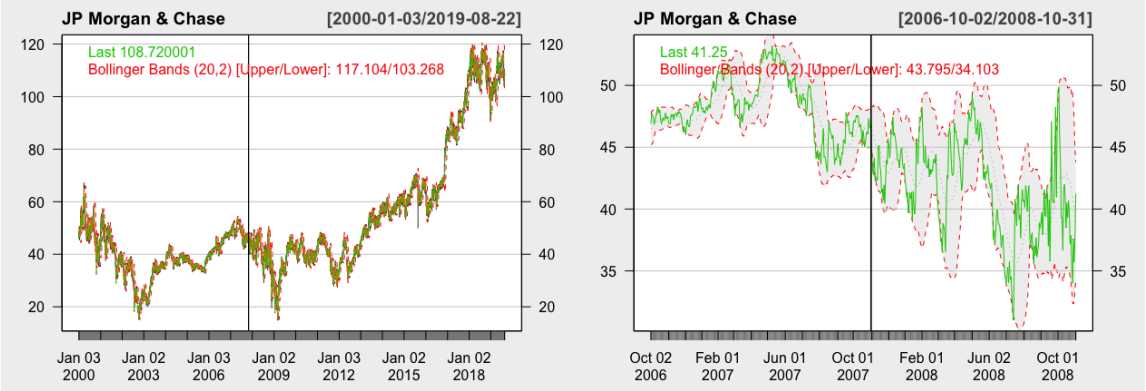

Figure 5: Change Point Detection on the JPM

Figure 6: zoom on Change Point area

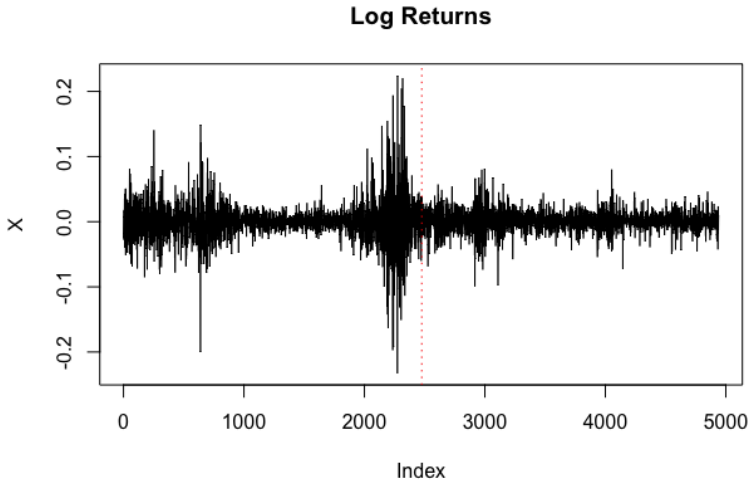

Figure 7: CPD on Log-returns of the JPM

2.6 Log Returns Distribution

At First glance at the histogram, one may think that the log-returns have a distribution similar to a Gaussian. The numerous studies on log-returns of financial securities have highlighted that the assumption on Gaussian distribution of log-returns failed many times. The empirical evidence shows that probabilities of extreme values of log-returns are much larger than those of Gaussian distribution. In two papers (1996a, 1996b), Markowitz and Usmen, analyzing SP500 log-returns, considered the rich family of Pearson distributions and identified the Student-t distribution with about 4 or 5 degrees of freedom as the best fit to daily log-return data of the SP500. Recently, Fergusson Platen concluded, at a high level of significance, that the log-returns of their index exhibited a Student-t behaviour with approximately four degrees of freedom. Right Tail zoom with respect Normal distribution is shown in Figure 9

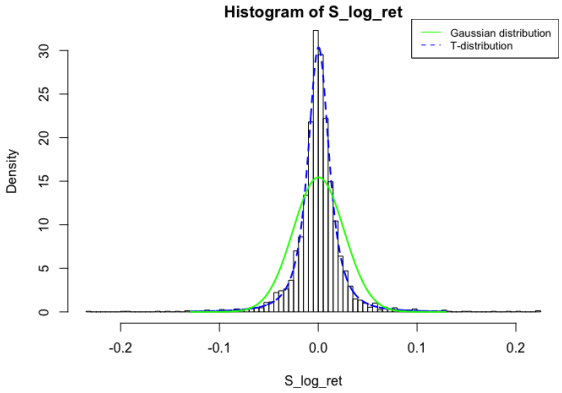

Figure 8: Histogram Log Returns

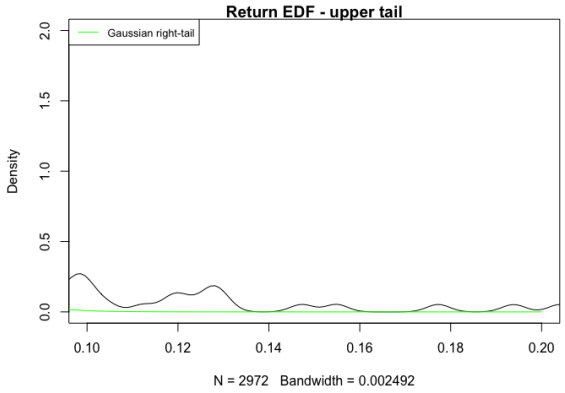

Figure 9: Right Tail zoom

This is true also for the considered asset. Indeed, by the QQ plot, the normal distribution is not an adequate model of the stochastic component

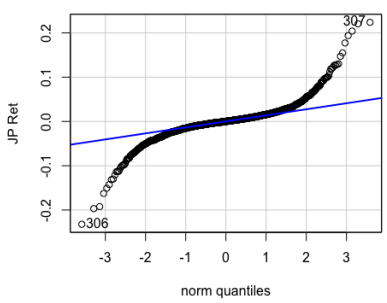

Figure 10: QQ-Plot Norm Distribution

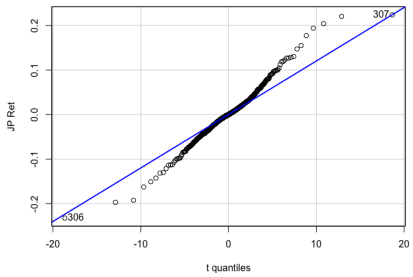

Figure 11: QQ-Plot T-distrib (df=3)

Up to here, it is possible to see that the main properties of log returns hold for JPM stock. It has been shown that log-returns of this asset have heavy tails, the frequency distribution of log-returns decrease more slowly than the Gaussian one. Log returns result to be slightly negatively skewed, showing an asymmetry. Moreover, it has been graphically shown that volatility is nonconstat, stochastic volatility[Figure 4].
Looking at the data, we notice a few extremely large returns, both positive and negative. These outliers are called jumps. A way to account for these jumps is to use Lévy process in diffusion models and this is shown in the second part of the paper through the use of Variance-Gamma process.

3 Option Pricing

3.1 Geometric Brownian motion

3.1.1 Failure of gBM Assumptions

The benchmark model for spot price dynamics in continuous time is the geometric Brownian motion where constants r and sigma represent the instantaneous annualized risk-free rate of interest and price volatility. One benefit of geometric Brownian motion is that negative asset prices are not possible because any price change is proportional to the current price. Bankruptcy could drive an asset price down to but not past the natural absorbing barrier at zero (Chance, 1994)

The so called geometric Brownian motion satisfies the stochastic differential equation

$$dS = \mu S_t dt + \sigma S_t dW_t \tag{1}$$

The solution of the SDE is explicit:

$$S_t = S_0 exp\{(r - \frac{\sigma^2}{2})t + \sigma W_t\} \tag{2}$$

Its popularity arises from the Black Scholes formula: if the stock price dynamics could be modelled as a gBM, the Black Scholes formula would be an optimal tool in pricing financial derivatives. As shown, the assumptions of gBM are violated: log-returns are neither normally distributed nor mutually independent; more seriously, returns exhibit volatility clustering, which means that high volatility in current returns then we can expect this higher volatility to continue, at least for a while. Although the reality does not match the gBM assumption, it plays a relevant role in the option pricing due to its application on the Black Schole formula.

All models are wrong but some are useful. (George Box,1978)

3.2 Parameters estimation

3.2.1 Efficient and unbiased estimators

Starting from the assumption that stock price dynamics follows a gBM process and,thus, the log returns are normally distributed, $X \sim N(\alpha\Delta t, \sigma^2\Delta t)$, the mean and the variance can be estimated.

$$\hat{\alpha} = \frac{1}{n}\frac{1}{\Delta t}\sum_{i=1}^{n} X_i$$
$$\hat{\sigma}^2 = \frac{1}{n-1}\frac{1}{\Delta t}\sum_{i=1}^{n}(X_i - \hat{\alpha}\Delta t)^2$$

And the results got

$$\hat{\alpha} = 0.07110845$$
$$\hat{\sigma}^2 = 0.4100527$$

3.2.2 Maximum Likelihood Estimation

Maximum likelihood is the most important and widespread method of estimation. It generally provides more efficient (less variable) estimators than other techniques of estimation.
Let X_i a sample formed by n random variables which are i.i.d. and belongs to a population with density function $f(x, \theta)$. This function is called *likelihood function*. It tells the likelihood of the sample that was actually observed. The maximum likelihood estimator (MLE) is the value of θ that maximizes the likelihood function.

$$\zeta(x_1, x_2, ...x_i, \theta) = \prod_{i=1}^{n}(f(x_i, \theta)$$
$$\hat{\theta} = arg\, max\ \zeta(x, \theta)$$

A useful mathematical trick for simplifying the computation of an MLE is to find instead a value of θ that maximizes the logarithm of ζ, called the log likelihood function

$$\ell = ln\, \zeta(x, \theta) \tag{3}$$

Maximizing ℓ is the same as maximizing ζ, because ln is a monotone function.
The parameters estimated with such method are:

$\hat{\mu} = 0.0002831703$ St.Err $= 0.0004737951$
$\hat{\sigma}^2 = 0.0258294405$ St.Err $= 0.0003328225$

with $-2logL = -13299.19$

3.2.3 Quasi Maximum Likelihood Estimation

The Quasi-Maximum Likelihood Estimate (QMLE) is an expansion of MLE. Such method is an estimate of a parameter θ in a statistical model; the function that is maximized to form a QMLE is often a simplified form of the actual log-likelihood function.
A common way to form such a simplified function is to use the log-likelihood function of a misspecified model. Under this misspecification, the model fitting procedure is known as the quasi-maximum likelihood (QML) and the estimates obtained are QMLEs. Usually, the Gaussian likelihood is treated as an objective function to be maximized rather than a proper likelihood. The QMLE estimates will, in general, be less precise than those from MLE unless the true density actually is normal.
Thus, there is a trade-off between theoretical asymptotic parameter efficiency and practicality.

	$\hat{\mu}$	$\hat{\sigma}$
Estimated values	0.1555073	0.4166834
Initial guess	0.02	0.05
Lower bound	-0.01	0.00
Upper bound	1.00	1.00

Table 1: QMLE - Attempt 1

The problem is that the QMLE leads to comparable results only if the true model does not fail in being a gBm which, as already abundantly proved, is not the case. Consequently, even implementing the method with a consistent initial guess, the estimation is totally misleading.

	$\hat{\mu}$	$\hat{\sigma}$
Estimated values	0.0009523236	0.0279349925
Initial guess	0.001	0.02
Lower bound	0.00002	0.015
Upper bound	0.001	0.028

Table 2: QMLE - Attempt 2

Even modifying the upper and lower bound to converge to the ML estimates, the results keep being far from the ML results got above. The third attempt is made choosing initial guess strongly greater than the previous one to see if the initial guess completely misleading may improve the results but it has not improved the output

	$\hat{\mu}$	$\hat{\sigma}$
Estimated values	0.009918694	0.089945730
Initial guess	0.01	0.06
Lower bound	0.00	0.00
Upper bound	0.01	0.09

Table 3: QMLE - Attempt 3

The estimated values in attempt 2 are much more similar to the ones obtained with the MLE methods. This is because of narrowing the corridor (upper and lower bounds) to estimate parameters. Repeating the operation and changing the initial guess and the lower and upper bound, it is noticeable that the QMLE tends to converge to values which are extremely near to the lower bound and this makes this method extremely inconsistent. This tendency is due to the inconsistency of the method, which is also justified by the fact that stock prices do not follow a gBM and its related assumptio are violated.

3.3 European Options

3.3.1 Black & Scholes Formula

As previously mentioned, a model for stock prices which is frequently used and is also the basis of the classical Black-Scholes approach is the so-called geometric Brownian motion. Modelling the underlying as geometric Brownian motion provides a useful approximation to stock prices accepted by practitioners for short and medium maturity.
We can interpret the price of a call as the discounted expected option payoff $f(x) = max(0, S_T - K)$, which is the terminal condition, given the current stock price S.

$$C(x,t) = e^{-r(T-t)}\,\mathbb{E}\, f(Z_T^{x,t})$$
$$\textit{where the solution is}$$
$$C(x,t) = S\phi(d_1) - e^{-r(T-t)}K\phi(d_2)$$

$$\mathrm{d}_1 = \frac{ln(\frac{S}{K} + (r\frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{(T-t)}} \qquad d_2 = d_1 - \sigma\sqrt{(T-t)}$$

$$\mathrm{Z}_T^{x,t} = x\, exp\{(r - \tfrac{\sigma^2}{2})(s-t) + \sigma(B_s - B_T\}$$

Setting the parameters:

$\mathrm{S}_0 = 108.72$	*Price at August 22,2019*
$\mathrm{K} = 100.00$	*Strike price*
$\sigma = 0.4100527$	*Estimated above*
$\mathrm{T} = 0.4801587$	*Expiration: December 30,2019*
$\mathrm{r} = 0.0199$	*3M treasury rate*

The Call price for the given parameters is

$$C = 17.19233$$

the high value of the call option is due to low interest rates, high volatilty estimated and an T-t equal to 4months which represents a large time-horizon.

Figure 12 represents the Black-Scholes prices for the European call option C(S,t) for different values of S. When S goes to 0, the price of the call is approaching to the payoff of the option. The difference between option price and payoff is due to the **time value** and reflects the probability that the option will become more profitable to exercise before it expires.

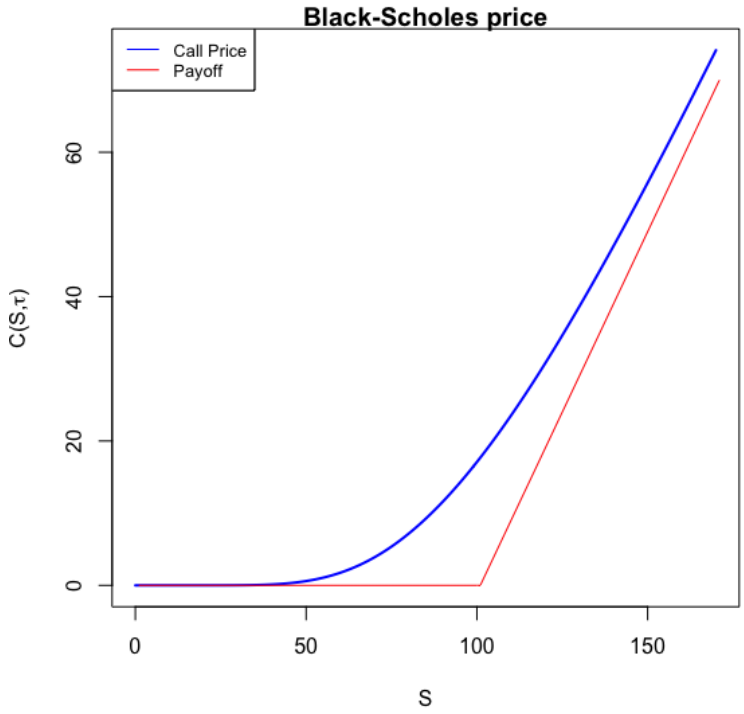

Figure 12: Black-Scholes

Limit cases:

- If $S > K : \phi(d_1) = 1$, and value of a call option on a non dividend paying stock can be approximated by the current stock price minus the discounted exercise price.
- If $S = 0 : \phi(d_1) = 10$ Thus the option is worthless: C(0,t) = 0.

3.3.2 Put-Call Parity

The closed Black - Scholes formula for the European Put options is

$$P(x,t) = e^{-r(T-t)}K\phi(-d_2) - S\phi(-d_1) \tag{3}$$

The Put price for the given parameters is

$$P = 7.534574$$

The Put - Call parity for European Options states that :

Theorem 1 *For the value of a European call and put option which have the same maturity date T, the same strike price K, the same underlying, the following holds:*

$$C + Ke^{-r(T-t)} = P^* S_0$$

$$17.19233 + 100e^{-r(0.4801587)} - 108.72 = 7.521367$$

3.3.3 Monte Carlo Method

The Monte-Carlo simulation is often a single reliable solution to many financial problems when is not possible to rely on a closed formula.
Monte Carlo Methods can be used to value any European-style derivative security. The price of a European-style option can be expressed as the discounted expected payoff, under the risk-neutral measure. Assuming a constant risk-free rate r, with continuous compounding, we have

$$C(x,t) = e^{-r(T-t)}\,\mathbb{E}\, f(Z_T^{x,t}) \tag{4}$$

Within the simulation of independent copies of the random variable from some prescribed distributions.

$$Z_T^{x,t} = x\, exp\{(r - \frac{\sigma^2}{2})(s-t) + \sigma\sqrt{T-t}u\} \tag{5}$$

with $u \sim N(0,1)$ and applying the payoff function to simulated values it is possible to compute the average of these values and discount them to get the price of the Europea-style option.
The justification of this methods relies on the strong law of large numbers that ensure that this procedure converges almost surely to the actual value as the number of simulations M increases.

M	100	1000	1MLN
B&S Call	17.19233	17.19233	17.19233
MC Call	16.66347	17.67678	17.2083

Table 4: B&S - Monte Carlo

This convergence is due to the fact that the gBm is a stable model and based on a normal distribution. If this hypotheis were not true, that is skewness or fat tails, such convergence may not be preserved.

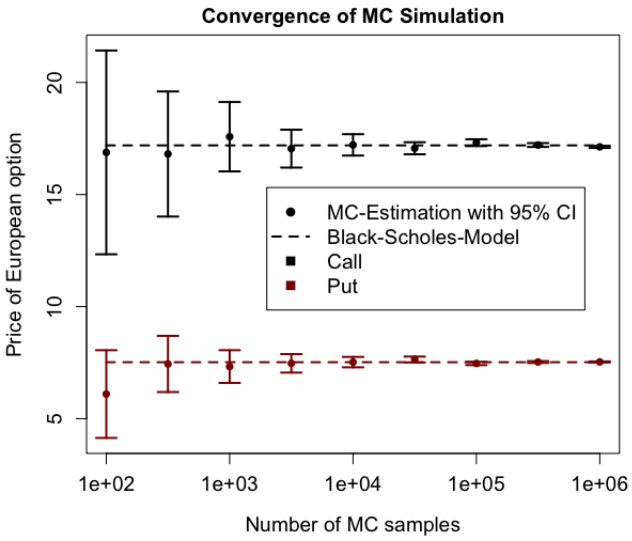

Figure 13: Convergence of MC Simulation

3.3.4 Fast Fourier Transfors

A modern field of derivative valuation is based on Fourier inversion, with several techniques now available to value derivative contracts with asset dynamics that are more complex than the lognormal distribution. The Fourier inversion of the characteristic function approach generates a result with a form similar to the classic Black–Scholes analytical equation (Schmelzle, 2010). The approach developed by Carr and Madan (1999) allows the use of the highly efficient fast Fourier transform (FFT) algorithms to transform the damped European option price. For a European call option the general formula for the price

$$C(x,t) = S\prod\nolimits_1 - e^{-r(T-t)}K\prod\nolimits_2 \tag{6}$$

where

$$\begin{aligned}
\prod\nolimits_1 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{exp(-iulogK)\,\mathbb{E}(expi(u-i)logS_T))}{iu\,\mathbb{E}(S_T)})du \\
&= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{exp(-iulogK)\gamma(u-i)S_T))}{iu\gamma(-i)})du \\
\prod\nolimits_2 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{exp(-iulogK)\,\mathbb{E}(expiulogS_T))}{iu\,\mathbb{E}(S_T)})du \\
&= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re(\frac{exp(-iulogK)\gamma(u)S_T))}{iu})du
\end{aligned}$$

As it is assumed that S_T follows a gBM, the explicit formula for $\prod_1$ and $\prod_2$ is known.

$$\gamma(u) = exp(iu(\mu - \frac{1}{2}\sigma^2) - \frac{\sigma^2 u^2}{2}) \tag{7}$$

The price got from the FFT is thus:
Call = 17.18998
The two prices look quite close and this is also shown in Figure 14 which reports the difference between the price given by the exact formula and with the FFT approximation.

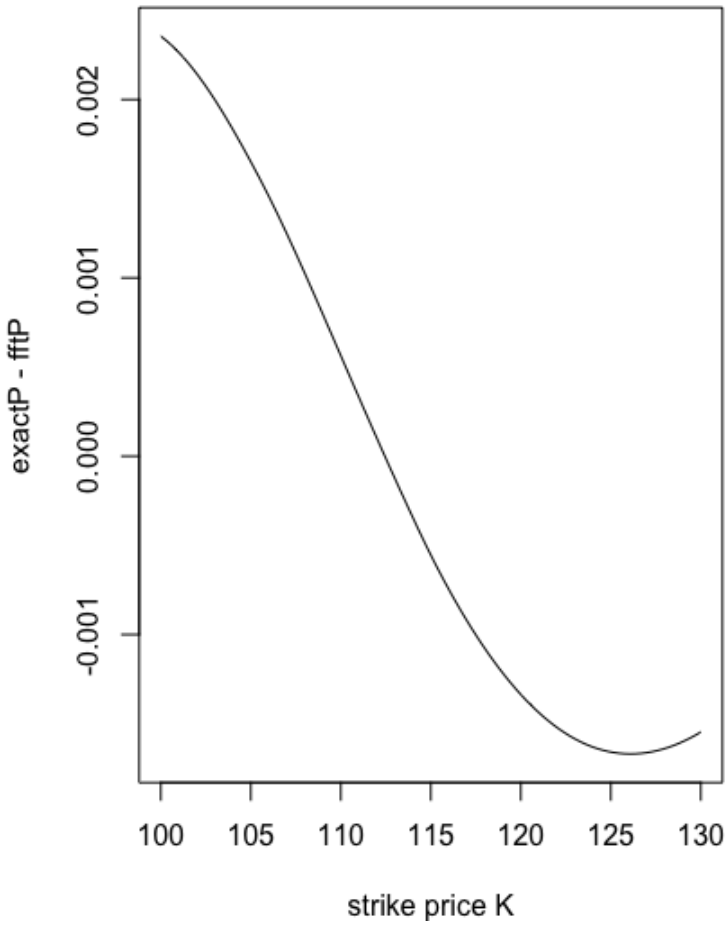

Figure 14: Difference between actual price and FFT one

This consistency is due to the assumption made on the gBm the model and therefore being the SDE is solved always the same.

3.4 Greeks - European Option

Greek letters shows the sensitivity of the premium to different paramenters. They are used In order to reduce the risk associated with option trading .

3.4.1 Delta

The delta of an option or a portfolio of options is the sensitivity of the option or portfolio to the underlying. Since the delta of an option depends on the stock price and time, among others, the position is only delta neutral for a short period of time.

Black&Scholes: The delta of a Call is:

$$\Phi(d_1) \qquad d_1 = \frac{ln(\frac{S}{K}) + (r\frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{(T-t)}} \qquad \Delta = 0.6808008 \tag{8}$$

With respect to S:

- For an increasing stock price delta converges to 1 and the seller of such an option should be long in the underlying to cover the exercise risk.
- For decreasing stock prices it converges to 0. the option is far out-of-the-money and probably not be exercised, thus the seller can hold only a smaller part of the underlying

With respect to T:

- The probability that an out-of-the-money (OTM) option will be exercised and an ITM option will not be exercised at maturity is higher the longer the time to maturity. This explains why the delta for longer times to maturity becomes flatter.

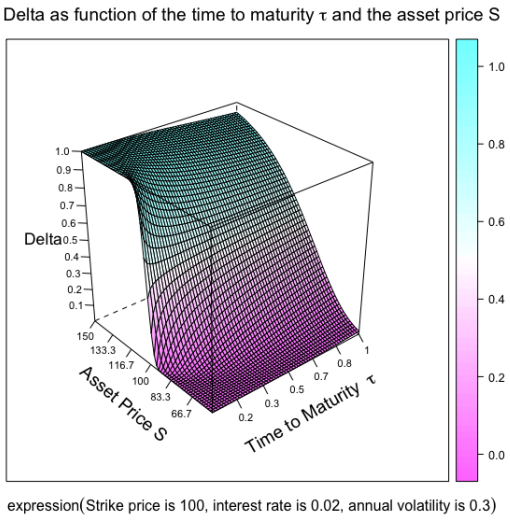

Figure 15: Delta of a Call Option

Numerical Approximation: It is possible to evaluate the Δ using the incremental ratio, which we can approximate numerically as follows,using centered derivative :

$$\frac{\partial}{\partial x}C(x,t) \sim \frac{C(t,x+h) - C(t,x-h)}{2h} \tag{9}$$

that with h= 0.01, approximate $\Delta = 0.6808008$

Monte Carlo Approach:the Monte Carlo method is useful in the evaluation of the Greeks. As usual, being a Monte Carlo method, it is always necessary to take into consideration the speed of convergence of the method.
It approximates $\Delta = 0.6729535$

Mixing Monte Carlo and numerical approximation : There are cases in which it is not possible to use the density method due to the fact that the underlying process is not the geometric Brownian motion. In this case, Monte Carlo is required to simulate paths of the underlying process and through the numerical approximation is possible to evaluate path wisely to get the delta of the option. The delta computed by this method is in this case $\Delta = 0.6780327$

3.4.2 Gamma

Gamma is the delta's sensitivity to small movements in the underlying asset price. Gamma is identical for put and call options, ceteris paribus, and is given by

$$\Gamma_{call,put} = \frac{\partial^2 C}{\partial S^2} = \frac{\partial^2 P}{\partial S^2} \tag{10}$$

Black&Scholes: The gamma of a Call is:

$$\Gamma = \frac{\phi(d_1)}{\sigma\sqrt{T-t}S} \qquad\qquad \Gamma = 0.01156413 \tag{11}$$

Since gamma is the sensitivity of the delta to the underlying it is a measure of by how much or how often a position must be rehedged in order to maintain a delta-neutral position. Although the delta also varies with time this effect is dominated by the Brownian nature of the movement in the underlying
With respect to S:

- Most sensitive to movements in stock prices are at-the-money options with a short time to maturity.

Monte Carlo Approach: Also in this case it is possible to approximates Gamma with MC methods but the problem on speed of convergence still remain.

$\Gamma = 0.01201843$

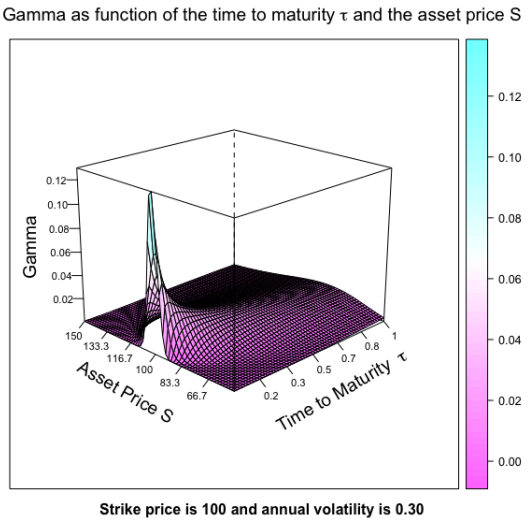

Figure 16: Gamma of a Call Option

3.4.3 Vega

Vega, also known as kappa, is the option's sensitivity to a small change in the implied volatility. Vega is equal for put and call options.

$$Vega_{call,put} = \frac{\partial C}{\partial \sigma} = \frac{\partial P}{\partial \sigma} \tag{12}$$

Black–Scholes' approach proceeds from the assumption of a constant volatility . The appearance of smiles indicates that this assumption does not hold in practice. Therefore, it can be useful to make the portfolio value insensitive to changes in volatility.

Black&Scholes: The Vega of a Call is:

$$Vega = \sqrt{(T-t)} * S * \phi(d_1) \qquad\qquad Vega = 26.91265 \tag{13}$$

With respect to S:

- the stock price approaches the strike price, vega increases and reaches its peak when the option becomes at-the-money
- if the option becomes out-of-the-money, the sensitivity of the option with respect to volatility is low again.

With respect to T:

- at-the-money options with a long time to maturity are most sensitive to changes in volatility

In case of Vanilla options, an options' buyer (being Call or Put) has always a positive Vega; which means that, by the increasing of volatility, the options' buyer always gains. Obviously, a Vanilla options' seller has always a negative Vega.

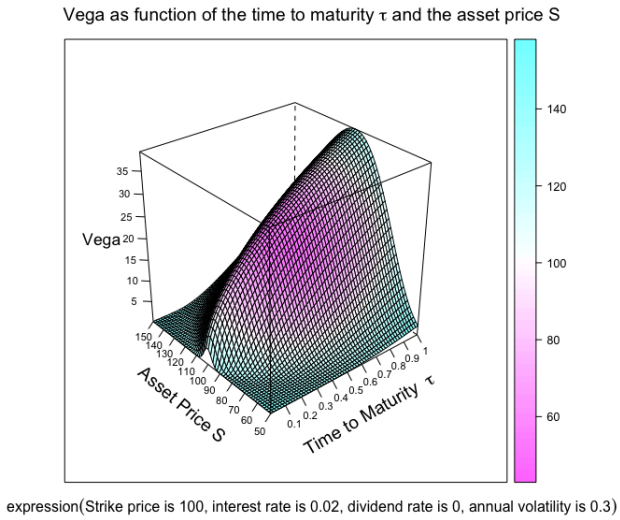

Figure 17: Vega of Vanilla Option

Monte Carlo Approach: $Vega_{MC} = 25.46074$

3.4.4 Theta

Theta, is the rate of change of the option price with time. It is related to the option value, the delta and the gammma by the Black-Scholes equation. Contrary to the evolution of the stock price the expiry of time is deterministic, and time does not involve any risk increasing the randomness The parameter Θ is for most options negative, i.e. the option value decreases as the maturity date approaches.

$$\Theta = -\frac{\partial C}{\partial T} \tag{14}$$

Black&Scholes: The Theta of a Call is:

$$\Theta = -rKe^{-r(T-t)}\Phi(d_2) - \frac{\sigma S}{2\sqrt{T-t}}\phi(d_1) \qquad \Theta = -12.62243 \tag{15}$$

With respect to S:

- When the underlying price is very low, Theta is close to zero.

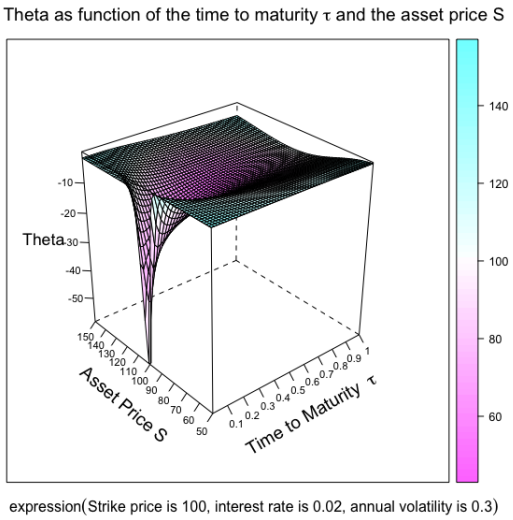

Figure 18: Theta of Call Option

Monte Carlo Approach: $Theta_{MC} = -13.55899$

3.4.5 Rho

Rho is the option's sensitivity to a small change in the risk-free interest rate. The call option's risk associated with movements in interest rates can be reduced by using rho to hedge the position

$$\rho = -\frac{\partial C}{\partial r} \tag{16}$$

Black&Scholes: The Rho of a Call is:

$$\rho = K(T-t)e^{-r(T-t)}\Phi(d_2) \qquad \rho = 27.2847 \tag{17}$$

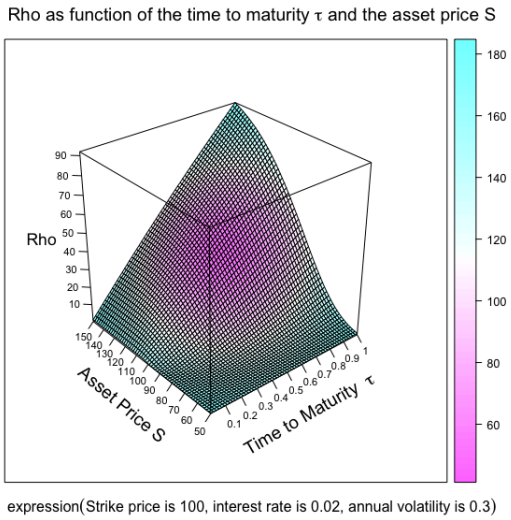

Figure 19: Rho of Call Option

3.4.6 Volga

Also known as Vega convexity,Volga, is the sensitivity of Vega to changes in implied volatility. Options far out-of-the money have the highest Vomma. If the option approaches at-the-money, volga becomes small, i.e., vega changes slowly. Consequently, the adjustments to keep a portfolio vega neutral need to be made relatively infrequently. On the contrary, if the option approaches in-the-money or out-of-the-money, volga becomes high, i.e., if stock price approaches strike price, the behaviour of vega is unstable

$$Volga = \frac{\partial^2 C}{\partial \sigma^2} \tag{18}$$

Black&Scholes: The Volga of a Call is:

$$\rho = S\sqrt{T-t}e^{-q(T-t)}\phi(d_1)d_1d_2 \qquad\qquad Volga = 0.05271008 \tag{19}$$

3.4.7 Vanna

The sensitivity of vega with respect to the stock price is given by vanna:

$$Vanna = \frac{\partial^2 C}{\partial \sigma \partial S} \tag{20}$$

Vanna approximates by how much your delta will change for a small change in the volatility, as well as how much your vega will change with a small change in the asset price. Vanna is positive for low strike options and negative for high strike options. At very high or low strikes vanna starts to tend towards to zero, more rapidly for short term option. Longer term options can have quite

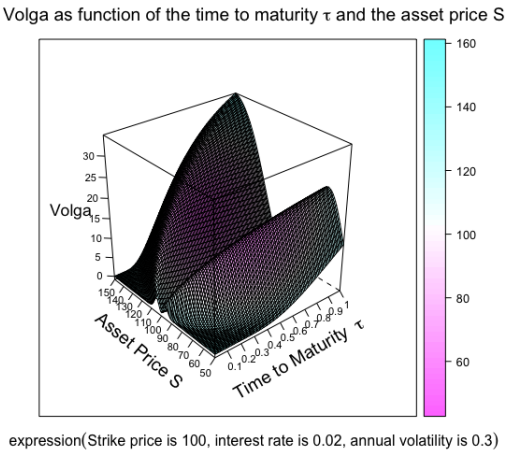

Figure 20: Volga of Call Option

pronounced vanna even when deep ITM or deep OTM

Black&Scholes: The Vanna of a Call is:

$$Vanna = \sqrt{T-t}e^{-q(T-t)}\phi(d_1)\frac{d_2}{\sigma} \qquad Vanna = 0.1121636 \tag{21}$$

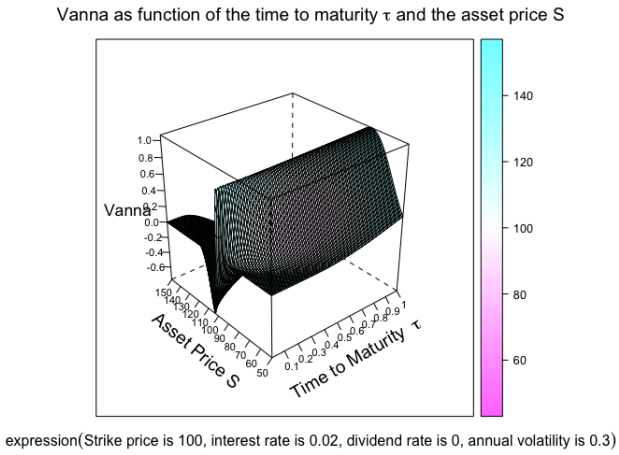

Figure 21: Vanna of Call Option

3.5 GARCH Models

A characteristic feature of financial series is that there is no evidence of autocorrelation in the return series while the absolute return shows a clear first-order autocorrelation pattern: days with large returns (in absolute value) tend to be followed by other days with large movements. The same holds true for days with small returns. This feature is named "volatility clustering". The empirical observations show that the distribution of returns exhibit features that strongly diverge from the classical hypothesis of independence and normality

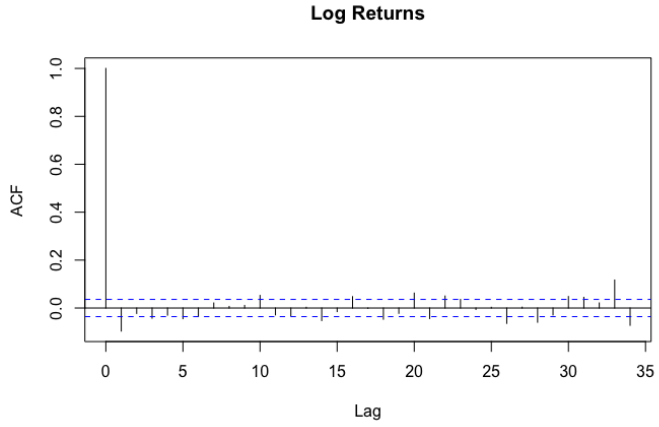

Figure 22: Autocorrelation Log Returns

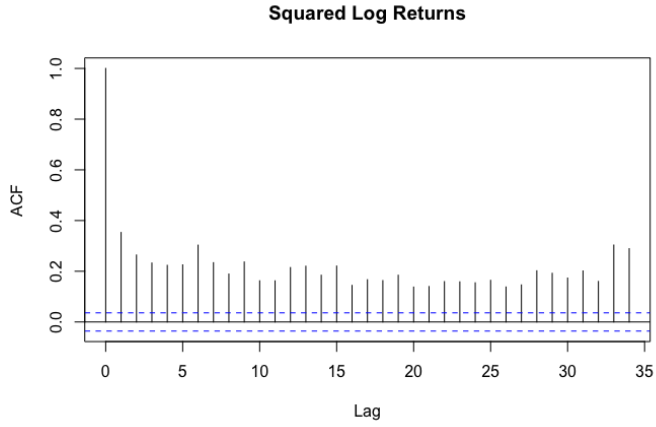

Figure 23: Autocorrelation Squared Log Returns

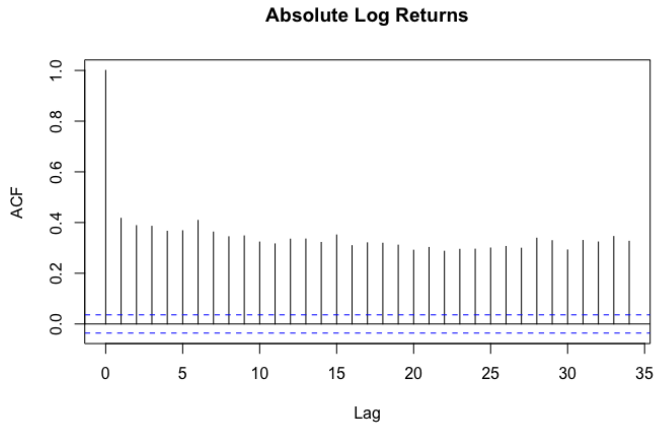

Figure 24: Autocorrelation Absolute Log Returns

As also confirmed by Box-Ljung test which reject the null hypothesis.

Box-Ljung test at lag 10	p-value
Squared Returns	$<$ 2.2e-16
Absolute Returns	$<$ 2.2e-16

Table 5: Box-Liung test

So we need better time series models if we want to model the nonconstant volatility. Models for conditional variances are often called variance function models. The GARCH models of this section are an important class of variance function models. The GARCH(p,q) models involve the estimation of volatility based on past observations.

3.5.1 GARCH(p,q)

The GARCH model is a generalized version of the ARCH model. satisfies the following system of equations:

$$\begin{cases} x_t = \sigma_t \epsilon_t \\ \sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_j x_{t-i}^2 + \sum_{j=1}^{q} \beta_i x_{t-j}^2 \end{cases}$$

where $x_t = r_t - \mu_t$ where r_t is the log return seris and μ_t the conditional mean. the conditional volatility process is determined linearly by its own lagged values and the lagged squared observations.

GARCH (1,1) model captures autoregression in volatility (volatility clustering) and leptokurtic asset return distributions.

GARCH(1,1)	**Coefficients**	**Std. Error**	**p-value**
μ	0.000870	0.000266	0.001054
ω	0.000006	0.000002	0.010593
α_1	0.099584	0.011370	0.000000
β_1	0.886392	0.014296	0.000000

Table 6: Garch(1,1) estimation

For the GARCH(1,1) model, every single coefficient is statistically significant at 1 % at least. However,the main drawback of simple GARCH is that it is symmetric, and cannot capture asymmetries in distributions and leverage effects,thus we need more advanced models.

News impact curves (Pagan and Schwert,1990) are tools to visualize the magnitude of volatility changes in response to shocks. The name comes from the interpretation of shocks as news influencing the market movements and the asymmetric effects in volatility.

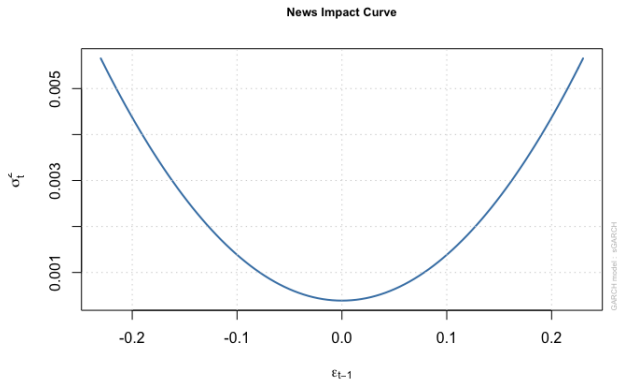

Figure 25: News Impact Curve Garch(1,1)

As clear in Figure 25 , no asymmetries are present in response to positive and negative shocks.

3.5.2 EGarch

Exponential GARCH models were introduced by Nelson (1991) and approach directly models the logarithm of the conditional volatility.

$$\begin{cases} x_t = \sigma_t \epsilon_t \\ log(\sigma_t^2) = \alpha_0 + \sum_{i=1}^{p} (\alpha_j \epsilon_{t-i} + \gamma(|\epsilon_{t-i}| - E|\epsilon_{t-i}|) + \sum_{j=1}^{q} \beta_j log(\sigma_{t-j}^2) \end{cases}$$

EGARCH(1,1)	**Coefficients**	**Std. Error**	**p-value**
μ	0.000326	0.000254	0.19909
ω	-0.138359	0.013637	0.00000
α_1	-0.095602	0.010467	0.00000
β_1	0.981847	0.001667	0.00000
γ	0.182918	0.016214	0.00000

Table 7: EGarch(1,1) estimation

The EGARCH EGARCH model does not require any restriction on the parameters because, since the equation is on log variance instead of variance itself, the positivity of the variance is automatically satisfied. The main advantage of the EGARCH model is to capture asymmetry: it is the empirically observed that negative shocks at time t_{-1} have a stronger impact in the variance at time t than positive shocks. This asymmetry is captured by the α_i parameter: a negative value indicates that the process reacts more to negative shocks, as observable in real data sets.

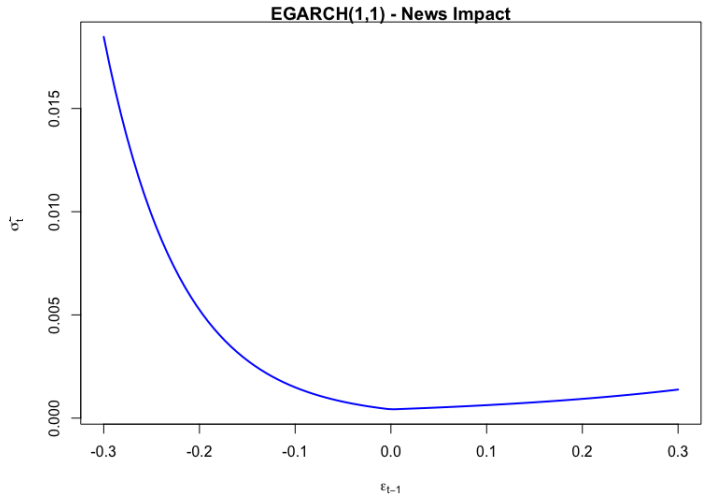

Figure 26: News Impact Curve EGarch(1,1)

News impact curve reflects the strong asymmetry in response of conditional volatility to shocks and confirms the necessity of asymmetric models.

3.5.3 Threshold Garch

Another prominent example is the TGARCH model, which is even easier to interpret. The TGARCH specification involves an explicit distinction of model parameters above and below a certain threshold.

$$\begin{cases} x_t = \sigma_t \epsilon_t \\ \sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} (\alpha_i + \gamma_i I_{t-i}) x_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \end{cases} \qquad I_{t-i} = \begin{cases} 1 \; if \; x_{t-1} < 0 \\ 0 \; if \; x_{t-1} \geq 0 \end{cases}$$

The interpretation is straightforward: if γ_i is positive, a negative error term will have a higher impact on the conditional volatility, as it is estimated from the dataset.

TGARCH(1,1)	**Coefficients**	**Std. Error**	**p-value**
μ	0.000170	0.000220	0.43942
ω	0.000424	0.000073	0.00000
α_1	0.102118	0.011973	0.00000
β_1	0.900413	0.011819	0.00000
γ	0.613472	0.069559	0.00000

Table 8: TGarch(1,1) estimation

Thanks to the specific functional form, the news impact curve for a Threshold-GARCH is less flexible in representing different responses, there is a kink at the zero point.

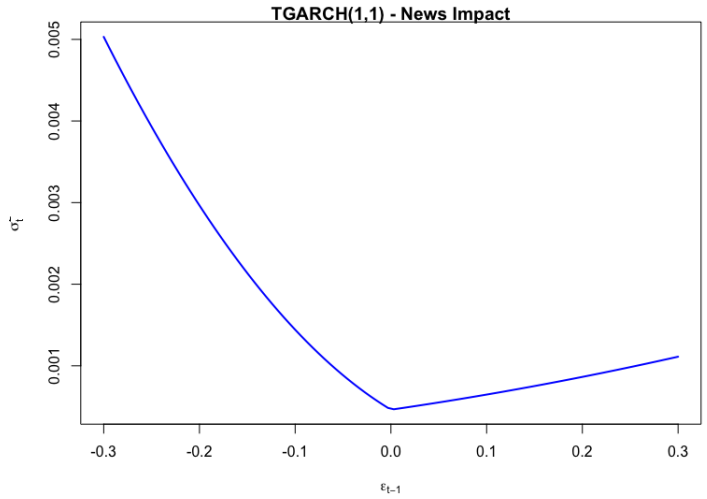

Figure 27: News Impact Curve TGarch(1,1)

Analysing residuals through the Ljung Box test, all the p-values are > 0.05, indicating that there is no evidence of serial correlation in the squared residuals and hence, and they behave as white noise process.

From the E-Garch and T-Garch estimation it has been proof that leverage effect exists: the corrisponding parameters in TGARCH and EGARCH models are statistically significant and,thus, negative shocks increase the volatility more than positive shocks.

3.6 Lévy Processes

3.6.1 Introduction to Lévy processes

One of the key assumptions of the Black-Scholes model is that the value of a risky asset should be modelled by an geometric Brownian Motion. It has long been known that this assumption drastically fails to match the reality of observed data: time series of log returns exhibit leptokurtic distributions which are inconsistent with the Gaussian distribution postulated by the GBM. Also, the volatility exhibits clustering and leverage effects, which contradict the random-walk property of a gBM.

In order to incorporate the stylized properties of asset prices is necessary to replace the Brownian motion W with another related process such as a Lévy process. Lévy models are able to incorporate several stylized features of asset prices such as heavy tails, high-kurtosis, and asymmetry of log returns. Moreover, they are capable of incorporating jumps in the dynamics of the stock prices. According to the a real valued Lévy process followed by Lévy-Khintchine characterization.

Definition 3.1 *A Lévy process $X = \{X_t : t \geq 0\}$ is a stochastic process defined on a probability space (Ω, F, P) which statisfies the following properties:*
(i) The paths of X are right continuous with left limits almost surely.
(ii) $X_0 = 0 almostsurely$
(iii) X has independet increments: for $0 \leq s \leq t$, $X_t - X_s$ is independent of $sigma(X_u : u \leq s)$
(iv) X has stationary increments;for $0 \leq s \leq t$, $X_t - X_s$ is equal in distribution to X_{t-s}

Among the better known models are the variance Gamma model of Carr et al. (1998).

3.6.2 Variance Gamma model

The VG distribution is a special case of the generalized hyperbolic (GH) distribution. The variance-gamma has an infinite activity, that is it is characterized by infinite number of jumps of small sizes in a finite time period. At the same time, The VG has a finite variation, so it is characterized by a finite number of jumps of big size in a finite time period. Variance-gamma process belongs to the time-changed Brownian motions.
Let

$$B_t(\theta, \sigma) = \theta_t + \sigma W_t \tag{22}$$

where W_t is a standard Brownian motion.
The Gamma process $\gamma(t, v)$ might be defined with independent gamma increments over intervals of length h and variance rate vh
The three parameter VG process $\mathrm{X}(t, \theta, \sigma, v)$ is defined by

$$X(t, \theta, \sigma, v) = B(\gamma(t, 1, v), \theta, \sigma) \tag{23}$$

It can be seen that the process X(t) is a Brownian motion with drift evaluated at a gamma time change.
Assuming an exponential Levy process for the Stock price

$$S_t = S_0 e^{rt + Z_t + \omega t} \qquad \omega = \frac{1}{v} log(1 - \theta v - \frac{\sigma^2 v}{2}) \tag{24}$$

where ω is the compensator which ensures a martingale property. The characteristic function of the logarithm of the process S_t is the following:

$$\phi_t(u) = exp(log(S_0 + (r + \omega)t)(1 - i\theta v u + \sigma^2 u^2 \frac{v}{2})^{-\frac{t}{v}}) \tag{25}$$

By definition of risk neutrality, the price of a European call option with strike K and maturity T is

3.6.3 Parameters Estimation

Nelder-Mead method
Estimating the parameters of a Variance-Gamma can be particularly demanding, due to its being a pure jump process with the characteristics above mentioned.

The Nelder-Mead method in order to optimize the likelihood function and to find the values for the above listed parameters), is one of the best known algorithms for multidimensional unconstrained optimization without derivatives. This method does not require any derivative information, which makes it suitable for problems with non-smooth functions. It is widely used to solve parameter estimation and similar statistical problems, where the function values are uncertain or subject to noise.

Quasi-Newton BFGS method
Another consistent method to estimate the parameters. Through the use of Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm for solving unconstrained nonlinear optimization problems.

	vgC	σ	θ	**v**
Nelder-Mead Method	-4.587454e-09	2.274994e-02	3.057779e-04	1.413375e+00
Quasi-Newton BFGS method	-0.0002045	0.0229675	0.0005034	1.4365213

Table 9: VG parameters estimation

3.6.4 Pricing FFT and Monte Carlo

FFT Method The Call price got using the Variance Gamma assumption and parameters obtained through Nelder-Mead Method is

$$Call = 9.662336 \tag{26}$$

while the price with parameters of BFGS methods is :

$$Call = 9.664084 \tag{27}$$

Monte Carlo Method Starting from the assumption that the dynamics of stock price follows an exponential Lévy process, it is possible to apply the monte carlo methods to simulate different paths and to price the option as usual. According to this method, the option value got is :

$$Call = 9.644712 \tag{28}$$

3.7 American Options

Pricing American options is a more complex task than for European options since they can be exercised any time up to expiry. Notice that the value of an American call is always higher than the value of the corresponding European option. Exact formulas do not exist but several approximation techniques have been implemented. In the following parts it will be provided the Finite-Difference Method (explicit and implicit) and the Monte Carlo and approximation method.

3.7.1 Explicit finite-difference Method

Assuming a maturity time T. Thus it is possible to create shorter interval with the same length Δ t and define Δ S = $S_{max} - S_{min}$. Now it is possible to define

$$\begin{aligned} C_{i,j} &= C(t_i, x_j) \\ &= C(i\Delta t, S_{min} + j\Delta S) \end{aligned}$$

and use different approximations of the partial derivatives

$$\begin{aligned} \frac{\partial}{\partial x} C(x,t) &= \frac{C_{i,j+1} - C_{i,j}}{\Delta S} \\ \frac{\partial}{\partial x} C(x,t) &= \frac{C_{i,j} - C_{i,j-1}}{\Delta S} \\ \frac{\partial^2}{\partial x^2} C(x,t) &= \frac{C_{i,j+1} + C_{i,j-1} - 2C_{i,j}}{\Delta^2 S} \end{aligned}$$

while for the time derivative

$$\frac{\partial}{\partial t} C(x,t) = \frac{C_{i+1,j} - C_{i,j}}{\Delta t}$$

which replaced in the Black Schole Equation :

$$r_j C_i, j = \frac{C_{i+1,j} - C_{i,j}}{\Delta t} + rj\Delta S \frac{C_{i,j+1} - C_{i,j-1}}{2\Delta S} + \frac{1}{2}\sigma^2 j^2 (\Delta S)^2 \frac{C_{i,j+1} + C_{i,j-1} - 2C_{i,j}}{2\Delta^2 S} \tag{29}$$

It can be rewritten as follows:

$$C_{i,j} = a_j^* C_{i+1,j-1} + b_j^* C_{i+1,j} + c_j * C_{i+1,j+1} \tag{30}$$

where

$$\begin{aligned} a_j^* &= \frac{1}{1 + r\Delta t}(-\frac{1}{2} rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t) \\ b_j^* &= \frac{1}{1 + r\Delta t}(1 - \sigma^2 j^2 \Delta t) \\ c_j^* &= -\frac{1}{1 + r\Delta t}(\frac{1}{2} rj\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t) \end{aligned}$$

Thus, payoff is given by:

$$C_{N,j} = max(K - j\Delta S, 0) \qquad j = 1...M \tag{31}$$

- If the value $C_{N-1,j}$ is smaller than $C_{N,j}$, the option would be exercised.
- If at time t= (N-1)Δt, $S_t = S_{min} + \Delta Sj$ and $C_{N-1,j} < C_{N,j}$, exercising the american put would not be optimal and would be convenient to keep contract alive.

The following Figure 28 shows the computation of this method for a option with an S_{max}= 65, S_{min} = 0, K = 60, T= 1 year and σ = 0.4100527. The following figure reports the computations and it is convenient to keep the option until the price S_t crosses the frontier marked as a dashed red line.

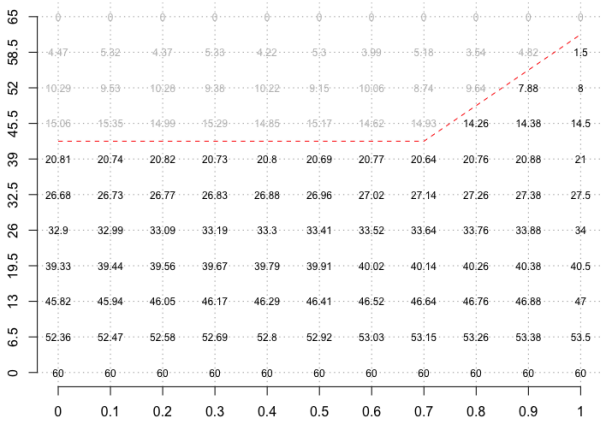

Figure 28: American Put Explicit FD method

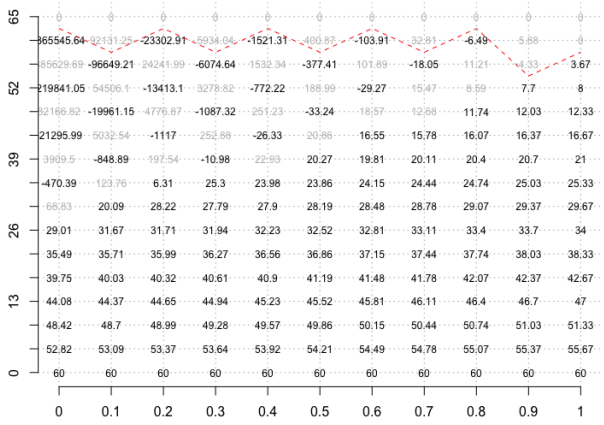

Figure 29: Am.Opt Explicit FD method Instability

This method shows instability, providing unrealistic results. This happens when :

$$rj > \sigma^2 j^2 \quad and\ or \quad \sigma^2 j^2 \Delta t > 1$$

and the empirical evidence is reported in Figure 29

3.7.2 Implicit finite-difference Method

The implicit method allows to avoid this instability. Defining the derivatives as follow:

$$\frac{\partial}{\partial t}C(x,t) = \frac{C_{i+1,j} - C_{i,j}}{\Delta t}$$
$$\frac{\partial}{\partial x}C(x,t) = \frac{C_{i,j+1} - C_{i,j-1}}{2\Delta S}$$
$$\frac{\partial^2}{\partial x^2}C(x,t) = \frac{C_{i,j+1} + C_{i,j-1} - 2C_{i,j}}{\Delta S^2}$$

which replaced in Black Scholes equation:

$$a_j C_{i,j-1} + b_j C_{i,j} + c_j C_{i,j+1} = C_{i+1,j} \tag{32}$$

where

$$a_j = \frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$
$$b_j = 1 + \sigma^2 j^2 \Delta t + r\Delta t$$
$$c_j = -\frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$

It is now possible to implement the implicit method for the option with the same parameters defined for the explicit method. In Figure 31 is computed the option with parameters equal to those previously generated instability, and it appears clearly how this method does not generate anomalous values.

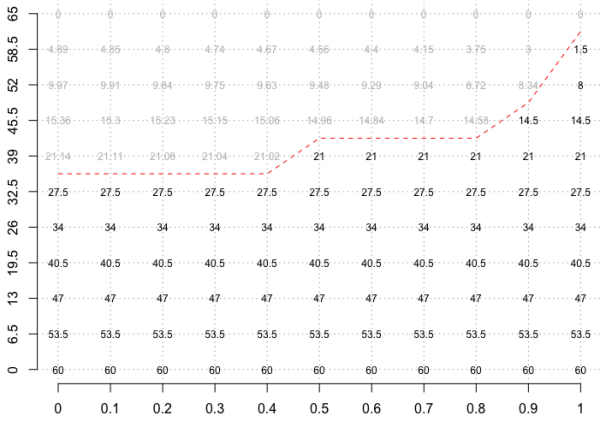

Figure 30: American Put IFD method

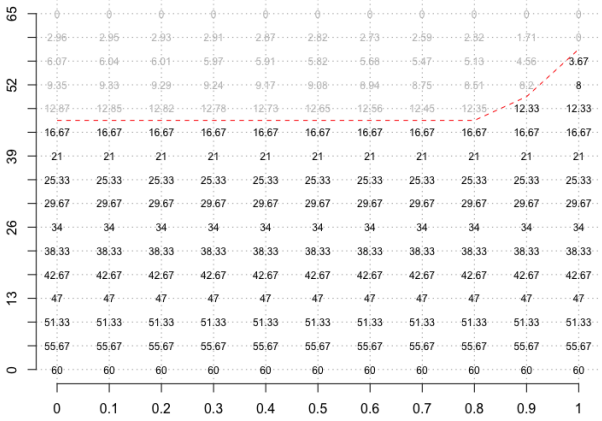

Figure 31: American Put IFD- Stabilty

3.7.3 Broadies and Glasserman simulation method

Given a time horizon $[0, T]$, this method aims to simulate for $\tau \leq T$ the stock price $S_1 = S_{t,1}$, $S_2 = S_{t,2}, ..., S_T$ starting from the S_0 and compute the option price using:

$$C = max_{i=0...d}\, \mathbb{E}\ e^{-rt_i} max(S_i - K, 0) \tag{33}$$

Simulating the stock dynamics as a tree with b new branches at each t, it is possible to estimate the upper Θ and lower θ estimator of the option prices. These estimator are both biased but asymptotically unbiased, thanks to the Monte Carlo simulations and number of branches in the tree. This allow to identify a confidence interval for the real price of the American Option.

3.7.4 Longstaff and Schwartz least squares method

Longstaff and Schwartz (2001) combine Monte Carlo Simulation with the Least Squares method by using backwards induction to approximate conditional continuation values to price American

N simulation	Θ	θ
100	14.77693	12.37267
1000	15.95794	13.35016
1e5	15.68291	13.13072

Table 10: Monte Carlo with different M

options. The main idea in their least-squares Monte Carlo (LSM) approach is the usage of least squares to estimate conditional continuation values to decide to exercise the option or not. This method requires a simulation of a single path, rather than the construction of a tree, on a grid of times ti, i = 0,1,...,d. Let $V_i(x)$ and $f_i(x)$ denote the price of the option and its related payoff function at time t_i.
The continuation value at time t_i in state S_{t_i} = x is

$$C_i(x) = \mathbb{E}\{V_{i+1}(S_{t_i+1)|S_{t_i}=x} = \sum_{r=1}^{M} \beta_{ir} v_r(x) \tag{34}$$

The coefficients could be estimated by simple regression. The LSM algorithm can be summarized as follows:

- simulate n independent paths for time steps
- at t= T set $\hat{V_{tj}} = f_d\ (S_{t_j})$, j = 1...,n
- for i = d-1, ..., 1
  - specify the I of paths in the money
  - discount the value $\hat{V}_{i+1,j} \in$ to input in the regression
  - given the estimates, run regression to get $\hat{\beta}_j$
  - estimate the continuation values
  - if $f_i\ ((S_i^j) > \hat{C}_i(S_i^j)$ set $\hat{V_{ij}}(S_i^j) = f_i(S_i^j$ else $\hat{V_{ij}} = \hat{V_{i+1,j}}$
- calculate $\hat{V_{1,}} + ... + \hat{V_{1,n}}$ and discount to get $\hat{V_0}$

Thus, for S=K=100, r=0.0199,sigma= 0.4100527

N simulation	American Put	Error	European Put
100	14.84718	1.203884	14.3016
1000	15.26234	0.3861621	14.94291
1e5	15.21921	0.03870776	15.12201

Table 11: LSM Method

3.8 Multi-Assets Options

The value of an option on multiple underlyings depends not only on the value of the single underlying but also on their relation. For this reason, pricing these kinds of options (which belongs to the exotic) is complicated.

3.8.1 Multi-dimensional Geometric Brownian Motion

The basic building block for option pricing with multiple underlyings is to restore the assumption on geometric Brownian motion and extend it to multiple assets.

$$dS_i = \mu_i S_i(t)dt + \sigma_i S_i(t) \sum_{j=1}^{m} \sigma_{ij} dB_j(t) \tag{35}$$

With the explicit solution:

$$S_i(t) = \ S_i(0) exp\{(\mu_i \frac{1}{2} \sum_{j=1}^{m} \sigma_{i,j}^2) + \sum_{j=1}^{m} \sigma_{ij} B_j(t)\} \tag{36}$$

for each process i= 1...n.

Considering the case of 2 assets with 2 independent Brownian Motions

$$dS_1 = \mu_1 S_1(t)dt + S_1(t)(\sigma_{11} dB_1(t) + \sigma_{12} dB_2(t)$$
$$dS_2 = \mu_2 S_2(t)dt + S_2(t)(\sigma_{21} dB_1(t) + \sigma_{22} dB_2(t)$$

Under this assumption, it is possible to price a basket option based on two assets: JP Morgan Stock and Microsoft Stock.

In contrast to normal random variables, a linear combination of lognormal variables is, however, not lognormal any more. Therefore, the Black-Scholes model does not hold exactly for the basket value Bt, and we have to use approximations or numerical procedures for pricing basket options. The payoff of A best-of call is:

$$X = max(max(S_1(T), S_2(T)) - K, 0) \tag{37}$$

and

$$Price_t = e^{-r(T-t)} \, \mathbb{E}_Q(X|F_t) \tag{38}$$

Given that $S_1 = 108.72$ $S_2 = 137.78$ T=1 year, r= 0.199, K =150. By numerical pricing, the value of the option under the up-stated assumption is:

$$P_t = 8.567221 \tag{39}$$

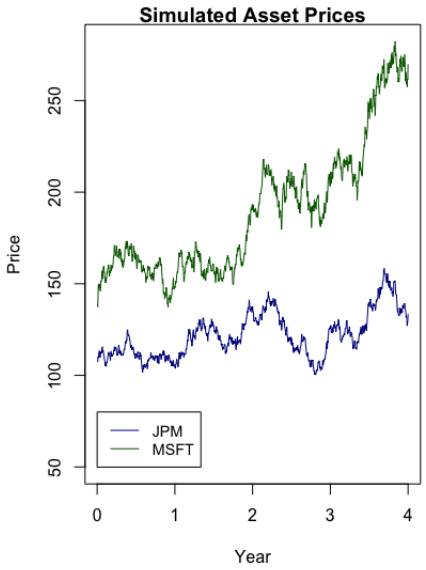

Figure 32: Simulated gBM prices

In pricing a derivative, it is necessary take into account the relationship between the 2 assets.

3.8.2 Copula Method

Empirical evidence on linear dependence is barely verified and an alternative model with more flexible dependence structure and arbitrary marginal distributions is needed. Copula method allows to deal with it and avoid linear correlation which results to be poor to model real data. The flexibility of copulae basically follows from Sklar's Theorem, which says that each joint distribution can be "decomposed" into its marginal distributions and a copula C "responsible" for the dependence structure.

$$F(x, y) = C(F_1(x), F_2(y)) \tag{40}$$

Copulae can separate marginal behaviour, as represented by the F_i from the association. For this reason, according to Deheuvels (1979) copulae are called also dependence functions.

Kendall Tau:

Definition 3.2 *Given random variable X and Y, the Kendall correlation coefficient τ between X and Y is defined as*

$\tau(X, Y) = p_c - p_d$

where, for any two independet pairs of values (X_i, Y_i),(X_j, Y_j) from (X, Y)

$p_c = P((X_j, X_i)(Y_j, Y_i) > 0)$ *and* $p_d = P((X_j, X_i)(Y_j, Y_i) < 0)$

p_c and p_d are the probabilities of concordance and discordance, respectively.

Kendall Tau is related to copulae since, in coupling a joint distribution function with its marginals, the copula captures certain aspects of the relationship between the variates, from which it follows that dependence concepts are properties of the copula. It is non-parametric measures since it does not care about the shape of dependence at all. The estimated Kendall sample tau is

$$\tau = 0.3272396$$

Notice that Gaussian and Elliptical copula are not so different from linear correlation, thus they may not be good and next section focuses on Archimean Copula family.

Archimean Copula: Archimean Copula family of copulae is worth studying for a number of reasons: many interesting parametric families of copulae are Archimedean, the class of Archimedean copulae allow for a great variety of different dependence structures and they have closed form expressions. The main drawback is that Archimedian Copulae depends only on a parameter and this may generate some issues when dealing with higher-dimension than bivariate copulae.

Gumbel (1960)	
$\phi_\alpha(t)$	$(-\ln t)^\alpha$
range for α	$[1, +\infty)$
$C(v, z)$	$\exp\{-[(-\ln v)^\alpha + (-\ln z)^\alpha]^{1/\alpha}\}$
Clayton (1978)	
$\phi_\alpha(t)$	$\frac{1}{\alpha}(t^{-\alpha} - 1)$
range for α	$[-1, 0) \cup (0, +\infty)$
$C(v, z)$	$\max[(v^{-\alpha} + z^{-\alpha} - 1)^{-1/\alpha}, 0]$
Frank (1979)	
$\phi_\alpha(t)$	$-\ln\frac{\exp(-\alpha t)-1}{\exp(-\alpha)-1}$
range for α	$(-\infty, 0) \cup (0, +\infty)$
$C(v, z)$	$-\frac{1}{\alpha}\ln\left(1 + \frac{(\exp(-\alpha v)-1)(\exp(-\alpha z)-1)}{\exp(-\alpha)-1}\right)$

Figure 33: Archimedean copulas - Copula Methods in Finace (Cherubini,Luciano)

The relationship of Archimedean Copula with the Kendall's Tau is :

Family	Kendall's τ
Gumbel (1960)	$1 - \alpha^{-1}$
Clayton (1978)	$\alpha/(\alpha + 2)$
Frank (1979)	$1 + 4\left[D_1(\alpha) - 1\right]/\alpha$

Figure 34: Archimedean Copulae

Archimedean copulae give the benefit of an analytical solution to the joint cumulative distribution function(copula function) but model co-dependence of entities using the same association

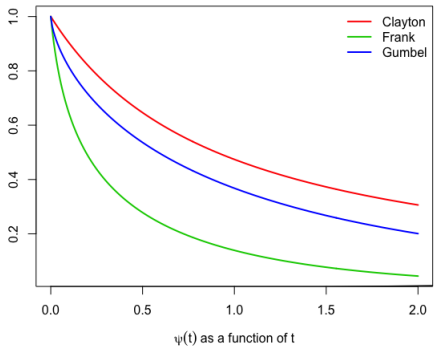

Figure 35: Graphs of the Archimedean generators

parameter,i.e. it is assumed that the correlation is homogeneous among all assets. The main feature of Clayton copula is that it stresses the left tail, while Gumbel copula stresses the right tail. Frank copula stresses well right and left tail in the same way with a lighter body.

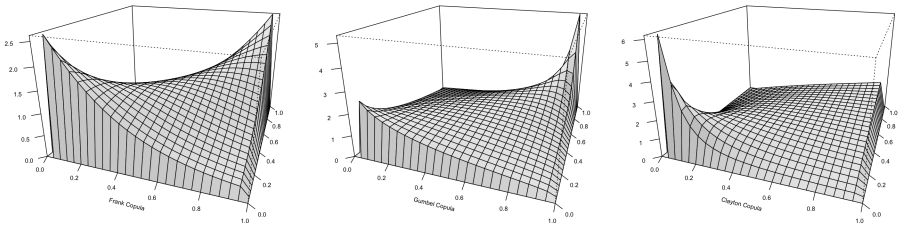

Figure 36: Archimedean Copulae

As above stated, Clayton copula stress more the left tail and this characteristic might be useful in modelling our asset dependence as they belongs to two different sectors but to the same nation, USA. It is reasonable to assume that the dependence of two assets gets stronger during an economic crisis or negative business cycles that the nation is experiencing.
Modelling the assets prices dynamics as a gBM under the Clayton Copula structure, it is possible to simulate stock prices and apply the numerical pricing to get the exotic derivative according to

the payoff function.
From here it will be computed the price for a basket option, best of option and worst of option which payoff is:

$$P_{basket} = max\{mean(\frac{S_{1t}}{S_{10}}, \frac{S_{2t}}{S_{20}}) - K\} \tag{41}$$

$$P_{best.of} = max\{max(\frac{S_{1t}}{S_{10}}, \frac{S_{2t}}{S_{20}}) - K\} \tag{42}$$

$$P_{worst.of} = max\{min(\frac{S_{1t}}{S_{10}}, \frac{S_{2t}}{S_{20}}) - K\} \tag{43}$$

These options are evaluated under the Clayton and t Copula (df=3). Moreover, for the Clayton Copula, it has been provided two different methods: the first one using the Marshall–Olkin algorithm and conditional distribution method (CDM) which involves the inversion method (Mathieu Cambou, Marius Hofert, Christiane Lemieux, 2016).

	Clayton CMD	Clayton MO	t 3
basket	20.09657	20.09624	20.00712
worst.of	14.87578	14.87607	15.15684
best.of	25.56858	25.56763	25.28755

Table 12: Basket Prices

3.9 Neural Networks for Option Pricing

This section is just an attempt to implement deep learning to option pricing. In particular, the main objective is to show the ability of Artificial Neural Networks to 'learn' the model from the dataset.

3.9.1 Artificial Neural Networks

In the context of financial derivative pricing, Artificial neural networks (ANNs) with multiple hidden layers have become successful methods to detect patterns from a large dataset. The implementation of a ANN can be typically decompose into two separate phases: training part and test part. During the training phase, the ANN 'learn' the model from the dataset while, in the test phase, the trained ANN can be employed to predict the results. An ANN is generally constitute by three components: neurons, layers and the architecture. The architecture is a combination of different layers which are made by a defined number of neurons. Connecting the neuron of previous layers with those of the followers, it is possible to use the output signals coming from the previous step as an input signals for the next layer.
The Figure 37 provide a graphical intuition of neurons and ANN.

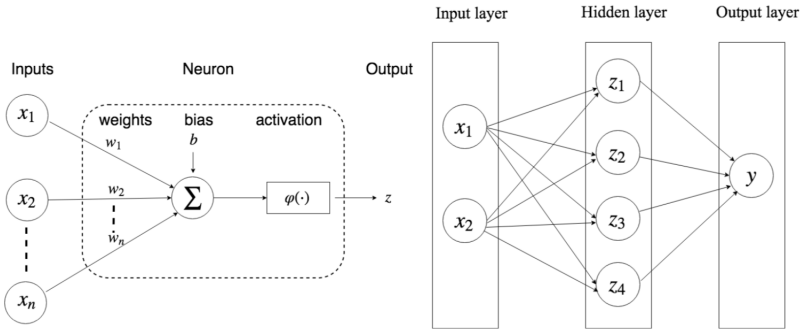

Figure 37: Simple ANN

A multi-layer perceptron is an ANN composed by three or more layers with parameters

$$\Theta = (W_1, b_1, W_2, b_2, ...W_L, b_L) \tag{44}$$

where W_l is the weight matrix of l-layer and b_l is the bias vector of the L-neural layer. Defining z_j^i as the value of the j-th neuron in the i-layer, then

$$z_j^i = \psi^l(\sum_i w_{ij} z_i^{l-1} + b_j^l) \tag{45}$$

where $z_i^{(l-1)}$ is the ouput of i-th neuron in the previous layer and $\psi()$ is the activation function. For l = 0, we have the input layer, while for l=L we get the ouptut layer. The activation function ψ () adds non linearity to the system. While there are many, to the purpouse of this section, I will used only the ReLu function,defined as follow:

$$\psi(x) = max(x, 0) \tag{46}$$

One reason to choose ReLu is that the sparse activation of the network meaning that training is faster. Secondly, ReLU activation function has high rate of success and popularity in recent ANN applications. To evaluate the performance the mean squared error (MSE) will be used.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(c_i - \overline{c})^2 \tag{47}$$

The training process try to learn the optimal weights and biases that minimize the loss function as possible.

$$arg\, min_\theta L(\theta|(x,y)) \tag{48}$$

where (x,y) are the input-ouput pairs. In order to solve the optimization problem, in this case it will be used the RMSprop optimizer. In general, the optimization alorigthm start with initial values and update parameters follows: $\begin{cases} W \leftarrow W - \eta(i)\frac{\partial L}{\partial W} \\ b \leftarrow b - \eta(i)\frac{\partial L}{\partial b} \\ i = 0, 1, 2... \end{cases}$ where η is the learning rate.

3.9.2 Methodology

To conduct the analysis, some assumptions have been made. First of all, during the analysis it has been used a dataset of simulated values. It has been assumed that the stock prices follow a gBM to get a sufficient number of stock prices. Thus, simulated values have been got to have enough strike price, maturities, interest rates and volatilities. From these parameters, it has been computed the price of an European Call Option by Black Scholes formula.Thus, it is assumed to be in a Black Scholes world, i.e. all assumption of the model are satisfied. Once that the dataset is built up, it has been split in two subset: training set and test set. The ANN has been trained on the training set and then evaluated on the test dataset. In the following table 13 are reported hyper parameters used for the ANN

Parameters	Model	Model 2
Activation	ReLu	ReLu
Dropout rate	0	0.2
Neurons	(200,130,50,1)	(200,130,50,1)
Layers	4	4
Batch-normalization	No	YES
Optimizer	RMSprop	RMSprop
Loss function	MSE	MSE
Batch size	256	256
Epochs	45	45
Automatically adjust learning rate	NO	YES

Table 13: Hyper parameters

3.9.3 Results

Starting from the above settings, the results got are reported in Table 14:

	Model 1	Model 2
Loss	0.007573	0.02409
Mean Absolute Error	0.02671	0.1036
Learning Rate	-	[1e-03 : 1e-07]

Table 14: ANN Results in training phase

During the prediction phase, the results in Table 15 are not so different from those got in during training phase (Table 14). This may suggest that overfitting problems does not represent a relevant issue in this case.

	Model 1	Model 2
Loss	0.0087899	0.004087737
Mean Absolute Error	0.02373787	0.01184842

Table 15: ANN Results in prediction phase

and the predicted result for the first three rows of the test set are:

Black & Scholes	Model 1	Model 2
57.5951102	57.4038728	57.5640321
48.8287273	48.9358415	48.9322014
42.2001232	42.0089468	42.4328429

Table 16: ANN Results

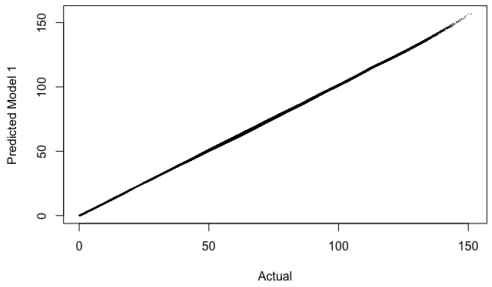

Figure 38: Actual vs Predicted - Model 1

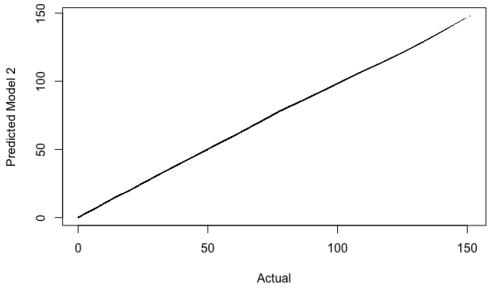

Figure 39: Actual vs Predicted - Model 2

Figure 38 and 39 plot the actual price against the predicted price of each option for both models, yielding a narrow line with very few deviations, indicating very few significant errors in pricing.

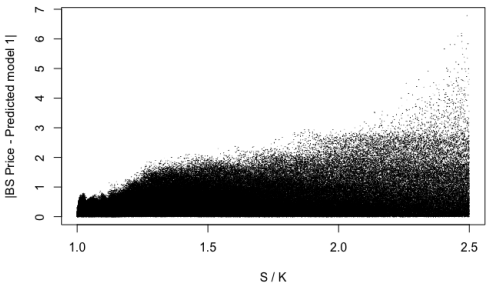

Figure 40: Abs Err Model 1 and moneyness

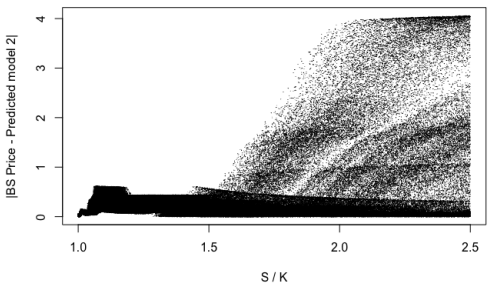

Figure 41: Abs Err Model 2 and moneynes

Figure 40 and 41 show the absolute error which models make as the option becomes ITM. Notice that the simulated dataset is made only of ATM or ITM, thus the least value of $\frac{S}{K}$ is 1. It is interesting the different behaviour that two models have as $\frac{S}{K}$ increases.
A deeper analysis in the models and fine tuning of parameters would definitely improve the results. In general, it is possible to see how, in the Black & Scholes world, the ANN provides a good method to compute the European-style derivatives.

3.10 Conclusions

To sum up, in the different parts of this paper it has been proof what is already well known: the hypothesis of gBM and Black & Scholes model are not hold in the reality. From the time series analysis it has shown as the distribution of log returns does not follow a Gaussian distribution, the volatility varies across times and it is not constant. Although this is known to anyone, Black Scholes model still plays an important role as benchmark for professionals and option traders. Moreover, nowadays, Black and Scholes Greeks are massively used for hedging strategies.

In the section dedicated to Garch models it has been shown as different econometric models have been implemented to catch the features of volatility and its asymmetric reaction to news.

To correctly model the stock price dynamics, Levy process has been introduced to allow models capture jumps. A specific section has been dedicated to the Variance Gamma model and pricing method with it.

A section has been dedicated to American Options, showing the problems related to the correct price and optimal stopping time. It has also been stated problems of Explicit Finite Difference methods and solution implemented. Then, it has been offered a short view on simulations methods for American Option pricing.

In the following section it has been shown how to pricing multi asset options. In doing that, it has clearly appeared what is the main problem in this case: multi dimensionality does not allow for a direct pricing formula and, moreover, it requires to correctly evaluate the relationship between the assets. In trying to model it correctly, Copula methods have been applied. This has allow to price the exotic derivatives in a more consistent way.

Finally, the last topic of this paper has been dedicated to implement two simple models using Artificial Neural Networks. To do so, a simulated dataset has been used and the results shows that even the two simple model proposed have been capable to price option with a good approximation level.

4 Bibliography

- Statistics and Data Analysis for Financial Engineering with R examples,David Ruppert,David S.Matteson
- Computational Finance, Argimiro Arratia,978-94-6239-069-0
- Christian Francq, Jean-Michel Zakoian - GARCH Models Structure, Statistical Inference and Financial Applications-Wiley (2019)
- Mastering R for Quantitative Finance,Edina Berlinger, Ferenc Illés... 2015 Packt Publishing
- Empirical Evidence on Student-t Log-Returns of Diversified World Stock Indices,Eckhard Platen 1 and Renata Sidorowicz,Research Paper 194, QUANTITATIVE FINANCE RESEARCH CENTRE
- THE COMPLETE GUIDE TO Option Pricing,Espen Gaarder Haug, McGraw-Hill
- Handbook of Computational Finance,Jin-Chuan Duan,Prof.Dr. Wolfgang Karl Ha rdle,Prof. James E. Gentle,Springer Handbooks of Computational Statistics
- Statistics of Financial Markets- An Introduction,Jürgen Franke, Wolfgang Karl Härdle, Christian Matthias Hafner,Springer Berlin Heidelberg 2011
- Option Pricing using Levy Processes,Yongqian Bu,G oteborg, Sweden 2007
- Option Pricing and Estimation of Financial Models with R,Stefano M. Iacus ,2011, Wiley
- Simulation and Inference for Stochastic Processes with YUIMA. A comprehensive R Framework for SDEs and other Stochastic Processes,Stefano M. Iacus, Nakahiro Yoshida, Springer (2018)
- Time Series Analysis With Applications in R- Jonathan D. Cryer, Kung-Sik Chan -Springer (2008) copy
- Analyzing Financial Data and Implementing Financial Models Using R,Clifford S. Ang, Springer
- Applied quantitative finance, Chen, Cathy Yi-Hsuan Härdle, Wolfgang Overbeck, Ludger,Springer (2017)
- - Exotic option pricing and advanced Levy models, [Wilmott Collection] Andreas Kyprianou, Wim Schoutens, Paul Wilmott (2005, Wiley).
- Quantitative Risk Management ,Alexander J. McNeil, Rudiger Frey, Paul Embrechts,Princeton University Press 2005
- Computational Finance, Numerical Methods for Pricing Financial Instruments,George Levy,Butterworth-Heinemann Elsevier
- Copula Methods in Finance,Umberto Cherubini Elisa Luciano,Walter Vecchiato, Wiley Finance Series
- Tools for Computational Finance,Rudiger U. Seydel,2017

- Applied Econometrics with R,Christian Kleiber, Achim Zeileis-Springer-Verlag New York (2008)
- Handbook in Monte Carlo Simulation Applications in Financial EngineeringRisk Managementand Economics,Paolo Brandimarte, Wiley
- Notes on Financial Risk and Analytics,N. Privault,2018
- An Introduction to Stochastic Calculus with Matlab, Ballotta, Fusai, John Wiley Sons 2015
- Financial Modelling with Jump Processes , Rama Cont, Tankov Peter, Chapman & Hall
- Financial Risk Forecasting: The Theory and Practice of Forecasting Market Risk, Jon Danielsson, Wiley 2011
- Implementing Models of Financial Derivatives Object Oriented Applications with VBA - Nick Webber, Wiley (2011)
- Nonlinear time series analysis, Chen, Rong Tsay, Ruey S John Wiley Sons (2019)
- John C. Hull-Options, Futures, and Other Derivatives-Prentice Hall, 2014
- Market Risk Analysis: Pricing, Hedging and Trading Financial Instruments, Carol Alexander (2008, Wiley)
- Analysis of Financial Time Series,RUEY S. TSAY, Wiley
- Practical C++ Financial Programming, Carlos Oliviera
- Paul Wilmott On Quantitative Finance,Paul Wilmott, Wiley Sons
- Exotic Derivatives and Deep Learning- AXEL BROSTRÖM RICHARD KRISTIANSSON
- Pricing options and computing implied volatilities using neural networks,Shuaiqiang Liu 1,*, Cornelis W. Oosterlee 1,2 and Sander M.Bohte 2
- Machine Learning in Finance:The Case of Deep Learning for Option Pricing, Robert Culkin, Sanjiv R. Das -2017
- Hedging and Pricing Options using Machine Learning, Jacob Michelsen Kolind, Jon Harris and Karol Przybytkowski,2009
- MACHINE LEARNING IN OPTION PRICING, Daniel Stafford,Master's Thesis,UNIVERSITY OF OULU,2018

5 Appendix

```
####TIME-Series Analisys####

###Download TS
library(quantmod)
library(tseries)
getSymbols('JPM', src = "yahoo", from = '2000-01-01',to='2019-08-23')
Stock<- JPM$JPM.Close #Stock
sma_50 <-TTR::SMA(Stock,n = 50)
sma_200 <-TTR::SMA(Stock,n = 200)
#Visualize Data
plot(Stock,main= "JP Morgan Stock Price",type='l')
lines(sma_50, col='red')
lines(sma_200, col='green')


chartSeries(Stock, name = "JP Morgan & Chase", TA= "addBBands()",theme = '
    white')
zoomChart("2006-10::2008-10")


#Log Return
S_log_ret <- na.remove(diff(log(Stock))) #na.remove from tseries


mean_S_log <- round(mean(S_log_ret), 2)
sd_S_log <- round(sd(S_log_ret), 2)
plot(S_log_ret,type='l', ylab='Log Returns')


#####Change point detection####
require(sde)
library(tseries)
S <- get.hist.quote("JPM", start = "2000-01-01", end = "2019-08-23")
chartSeries(S, name = "JP Morgan & Chase", TA= "addBBands()",theme = 'white
    ')
S <- S$Close
cpoint(S)
addVLine = function(dtlist) plot(addTA(xts(rep(TRUE, NROW(dtlist)), dtlist)
    , on = 1, col = "#FF0000"))
addVLine(cpoint(S)$tau0)
S <- as.numeric(S)
n <- length(S)
X <- log(S[-1]/S[-n])



plot(X, type = "l", main = "Log Returns")
abline(v = cpoint(X)$tau0, lty = 3, col = "red")
```

```
####POLISHED Dataset####

getSymbols('JPM', src = "yahoo", from = '2007-10-31',to='2019-08-23')
Stock<- JPM$JPM.Close #Stock
chartSeries(Stock, name = "JP Morgan & Chase", TA= "addBBands()",theme = '
    white')

S <- get.hist.quote("JPM", start = "2007-10-31", end = "2019-08-23")
S <- na.remove(S$Close)
n <- length(S)
X <- na.omit(diff(log(S)))



S_log_ret <- na.remove(diff(log(Stock))) #na.remove from tseries

mean_S_log <- round(mean(S_log_ret), 8)
sd_S_log <- round(sd(S_log_ret), 8)
plot(S_log_ret,type='l', ylab='Log Returns')


# Plot the histogram along with a legend
hist(S_log_ret, breaks = 100, prob=T, cex.main = 0.9)
abline(v = mean_S_log, lwd = 2, col='red')
legend("topright", cex = 0.8, border = NULL, bty = "n",
       paste("mean=", mean_S_log, "; sd=", sd_S_log)) ##ADD GREENLEGEND



x <- seq(-5 * sd_S_log, 5 * sd_S_log, length = nrow(S_log_ret))
lines(x, dnorm(x, mean_S_log, sd_S_log), col = "green", lwd = 2)
legend("topleft", legend=c("Gaussian distribution"),col=c('green'), lty=1:2
    , cex=0.8)

# Right Tail zoom with respect Normal dist
plot(density(S_log_ret), main='Return EDF - upper tail', xlim = c(0.1, 0.2)
    ,
     ylim=c(0,2));
curve(dnorm(x, mean=mean(S_log_ret),sd=sd(S_log_ret)), from = -0.3, to = 0.
    2, add=TRUE, col="green")
legend("topleft", legend=c("Gaussian right-tail"),col=c('green'), lty=1:2,
```

```
    cex=0.8)


####Test QQ plot Normal####
library(car)
qqPlot(coredata(S_log_ret), distribution = "norm", ylab = "JP Returns",
    envelope = FALSE)

####t distribution####
library(fGarch)
fit_T_dist <-fitdistr( S_log_ret, 't')
params.T = fit_T_dist$estimate
mu_T = as.numeric(params.T[1])
sigma_T = as.numeric(params.T[2] * sqrt(params.T[3] / (params.T[3] - 2)))
nu.T = params.T[3]
x <- seq(-5 * sd_S_log, 5 * sd_S_log, length = nrow(S_log_ret))
hist(S_log_ret, 80, freq = FALSE)
lines(x, dstd(x, mean = mu_T, sd = sigma_T, nu = nu.T),
      lwd = 2, lty = 2, col = 'blue')
y <- seq(-5 * sd_S_log, 5 * sd_S_log, length = nrow(S_log_ret))
lines(x=y, dnorm(x, mean_S_log, sd_S_log), col = "green", lwd = 2)
legend("topright", legend=c( "Gaussian distribution ","T-distribution"),col
    =c( "green","blue"), lty=1:2, cex=0.8)

qqPlot(coredata(S_log_ret), distribution = "t",df = 3, ylab = "JP Ret",
    envelope = FALSE)

####Parameters Estimation####

#Alfa & sigma hat
Delta <- 1/252
alpha.hat<-mean(X,na.rm=TRUE)/Delta
sigma.hat <- sqrt(var(X,na.rm=TRUE)/Delta)
mu.hat <- alpha.hat + 0.5*sigma.hat^2
sigma.hat
mu.hat
alpha.hat

##MLE function##

library(MASS)
library("stats4")

LogL <- function(mu=1, sigma=1) {
  A =suppressWarnings( dnorm(S_log_ret, mu, sigma))
  -sum(log(A))}

fit1 <- mle(LogL, start = list(mu = 1, sigma = 1),method = "BFGS")
summary(fit1)
confint(fit1)
```

```
mu_mle <-as.numeric(fit1@coef[1])
sigma_mle<- as.numeric(fi1t@coef[2])

###QUASI MLE###
library(yuima)

Delta_t <- 1/252

gBm <- setModel(drift="mu*x", diffusion="sigma*x",solve.var = "S")

mod <- setYuima(model=gBm, data=setData(S, delta=Delta_t))

mle1 <- qmle(mod, start = list(mu = 0.02, sigma = 0.05),
             lower = list(mu=-0.01, sigma=0.0), upper = list(mu=1, sigma=1)
                 ,
             method = "L-BFGS-B")
summary(mle1)

mle2 <- qmle(mod, start = list(mu = 0.001, sigma = 0.02),
             lower = list(mu=0.00002, sigma=0.015), upper = list(mu=.001,
                 sigma=0.028),
             method = "L-BFGS-B")#### Attempt 4
summary(mle2)


mle3 <- qmle(mod, start = list(mu = 0.01, sigma = 0.06),
             lower = list(mu=0, sigma=0), upper = list(mu=0.01, sigma=0.09)
                 ,
             method = "L-BFGS-B")
summary(mle3)



##########################
##European Option Pricing:
##########################
####BS CALL####

EU_call_bs = function(S, K, r, sigma,t=0, T){
  d1 = (log(S/K)+(r+((sigma)^2)/2)*(T-t))/(sigma*sqrt(T-t))
  d2 = d1-(sigma*sqrt(T-t))
  return((S*pnorm(d1))-(K*exp(-r*(T-t))*pnorm(d2)))
}

####BS-Call-PRICE-PLOT####

lb     = 0          # lower limit of x axis
ub     = 170        # upper limit of x axis
```

```
tau <- 1
r = 0.0199

K <- 100
call1   = EU_call_bs(c(lb:ub), K, r, T=tau, sigma = sigma.hat)
payoff <- ifelse(c(lb:ub<100),0,c(lb:ub)-K)
plot(x = c(lb:ub), y = call1, main = "Black-Scholes price", xlab = "S",
    ylab = expression(paste("C(S,", tau,")")), xlim = c(lb, ub), ylim = c(0,
     max(call1)), type = "l", col = "blue",  lwd = 2)
lines(payoff,col='red')
legend("topleft", legend=c("Call Price", "Payoff"),col=c("blue", "red"),
    lty=1:1, cex=0.8)



####BS_PUT####
EU_put_bs <- function(x,t=0,T,r,K,sigma){
  d2 <- (log(x/K) + (r - 0.5* sigma^2)*(T-t))/ (sigma* sqrt(T-t))
  d1 <- d2 + sigma* sqrt(T-t)
  P <- exp(-r * (T-t))*K*pnorm(-d2) - x*pnorm(-d1)
  return(P)
}
####Montecarlo Method CALL & PUT####
f <- function(x) max(0, x - K)
g<- function(x) max(0, K - x)
set.seed(123)

MCprice <- function(x,t=0,T=1,r,sigma,M=1000,type){
  h <- function(m) {
    u <- rnorm(m/2)
    tmp <- c(x * exp((r - 0.5*sigma^2)*(T-t)+ sigma*sqrt(T-t)*u),#dynamics
        of Stock price Brownian Motio
             x * exp((r - 0.5*sigma^2)*(T-t)+ sigma*sqrt(T-t)*(-u)))
    if (type=='call') {
      mean(sapply(tmp, function(xx) f(xx)))
    } else if (type == 'put') {
      mean(sapply(tmp, function(xx) g(xx)))
    }

  }
  p <- h(M)
  p * exp(-r * (T-t))
}
####Parameter BS##########
#Call at December 30, 2019
estimationDate <- as.Date(Sys.Date(), format="%Y-%m-%d")
optionExpiryDate <- as.Date("2019-12-30", format="%Y-%m-%d")

T <- as.numeric((optionExpiryDate - estimationDate)/252) #to compute actual
    T-t
```

```
threeM_rates <- (getSymbols("DGS3MO", src = "FRED", auto.assign = FALSE))#
    get actual r from FRED
r <-as.numeric(threeM_rates[as.Date(end(threeM_rates), format="%Y-%m-%d")])
    /100

S <- as.numeric(Stock[as.Date(end(Stock), format="%Y-%m-%d")])
K <- 100
T<-.4801587
sigma <- sigma.hat
r <- 0.0199
set.seed(123)
mc_call <- MCprice(x = S, t = 0, T = T, r = r,sigma=as.numeric(sigma.hat),
    M = 1000, type ="call" )
mc_call
set.seed(123)
mc_put <- MCprice(x = S, t = 0, T = T, r = r,sigma=as.numeric(sigma.hat), M
     = 100000, type ="put" )
mc_put
P <- EU_put_bs(S,K=K,t=0,T=T,r=r,sigma=as.numeric(sigma))
C <- EU_call_bs(S = S, t = 0, T = T, r = r, K = K, sigma = as.numeric(sigma
    .hat) )
P
C
####Put-Call parity####
C - S + K* exp(-r*(T)) #this proves that put-call parity is kept


####FFT######
FFTcall.price <- function(phi, S0, K, r, T, alpha = 1, N = 2^12, eta = 0.25
    ) {
  m <- r - log(phi(-(0+1i)))
  phi.tilde <- function(u) (phi(u) * exp((0+1i) * u * m))^T
  psi <- function(v) exp(-r * T) * phi.tilde((v - (alpha + 1) *(0+1i)))/(
      alpha^2 + alpha - v^2 + (0+1i) * (2 * alpha + 1) * v)
  lambda <- (2 * pi)/(N * eta)
  b <- 1/2 * N * lambda
  ku <- -b + lambda * (0:(N - 1))
  v <- eta * (0:(N - 1))
  tmp <- exp((0+1i) * b * v) * psi(v) * eta *(3 + (-1)^(1:N) - ((1:N) - 1
      == 0))/3
  ft <- fft(tmp)
  res <- exp(-alpha * ku) * ft/pi
  inter <- spline(ku, Re(res), xout = log(K/S0))
  return(inter$y * S0)
}
phiBS <- function(u) exp((0+1i) * u * (mu - 0.5 * sigma^2) -
                           0.5 * sigma^2 * u^2)
mu=mu.hat
sigma =sigma.hat
```

```
FFTcall.price(phiBS, S0 = S, K = K, r = r, T = T)
#########Quality of approximation####
par(mar=c(4,4,1,1))
K.seq <- seq(100, 130, length=100)
exactP <- NULL
fftP <- NULL
for(K in K.seq){
  exactP  <- c(exactP , fOptions::GBSOption(TypeFlag = "c", S = S, X = K,
      Time = T, r = r, b = r, sigma = sigma.hat)@price )
  fftP  <- c(fftP , FFTcall.price(phiBS, S0=S, K=K, r=r, T=T) )
}
plot(K.seq, exactP-fftP, type="l",xlab="strike price K")
########MC -BS convergence PLOT######

# settings
s.0   = S    # current stock price
k     = 100   # strike price
tau   = T   # time to maturity in years
r     = r  # annual interest rate
sigma = sigma.hat  # volatility
set.seed(123)

# function for simulating stock price by geometric brownian motion
MCPath = function(s.0, tau, n.t, r, sigma, n.mc) {
  dt = tau/n.t
  t = seq(0, tau, by = dt)
  s.t = s.0 * exp((r - sigma^2/2) * t) *
    exp(replicate(n.mc, c(0, cumsum(sigma * sqrt(dt) * rnorm(n.t)))))
  return(s.t)
}

# function for calculating MC estimate and confidence intervall of european
    option price
MCEurop = function(s.0, k, tau, n.t, r, sigma, n.mc) {

  s.t = MCPath(s.0, tau, n.t, r, sigma, n.mc)

  # payoff
  c = matrix(pmax(0, s.t[nrow(s.t), ] - k), 1, ncol(s.t))  # call
  p = matrix(pmax(0, k - s.t[nrow(s.t), ]), 1, ncol(s.t))  # put
  v = exp(-r * tau) * rbind(c, p)                          # discounted

  # MC estimate
  v.mc = rowSums(v)/ncol(s.t)

  # 95% confidence intervall for MC-estimate
  conf95 = 1.96 * 1/sqrt(ncol(s.t)) * sqrt(1/(ncol(s.t) - 1) * rowSums((v -
      v.mc)^2))
```

```
  return(cbind(v.mc, conf95))
}

# function for calculating the Black-Scholes price of European option
BSEurop = function(s0, k, tau, r, sigma) {
  d1 = (log(s0/k) + (r + (sigma^2/2)) * tau)/(sigma * sqrt(tau))
  d2 = d1 - (sigma * sqrt(tau))
  c  = s0 * pnorm(d1) - k * exp(-r * tau) * pnorm(d2)     # call
  p  = -s0 * pnorm(-d1) + k * exp(-r * tau) * pnorm(-d2)  # put
  return(c(c, p))
}

# main

j      = seq(2, 6, 0.5)
n.mc   = 10^j
v.call = matrix(0, length(n.mc), 2)
v.put  = matrix(0, length(n.mc), 2)

# calculate MC prices for several n.mc
for (i in 1:length(n.mc)) {

  mc = MCEurop(s.0, k, tau, n.t = 1, r, sigma, n.mc = n.mc[i])

  # MC estimate
  v.call[i, 1] = mc[1, 1]
  v.put[i, 1]  = mc[2, 1]

  # 95% confidence
  v.call[i, 2] = mc[1, 2]
  v.put[i, 2]  = mc[2, 2]
}

# calculate Black-Scholes prices
bs = BSEurop(s.0, k, tau, r, sigma)

# plotting

# up = value + 95%CI, dw = value - 95%CI
v.call.up = v.call[, 1] + v.call[, 2]
v.call.dw = v.call[, 1] - v.call[, 2]
v.put.up  = v.put[, 1] + v.put[, 2]
v.put.dw  = v.put[, 1] - v.put[, 2]

# y-axis limit for plot
ylimit = c(min(v.call.dw, v.put.dw), max(v.call.up, v.put.up))

par(mar = c(5, 5, 2, 1))

# plot MC values
```

```
plot(n.mc, v.call[, 1],
     log      = "x", type = "p", pch = 16, ylim = ylimit, col = "black",
     main     = "Convergence of MC Simulation",
     ylab     = "Price of European option",
     xlab     = "Number of MC samples",
     cex.lab = 1.2, cex.axis = 1.2, cex.main = 1.2, cex.sub = 1.2)
lines(n.mc, v.put[, 1], type = "p", pch = 16, col = "darkred")

# plot Black-Scholes values
lines(n.mc, matrix(bs[1], length(n.mc), 1)[, 1],
      type = "l", lty = 2, lwd = 2, col = "black")
lines(n.mc, matrix(bs[2], length(n.mc), 1)[, 1],
      type = "l", lty = 2, lwd = 2, col = "darkred")

# plot errorbars
arrows(n.mc, v.call.dw, n.mc, v.call.up,
       code = 3, angle = 90, length = 0.1, lwd = 2, col = "black")
arrows(n.mc, v.put.dw, n.mc, v.put.up,
       code = 3, angle = 90, length = 0.1, lwd = 2, col = "darkred")

# plot legend
legend(2*10^3, max(bs[1], bs[2]) - 0.25 * min(bs[1], bs[2]),
       legend = c("MC-Estimation with 95% CI", "Black-Scholes-Model", "Call
           ", "Put"),
       lwd = 2, lty = c(0, 2, 0, 0), pch = c(16, NA, 15, 15), cex = 1.2,
       col = c("black", "black", "black", "darkred"))
###################
####Greeks####

T=tau
K <- 100
##Delta


#tau = seq(tau_min, tau_max, by = (tau_max - tau_min)/(steps - 1))
#S   = seq(S_max, S_min, by = -(S_max - S_min)/(steps - 1))
Delta_BS <- function(tau, S, K, r, sig){
  d1 <- (log(S/K)+ (r+0.5*sig^2)*(tau))/(sqrt(tau)*sig)
  pnorm(d1)
}

fOptions::GBSCharacteristics(TypeFlag = 'c', S=S,X=K,Time=T,r=r,b=r,sigma=
    sigma.hat)

#Montecarlo Delta
library(foreach)
MCdelta <- function(x = 1, t = 0, T = 1, r = 1, sigma = 1, M = 1000, f){
  h <- function(m){
    u <- rnorm(M/2)
    tmp <- c(x * exp((r - 0.5 * sigma^2) * (T - t) + sigma *sqrt(T - t) * u
```

```
        ),
              x * exp((r - 0.5 * sigma^2) * (T - t) + sigma *sqrt(T - t) *
                  (-u)))
    g <- function(z) f(z) * (log(z/x) - (r - 0.5 * sigma^2) * (T - t))/(x *
         sigma^2 * (T - t))
    mean(sapply(tmp, function(z) g(z)))
  }
  nodes <- getDoParWorkers()
  p <- foreach(m = rep(M/nodes, nodes), .combine = "c") %dopar%
    h(m)
  p <- mean(p)
  p * exp(-r * (T - t))
}
set.seed(123)
MCdelta(x=S,T=T,r=r,sigma=sigma.hat,f=f,M=10000)

#The numerical approximation -  use the centered derivative
h <- 0.01

delta.num <- function(x){ (EU_call_bs(S = S+h, t = 0, T = T, r = r, K = K,
    sigma = as.numeric(sigma.hat))-
                              EU_call_bs(S = S-h, t = 0, T = T, r = r, K = K
                                  , sigma = as.numeric(sigma.hat) ))/( 2*h)}



delta.num(S)

MCdelta_mix <- function(x = 1, t = 0, T = 1, r = 1, sigma = 1, M = 1000, f,
     dx = 0.001){
  h <- function(m) {
    u <- rnorm(M/2)
    tmp1 <- c((x + dx) * exp((r - 0.5 * sigma^2) * (T - t) +sigma * sqrt(T
        - t) * u),
              (x + dx) * exp((r - 0.5 * sigma^2) * (T - t) + sigma * sqrt(T
                  - t) * (-u)))
    tmp2 <- c((x - dx) * exp((r - 0.5 * sigma^2) * (T - t) +sigma * sqrt(T
        - t) * u),
              (x - dx) * exp((r - 0.5 * sigma^2) * (T - t) + sigma * sqrt(T
                  - t)* (-u)))
    mean(sapply(tmp1, function(x) f(x)) - sapply(tmp2, function(x) f(x)))/(
        2 * dx)
  }
  nodes <- getDoParWorkers()
  p <- foreach(m = rep(M/nodes, nodes), .combine = "c") %dopar%
    h(m)
  p <- mean(p)
  p * exp(-r * (T - t))
}
set.seed(123)
```

```
MCdelta_mix(x=S,T=T,r=r,sigma=sigma.hat,f=f)

#PLOT
# parameter settings for plots
S_min   = 50          # lower bound of Asset Price
S_max   = as.numeric(max(Stock))         # upper bound of Asset Price
tau_min = 0.01        # lower bound of Time to Maturity
tau_max = 1           # upper bound of Time to Maturity
K       = 100         # exercise price
r       = 0.02         # riskfree interest rate
sig     = 0.3       # volatility
d       = 0        # dividend rate
steps   = 60          # steps
q=0
tau = seq(tau_min, tau_max, by = (tau_max - tau_min)/(steps - 1))
S   = seq(S_max, S_min, by = -(S_max - S_min)/(steps - 1))
mesh = outer(tau, sort(S), Delta_BS, K = K, r = r, sig = sig)
title = bquote(expression(paste("Strike␣price␣is␣", .(K), ",␣interest␣rate␣
    is␣",
                                .(r), ",␣annual␣volatility␣is␣", .(sig))))



lattice::wireframe(mesh, drape = T, main = expression(paste("Delta␣as␣
    function␣of␣the␣time␣to␣maturity␣", tau, "␣and␣the␣asset␣price␣S")), sub
     = title, scales = list(arrows = FALSE,col = "black", distance = 1, tick
    .number = 8, cex = 0.7, x = list(labels = round(seq(tau_min,  tau_max,
    length = 7), 1)), y = list(labels = round(seq(S_min, S_max, length = 7),
      1))),xlab = list(expression(paste("Time␣to␣Maturity␣␣", tau)), rot = 3
    0, cex = 1.2), ylab = list("Asset␣Price␣S", rot = -40, cex = 1.2), zlab
    = list("Delta",   cex = 1.1))


##Vega

Vega_BS <- function(S,sig,Tau,K,r) {
  d1 <- (log(S/K)+ (r+0.5*sig^2)*(Tau))/(sqrt(Tau)*sig)

  return(sqrt(Tau)*S*dnorm(d1))
}

MCvega<- function(x=1, t=0, T=1, r=1, sigma=1, M=1000, f, ds=0.01){
  u <- rnorm(M)
  h <- x*exp((r-0.5*(sigma+ds)^2)*(T-t)+(sigma+ds)*sqrt(T-t)*u)
  p <- sapply(h, function(x) f(x))
  p <- mean(p)
  p1 <- p*exp(-r*(T-t))
  h <- x*exp((r-0.5*(sigma-ds)^2)*(T-t)+(sigma-ds)*sqrt(T-t)*u)
  p <- sapply(h, function(x) f(x))
  p <- mean(p)
```

```
  p2 <- p*exp(-r*(T-t))
  (p1-p2)/(2*ds)
}
#plot
meshgrid = function(a, b) {
  list(x = outer(b * 0, a, FUN = "+"), y = outer(b, a * 0, FUN = "+"))
}
first = meshgrid(seq(tau_min, tau_max, (tau_max - tau_min)/(steps - 1)),
    seq(tau_min,  tau_max, (tau_max - tau_min)/(steps - 1)))

tau    = first$x
dump   = first$y
second = meshgrid(seq(S_max, S_min, -(S_max - S_min)/(steps - 1)), seq(S_
    max, S_min,  -(S_max - S_min)/(steps - 1)))

dump2 = second$x
S     = second$y
d1   = (log(S/K) + (r  + sig^2/2) * tau)/(sig * sqrt(tau))
Vega = S  * dnorm(d1) * sqrt(tau)
title = bquote(expression(paste("Strike price is ", .(K), ", interest rate
    is ", .(r), ", dividend rate is ", .(d), ", annual volatility is ", .(
    sig))))
lattice::wireframe(Vega ~ tau * S, drape = T, ticktype = "detailed", main =
     expression(paste("Vega as function of the time to maturity ",  tau, " 
    and the asset price S")), sub = title, scales = list(arrows = FALSE,col
    = "black", distance = 1, tick.number = 8, cex = 0.7, x = list(labels =
    round(seq(tau_min, tau_max, length = 11), 1)), y = list(labels = round(
    seq(S_min, S_max, length = 11), 1))), xlab = list(expression(paste("Time
    to Maturity  ", tau)), rot = 30,  cex = 1.2), ylab = list("Asset Price 
    S", rot = -40, cex = 1.2), zlab = list("Vega",  cex = 1.1))



##Gamma
Gamma_BS <- function(tau, S, K, r, sig) {
  d1 <- (log(S/K)+ (r+0.5*sig^2)*(tau))/(sqrt(tau)*sig)
  return(dnorm(d1)/(sig*S*sqrt(tau)))
}

tau = seq(tau_min, tau_max, by = (tau_max - tau_min)/(steps - 1))
S   = seq(S_max, S_min, by = -(S_max - S_min)/(steps - 1))

MCgamma <- function(x=1, t=0, T=1, r=1, sigma=1, M=1000, f, dS=0.01){
  u <- rnorm(M)
  z <- (x+dS)*exp((r-0.5*sigma^2)*(T-t)+sigma*sqrt(T-t)*u)
  g <- function(t,x,z)f(z) * (log(z/x)-(r-0.5*sigma^2)*(T-t))/(x*sigma^2*(T
      -t))
  p <- sapply(z, function(z) g(t,(x+dS),z) )
  p <- mean(p)
  p1 <- p*exp(-r*(T-t))
```

```
  z <- (x-dS)*exp((r-0.5*sigma^2)*(T-t)+sigma*sqrt(T-t)*u)
  g <- function(t,x,z)
    f(z) * (log(z/x)-(r-0.5*sigma^2)*(T-t))/(x*sigma^2*(T-t))
  p <- sapply(z, function(z) g(t,(x-dS),z) )
  p <- mean(p)
  p2 <- p*exp(-r*(T-t))
  (p1-p2)/(2*dS)
}

gamma = function(tau, S, K, r, sig) {
  d1 = (log(S/K) + (r + sig^2/2) * tau)/(sig * sqrt(tau))
  return(dnorm(d1)/(S * (sig * sqrt(tau))))
}

mesh = outer(tau, sort(S), gamma, K = K, r = r, sig = sig)

lattice::wireframe(mesh, drape = T, main = expression(paste("Gamma␣as␣
    function␣of␣the␣time␣to␣maturity␣", tau, "␣and␣the␣asset␣price␣S")),
    aspect = c(1, 0.75), sub = "Strike␣price␣is␣100␣and␣annual␣volatility␣is
    ␣0.30",  scales = list(arrows = FALSE, col = "black", distance = 1, tick
    .number = 8, cex = 0.7, x = list(labels = round(seq(tau_min, tau_max,
    length = 7), 1)), y = list(labels = round(seq(S_min, S_max, length = 7),
     1))), xlab = list(expression(paste("Time␣to␣Maturity␣␣", tau)), rot = 3
    0, cex = 1.2), ylab = list("Asset␣Price␣S", rot = -30, cex = 1.2),  zlab
     = list("Gamma", rot = 90, cex = 1.2), screen = list(z = 45, x = -70))

#Theta
Theta_BS <- function(S,K,r,sig,tau,Type){
  d1 <- (log(S/K)+ (r+0.5*sig^2)*(tau))/(sqrt(tau)*sig)
  d2 <- d1-(sig*sqrt(tau))
  if (Type=="call") {
    return((-r*K*exp(-r*(tau))*pnorm(d2)) - ((sig*S)/(2*sqrt(tau)))*dnorm(d
        1))
  } else if (Type =="put") {

    return((r*K*exp(-r*(tau))*pnorm(-d2)) - ((sigma*S)/(2*sqrt(tau)))*dnorm
        (d1))
  }

}

MCtheta <- function(x=1, t=0, T=1, r=1, sigma=1, M=1000, f,dt=0.01){
  u <- rnorm(M)
  h <- x*exp((r-0.5*sigma^2)*(T-(t+dt))+sigma*sqrt(T-(t+dt))*u)
  p <- sapply(h, function(x) f(x))
  p <- mean(p)
  p1 <- p*exp(-r*(T-(t+dt)))
  h <- x*exp((r-0.5*sigma^2)*(T-(t-dt))+sigma*sqrt(T-(t-dt))*u)
  p <- sapply(h, function(x) f(x))
  p <- mean(p)
```

```
  p2 <- p*exp(-r*(T-(t-dt)))
  (p1-p2)/(2*dt)
}

meshgrid = function(a, b) {
  list(x = outer(b * 0, a, FUN = "+"), y = outer(b, a * 0, FUN = "+"))
}
first = meshgrid(seq(tau_min, tau_max, (tau_max - tau_min)/(steps - 1)),
    seq(tau_min, tau_max, (tau_max - tau_min)/(steps - 1)))

tau    = first$x
dump   = first$y
second = meshgrid(seq(S_max, S_min, -(S_max - S_min)/(steps - 1)), seq(S_
    max, S_min,  -(S_max - S_min)/(steps - 1)))

dump2 = second$x
S     = second$y


d1    = (log(S/K) + (r + sig^2/2) * tau)/(sig * sqrt(tau))
d2 = d1-(sig*sqrt(tau))

theta = (-r*K*exp(-r*(tau))*pnorm(d2)) - ((sig*S)/(2*sqrt(tau)))*dnorm(d1)
# Plot
title = bquote(expression(paste("Strike␣price␣is␣", .(K), ",␣interest␣rate␣
    is␣",  .(r),  ",␣annual␣volatility␣is␣", .(sig))))
lattice::wireframe(theta ~ tau * S, drape = T, ticktype = "detailed", main
    = expression(paste("Theta␣as␣function␣of␣the␣time␣to␣maturity␣",  tau, "
    ␣and␣the␣asset␣price␣S")), sub = title, scales = list(arrows = FALSE,
    col = "black", distance = 1, tick.number = 8, cex = 0.7, x = list(labels
     = round(seq(tau_min, tau_max, length = 11), 1)), y = list(labels =
    round(seq(S_min, S_max, length = 11),  1))), xlab = list(expression(
    paste("Time␣to␣Maturity␣␣", tau)), rot = 30,  cex = 1.2), ylab = list("
    Asset␣Price␣S", rot = -40, cex = 1.2), zlab = list("Theta", cex = 1.1))




#Rho
Rho_BS <- function(x,Type,sigma,T,t=0){
  d1 <- (log(x/K)+ (r+0.5*sigma^2)*(T-t))/(sqrt(T-t)*sigma)
  d2 <- d1-(sigma*sqrt(T-t))
  if (Type=="call") {
    return((K*(T-t)*exp(-r*(T-t))*pnorm(d2)))
  } else if (Type =="put") {
    return((-K*(T-t)*exp(-r*(T-t))*pnorm(-d2)))
  }

}
```

```
rho =(K*(tau)*exp(-r*(tau))*pnorm(d2))
lattice::wireframe(rho ~ tau * S, drape = T, ticktype = "detailed", main =
    expression(paste("Rho␣as␣function␣of␣the␣time␣to␣maturity␣", tau, "␣and
    ␣the␣asset␣price␣S")), sub = title, scales = list(arrows = FALSE, col =
    "black", distance = 1, tick.number = 8, cex = 0.7, x = list(labels =
    round(seq(tau_min, tau_max, length = 11), 1)), y = list(labels = round(
    seq(S_min, S_max, length = 11), 1))), xlab = list(expression(paste("Time
    ␣to␣Maturity␣␣", tau)), rot = 30, cex = 1.2), ylab = list("Asset␣Price␣
    S", rot = -40, cex = 1.2), zlab = list("Rho", cex = 1.1))
```

```
#Volga
Volga_BS <- function(x,q=0,sigma,t=0,T){
  d1 <- (log(x/K)+ (r+0.5*sigma^2)*(T-t))/(sqrt(T-t)*sigma)
  d2 <- d1-(sigma*sqrt(T-t))

  return(exp(-q*t)*sqrt(T-t)*dnorm(d1)*((d1*d2)/(sigma)))
}
```

```
q       = 0           # dividend rate
steps   = 100         # steps

meshgrid = function(a, b) {
  list(x = outer(b * 0, a, FUN = "+"), y = outer(b, a * 0, FUN = "+"))
}

first = meshgrid(seq(tau_min, tau_max, -(tau_min - tau_max)/(steps - 1)),
    seq(tau_min, tau_max, -(tau_min - tau_max)/(steps - 1)))

tau  = first$x
dump = first$y

second = meshgrid(seq(S_min, S_max, -(S_min - S_max)/(steps - 1)), seq(S_
    min, S_max, -(S_min - S_max)/(steps - 1)))

dump2 = second$x
S     = second$y

d1    = (log(S/K) + (r + sig^2/2) * tau)/(sig * sqrt(tau))
volga = (S * sqrt(tau) * exp(-q * tau) * dnorm(d1) * d1 * (d1 - sig * sqrt(
    tau)))

# Plot
title = bquote(expression(paste("Strike␣price␣is␣", .(K), ",␣interest␣rate␣
```

```
    is␣", .(r), ",␣annual␣volatility␣is␣", .(sig))))
lattice::wireframe(volga ~ tau * S, drape = T, ticktype = "detailed", main
    = expression(paste("Volga␣as␣function␣of␣the␣time␣to␣maturity␣", tau, "␣
    and␣the␣asset␣price␣S")), sub = title, scales = list(arrows = FALSE,
    col = "black", distance = 1, tick.number = 8, cex = 0.7, x = list(labels
     = round(seq(tau_min,  tau_max, length = 11), 1)), y = list(labels =
    round(seq(S_min, S_max, length = 11),  1))), xlab = list(expression(
    paste("Time␣to␣Maturity␣␣", tau)), rot = 30,  cex = 1.2), ylab = list("
    Asset␣Price␣S", rot = -40, cex = 1.2), zlab = list("Volga",  cex = 1.1))

##Vanna
Vanna<- function(x,q=0,t=0,T,sigma){
  d1 <- (log(x/K)+ (r+0.5*sigma^2)*(T-t))/(sqrt(T-t)*sigma)
  d2 <- d1-(sigma*sqrt(T-t))

  return(exp(-q*t)*sqrt(T-t)*dnorm(d1)*((d2)/(sigma)))
}

exp(-q*t)*sqrt(T-t)*dnorm(d1)*((d1*d2)/(sigma))

q       = 0           # dividend rate
steps   = 100         # steps
b=r-q
meshgrid = function(a, b) {
  list(x = outer(b * 0, a, FUN = "+"), y = outer(b, a * 0, FUN = "+"))
}

first = meshgrid(seq(tau_min, tau_max, -(tau_min - tau_max)/(steps - 1)),
    seq(tau_min, tau_max, -(tau_min - tau_max)/(steps -  1)))

tau  = first$x
dump = first$y

second = meshgrid(seq(S_min, S_max, -(S_min - S_max)/(steps - 1)), seq(S_
    min, S_max, -(S_min - S_max)/(steps - 1)))

dump2 = second$x
S     = second$y

d1    = (log(S/K) + (r - q - sig^2/2) * tau)/(sig * sqrt(tau))
d2    = d1 - sig * sqrt(tau)
Vanna = -(exp((b - r) * tau) * d2)/sig * dnorm(d1)

# Plot
title = bquote(expression(paste("Strike␣price␣is␣", .(K), ",␣interest␣rate␣
    is␣", .(r), ",␣dividend␣rate␣is␣", .(q), ",␣annual␣volatility␣is␣", .(
    sig))))
lattice::wireframe(Vanna ~ tau * S, drape = T, ticktype = "detailed", main
    = expression(paste("Vanna␣as␣function␣of␣the␣time␣to␣maturity␣", tau, "␣
    and␣the␣asset␣price␣S")), sub = title, scales = list(arrows = FALSE,
```

```
    col = "black", distance = 1, tick.number = 8, cex = 0.7, x = list(labels
     = round(seq(tau_min, tau_max, length = 11), 1)), y = list(labels =
    round(seq(S_min, S_max, length = 11),  1))), xlab = list(expression(
    paste("Time to Maturity  ", tau)), rot = 30,  cex = 1.2), ylab = list("
    Asset Price S", rot = -40, cex = 1.2), zlab = list("Vanna",  cex = 1.1))
```

```
####GARCH####

library(moments)
acf(S_log_ret,type = 'correlation',main='Log Returns')
acf(S_log_ret^2,type = 'correlation',main='Squared Log Returns')
acf(abs(S_log_ret),type = 'correlation',main='|Log Returns|')



Box.test(abs(S_log_ret), lag = 10, type = c("Ljung-Box"))
Box.test(S_log_ret^2, lag = 10, type = c("Ljung-Box"))

library(rugarch)
ret.jpm <- dailyReturn(Cl(JPM), type='log')

garch11.spec = ugarchspec(variance.model = list(model="sGARCH",garchOrder=c
    (1,1)), mean.model = list(armaOrder=c(0,0)))
jpm.garch11.fit = ugarchfit(spec=garch11.spec, data=ret.jpm)
show(jpm.garch11.fit )
#plot(jpm.garch11.fit)
jpm.garch11.fit


egarch11.spec = ugarchspec(variance.model = list(model="eGARCH",garchOrder=
    c(1,1)), mean.model = list(armaOrder=c(0,0)))
jpm.egarch11.fit = ugarchfit(spec=egarch11.spec, data=ret.jpm)
ni.egarch11 <- newsimpact(jpm.egarch11.fit)
jpm.egarch11.fit
plot(ni.egarch11$zx, ni.egarch11$zy, type="l", lwd=2, col="blue",
     main="EGARCH(1,1) - News Impact",
     ylab=ni.egarch11$yexpr, xlab=ni.egarch11$xexpr)


tgarch11.spec = ugarchspec(variance.model = list(model="fGARCH",
                                                 submodel="TGARCH",
                                                     garchOrder=c(1,1)),
                           mean.model = list(armaOrder=c(0,0)))
jpm.tgarch11.fit = ugarchfit(spec=tgarch11.spec, data=ret.jpm)
ni.tgarch11 <- newsimpact(jpm.tgarch11.fit)
```

```
jpm.tgarch11.fit
plot(ni.tgarch11$zx, ni.tgarch11$zy, type="l", lwd=2, col="blue",
     main="TGARCH(1,1) - News Impact",
     ylab=ni.tgarch11$yexpr, xlab=ni.tgarch11$xexpr)

#plot(jpm.tgarch11.fit)

jpm.garch11.fit = ugarchfit(spec=garch11.spec, data=ret.jpm, out.sample=20)
jpm.garch11.fcst = ugarchforecast(jpm.garch11.fit, n.ahead=10,n.roll=10)
#plot(jpm.garch11.fcst)

####Levy#####
##Retunrs & ECDF
 stock.rtn=na.remove(diff(log(JPM$JPM.Close)))
 returns <- as.vector(stock.rtn)
 m=mean(returns,na.rm=TRUE)
 s=sd(returns,na.rm=TRUE)
 times=index(stock.rtn)
 n = sum(is.na(returns))+sum(!is.na(returns))
 x=seq(1,n)
 y=rnorm(n, m, s)
 plot(times,returns,pch=19,xaxs="i",cex=0.03,col="blue", ylab="X", xlab="n"
     , main = '')
 segments(x0 = times, x1 = times, y0 = 0, y1 = returns,col="blue")
 points(times,y,pch=19,cex=0.3,col="red", ylab="X", xlab="n", main = '')
 abline(h = 3*s, col="black", lwd =1)
 abline(h = -3*s, col="black", lwd =1)

 stock.ecdf=ecdf(as.vector(stock.rtn))
 x <- seq(-0.25, 0.25, length=100)
 px <- pnorm((x-m)/s)
 plot(stock.ecdf, xlab = 'Sample Quantiles', col="blue",ylab = '', main = '
     Empirical Cumulative
 Distribution')
 lines(x, px, type="l", lty=2, col="red",xlab="x value",ylab="Density",
     main="Gaussian cdf")
 legend("topleft", legend=c("Empirical cdf", "Gaussian cdf"),col=c("blue",
     "red"), lty=1:2, cex=0.8)

##FITTING LEVY
par(mfrow=c(2, 2))
par(mar=c(3, 3, 3, 1))
grid <- NULL
library(fBasics)
nFit(X)
nigFit(X, trace=FALSE)
hypFit(X, trace=FALSE)
ghFit(X, trace=FALSE)

#FFT method for VG
```

```
FFTcall.price <- function(phi, S0, K, r, T, alpha = 1, N = 2^12, eta = 0.25
    ) {
   m <- r - log(phi(-(0+1i)))
   phi.tilde <- function(u) (phi(u) * exp((0+1i) * u * m))^T
   psi <- function(v) exp(-r * T) * phi.tilde((v - (alpha +
                                                      1) * (0+1i)))/(alpha^
                                                          2 + alpha - v^2 +
                                                          (0+1i) * (2 *
   lambda <- (2 * pi)/(N * eta)
   b <- 1/2 * N * lambda
   ku <- -b + lambda * (0:(N - 1))
   v <- eta * (0:(N - 1))
   tmp <- exp((0+1i) * b * v) * psi(v) * eta * (3 + (-1)^(1:N) -
                                                    ((1:N) - 1 == 0))/3
   ft <- fft(tmp)
   res <- exp(-alpha * ku) * ft/pi
   inter <- spline(ku, Re(res), xout = log(K/S0))
   return(inter$y * S0)
 }


#Variance-Gamma method
#Parameters fitting
library(VarianceGamma)

vg_param <- vgFit(S_log_ret)$param #esitmate VG parameters on the sample


c <- as.numeric(vg_param[1])
sigma <- as.numeric(vg_param[2])
theta <- as.numeric(vg_param[3])
nu <- as.numeric(vg_param[4])

phiVG <- function(u) {
  omega <- (1/nu) * (log(1 - theta * nu - sigma^2 * nu/2))
  tmp<-1-(0+1i)*theta*nu*u+0.5*sigma^2*u^2*nu
  tmp <- tmp^(-1/nu)
```

```
  exp((0+1i)*u*log(S)+u*(r+omega)*(0+1i))*tmp }
#setting parameters
S <- as.numeric(Stock[as.Date(end(Stock), format="%Y-%m-%d")])
K <- 100
T<-.4801587

r <- 0.0199

price_nel_pa <-FFTcall.price(phiVG, S0 = S, K = K, r = r, T = T)
price_nel_pa
#MC VG
MCpriceVG <- function(x,t=0,T=1,r,sigma,M=1000,type){
  h <- function(m) {
    u <- rnorm(m/2)
    t <- rgamma(n, shape = T/nu, scale = nu)
    N <- rnorm(n, 0, 1)
    X <- theta * t + N * sigma * sqrt(t)
    omega <- (1/nu) * (log(1 - theta * nu - sigma^2 * nu/2))
    tmp <- c(x * exp(r * T + omega * T + X),#dynamics of Stock price
        Brownian Motio
             x * exp(r * T + omega * T + X))
    if (type=='call') {
      mean(sapply(tmp, function(xx) f(xx)))
    } else if (type == 'put') {
      mean(sapply(tmp, function(xx) g(xx)))
    }

  }
  p <- h(M)
  p * exp(-r * (T-t))
}
f <- function(x) max(0, x - K)
g<- function(x) max(0, K - x)
set.seed(123)
MC_VG_price <- MCpriceVG(x=S,T=T,sigma=sigma,type='call',r=r,M=n)
MC_VG_price


set.seed(123)
#Quasi-Newton BFGS method
vg.QN <-vgFit(S_log_ret, method = "BFGS", hessian = TRUE)


vg_parm_QN <- vgFit(S_log_ret, method = "BFGS", hessian = TRUE)$param
c <- as.numeric(vg_parm_QN[1])
sigma <- as.numeric(vg_parm_QN[2])
theta <- as.numeric(vg_parm_QN[3])
nu <- as.numeric(vg_parm_QN[4])

price_QN_par <-FFTcall.price(phiVG, S0 = S, K = K, r = r, T = T)
```

```
price_QN_par
################## American options: Explicit finite-difference method####


AmericanPutExp <- function(Smin=0, Smax,  T=1, N=10, M=10, K, r=0.05, sigma
    =0.01){
  Dt = T/N
  DS = (Smax-Smin)/M
  t <- seq(0, T, by =Dt)
  S <- seq(Smin, Smax, by=DS)
  A <- function(j) (-0.5*r*j*Dt + 0.5*sigma^2*j^2*Dt)/(1+r*Dt)
  B <- function(j) (1-sigma^2*j^2*Dt)/(1+r*Dt)
  C <- function(j) (0.5*r*j*Dt + 0.5*sigma^2*j^2*Dt)/(1+r*Dt)
  P <- matrix(, M+1, N+1)
  colnames(P) <- round(t,2)
  rownames(P) <- round(rev(S),2)
  P[M+1, ] <- K   # C(,j=0) = K
  P[1,] <- 0   # C(,j=M) = 0
  P[,N+1] <- sapply(rev(S), function(x) max(K-x,0))
  optTime <- matrix(FALSE, M+1, N+1)
  optTime[M+1,] <- TRUE
  optTime[which(P[,N+1]>0),N+1] <- TRUE

  for(i in (N-1):0){
    for(j in 1:(M-1)){
      J <- M+1-j
      I <- i+1
      P[J,I] <- A(j)*P[J+1,I+1] + B(j)*P[J,I+1] + C(j)*P[J-1,I+1]
      if(P[J,I] < P[J,N+1])
        optTime[J,I] <- TRUE
    }
  }
  colnames(optTime) <- colnames(P)
  rownames(optTime) <- rownames(P)
  ans <- list(P=P, t=t, S=S, optTime=optTime,N=N,M=M)
  class(ans) <- "AmericanPut"
  return(invisible(ans))
}


### plot method for American put

plot.AmericanPut <- function( obj ){
  plot(range(obj$t),range(obj$S),type="n",axes=F,xlab="t", ylab="S")
  axis(1,obj$t,obj$t)
  axis(2,obj$S,obj$S)
  abline(v = obj$t, h = obj$S, col = "darkgray", lty = "dotted")
  for(i in 0:obj$N){
    for(j in 0:obj$M){
      J <- obj$M+1-j
```

```
      I <- i+1
      cl <- "grey";
      if(obj$optTime[J,I])
        cl <- "black"
      text(obj$t[i+1],obj$S[j+1], round(obj$P[J,I],2),cex=0.75, col=cl)
    }
  }
  DS <- mean(obj$S[1:2])
  y <- as.numeric(apply(obj$optTime,2, function(x) which(x)[1]))
  lines(obj$t, obj$S[obj$M+2-y]+DS, lty=2, col='red')
}



put <- AmericanPutExp(Smax = 65, sigma = sigma.hat, K = 60, T=1, Smin = 0,
    r=r,N=10,M=10)
round(put$P,2)

par(mar=c(3,3,1,1))
plot(put)



S0 <- 36
myval <- round(put$P[which(rownames(put$P)==S0),1],2)


### instability

put.bad <- AmericanPutExp(Smax = 65, sigma = sigma.hat, K = 60, M=15)
round(put.bad$P,2)
plot(put.bad)

######################## American options: Implicit finite-difference method
    ###################
AmericanPutImp <- function( Smin=0, Smax,  T=1, N=10, M=10, K, r=0.05,
    sigma=0.01){
  Dt = T/N
  DS = (Smax-Smin)/M
  t <- seq(0, T, by =Dt)
  S <- seq(Smin, Smax, by=DS)

  A <- function(j) 0.5*r*j*Dt - 0.5*sigma^2*j^2*Dt
  B <- function(j) 1+sigma^2*j^2*Dt+r*Dt
  C <- function(j) -0.5*r*j*Dt - 0.5*sigma^2*j^2*Dt

  a <- sapply(0:M, A)
  b <- sapply(0:M, B)
  c <- sapply(0:M, C)
```

```
  P <- matrix(, M+1, N+1)
  colnames(P) <- round(t,2)
  rownames(P) <- round(rev(S),2)

  P[M+1, ] <- K   # C(,j=0) = K
  P[1,] <- 0   # C(,j=M) = 0
  P[,N+1] <- sapply(rev(S), function(x) max(K-x,0))

  AA <- matrix(0, M-1, M-1)
  for(j in 1:(M-1)){
    if(j>1) AA[j,j-1] <- A(j)
    if(j<M) AA[j,j] <- B(j)
    if(j<M-1) AA[j,j+1] <- C(j)
  }

  optTime <- matrix(FALSE, M+1, N+1)
  for(i in (N-1):0){
    I <- i+1
    bb <- P[M:2,I+1]
    bb[1] <- bb[1]-A(1)*P[M+1-0,I+1]
    bb[M-1] <- bb[M-1]-C(M-1)*P[M+1-M,I+1]
    P[M:2,I] <- solve(AA,bb)
    idx <- which(P[,I] < P[,N+1])
    P[idx,I] <- P[idx,N+1]
    optTime[idx, I] <- TRUE
  }
  optTime[M+1,] <- TRUE
  optTime[which(P[,N+1]>0),N+1] <- TRUE
  colnames(optTime) <- colnames(P)
  rownames(optTime) <- rownames(P)
  ans <- list(P=P, t=t, S=S, optTime=optTime,N=N,M=M)
  class(ans) <- "AmericanPut"
  return(invisible(ans))
}



put <- AmericanPutImp(Smax = 65, sigma = sigma.hat, K = 60, T=1, Smin = 0,
    r=r,N=10,M=10)
round(put$P,2)
putputbad <- AmericanPutImp(Smax = 65, sigma = sigma.hat, K = 60, M=15)

par(mar=c(3,3,1,1))
plot(put)
plot(putputbad)



###### Broadie and Glasserman Monte Carlo method#####
```

```
simTree <- function(b,d, S0, sigma, T, r){
  tot <- sum(b^(1:(d-1)))
  S <- numeric(tot+1)
  S[1] <- S0
  dt <- T/d
  for(i in 0:(tot - b^(d-1))){
    for(j in 1:b){
      S[i*b+j +1] <- S[i+1]*exp((r - 0.5*sigma^2)*dt + sigma*sqrt(dt)*rnorm
          (1))
    }
  }
  S
}



upperBG <- function(S, b, d, f){
  tot <- sum(b^(1:(d-1)))
  start <- tot - b^(d-1) +1
  end <- tot +1
  P <- S
  P[start:end] <- f(S[start:end])
  tot1 <- sum(b^(1:(d-2)))
  for(i in tot1:0){
    m <- mean(P[i*b+1:b+1])
    v <- f(S[i+1])
    P[i+1] <- max(v,m)
  }
  P
}

lowerBG <- function(S, b, d, f){
  tot <- sum(b^(1:(d-1)))
  start <- tot - b^(d-1) +1
  end <- tot +1
  p <- S
  p[start:end] <- f(S[start:end])
  tot1 <- sum(b^(1:(d-2)))

  m <- numeric(b)
  for(i in tot1:0){
    v <- f(S[i+1])
    for(j in 1:b){
      m[j] <- mean(p[i*b+(1:b)[-j]+1])
      m[j] <- ifelse( v>m[j], v, p[i*b+(1:b)[j]+1])
    }
    p[i+1] <- mean(m)
  }
  p
}
```

```
#setting parameters
set.seed(123)
b <- 3
d <- 3
K <- 100
f <- function(x) sapply(x, function(x) max(x-K,0))
g <- function(x) sapply(x, function(x) max(K-x,0))
T <- 1
r <- 0.0199
sigma <- sigma.hat
S0 <- 100

low <- 0
upp <- 0
M <- 1e5
for(i in 1:M){
  S <- simTree(b,d, S0, sigma, T, r)
  low <- low + lowerBG(S, b,d,f)[1]
  upp <- upp + upperBG(S, b,d,f)[1]
}
low/M
upp/M


#################Longstaff and Schwartz least squares method#####

LSM <- function(n, d, S0, K, sigma, r, T){
  s0 <- S0/K
  dt <- T/d
  z <- rnorm(n)
  s.t <- s0*exp((r-1/2*sigma^2)*T+sigma*z*(T^0.5))
  s.t[(n+1):(2*n)] <- s0*exp((r-1/2*sigma^2)*T-sigma*z*(T^0.5))
  CC <- pmax(1-s.t, 0)
  payoffeu <- exp(-r*T)*(CC[1:n]+CC[(n+1):(2*n)])/2*K
  euprice <- mean(payoffeu)

  for(k in (d-1):1){
    z <- rnorm(n)
    mean <- (log(s0) + k*log(s.t[1:n]))/(k+1)
    vol <- (k*dt/(k+1))^0.5*z
    s.t.1 <- exp(mean+sigma*vol)
    mean <- (log(s0) + k*log( s.t[(n+1):(2*n)] )) / ( k + 1 )
    s.t.1[(n+1):(2*n)] <- exp(mean-sigma*vol)
    CE <- pmax(1-s.t.1,0)
    idx<-(1:(2*n))[CE>0]
    discountedCC<- CC[idx]*exp(-r*dt)
    basis1 <- exp(-s.t.1[idx]/2)
    basis2 <- basis1*(1-s.t.1[idx])
    basis3 <- basis1*(1-2*s.t.1[idx]+(s.t.1[idx]^2)/2)
```

```
    p <- lm(discountedCC ~ basis1+basis2+basis3)$coefficients
    estimatedCC <- p[1]+p[2]*basis1+p[3]*basis2+p[4]*basis3
    EF <- rep(0, 2*n)
    EF[idx] <- (CE[idx]>estimatedCC)
    CC <- (EF == 0)*CC*exp(-r*dt)+(EF == 1)*CE
    s.t <- s.t.1
  }

  payoff <- exp(-r*dt)*(CC[1:n]+CC[(n+1):(2*n)])/2
  usprice <- mean(payoff*K)
  error <- 1.96*sd(payoff*K)/sqrt(n)
  earlyex <- usprice-euprice
  data.frame(usprice, error, euprice)
}
set.seed(123)
LSM(100, 3, S0, K, sigma, r, T)
set.seed(123)
LSM(1000, 3, S0, K, sigma, r, T)
set.seed(123)
LSM(1e5, 3, S0, K, sigma, r, T)



#####gBM & BASKET####

GBM <- function(N, sigma, mu, S0, Wt = NULL) {
  if (is.null(Wt)) {
    Wt <- cumsum(rnorm(N, 0, 1))
  }
  t <- (1:N)/252
  p1 <- (mu - 0.5*(sigma*sigma)) * t
  p2 <- sigma * Wt
  St = S0 * exp(p1 + p2)
  return(St)
}

CorrelatedGBM <- function(N, S0, mu, sigma, cor.mat) {
  mu <- as.matrix(mu)
  sigma <- as.matrix(sigma)
  GBMs <- matrix(nrow = N, ncol = nrow(mu))
  Wt <- matrix(rnorm(N * nrow(mu), 0, 1), ncol = nrow(mu))
  Wt <- apply(Wt, 2, cumsum)
  chol.mat <- chol(cor.mat) # upper triangular cholesky decomposition
  Wt <- Wt %*% chol.mat   # key trick for creating correlated paths
  for (i in 1:nrow(mu)) {
    GBMs[,i] <- GBM(N, sigma[i], mu[i] , S0[i], Wt[, i])
  }
  return(GBMs)
```

```
}

GetPrices <- function(tickers, startDate, endDate) {
  prices <- get.hist.quote(instrument = tickers[1], start = startDate,end=
      endDate,
                           quote = 'AdjClose')
  # download the rest of the prices
  for (tik in 2:length(tickers)) {
    tmp <- get.hist.quote(instrument = tickers[tik],
                          start = start, end=end, quote = 'AdjClose')
    prices <- merge(prices, tmp)
  }
  return(prices)
}

set.seed(123)
N <- 4 * 252 # 4 years, each with 252 trading days
t <- (1:N)/252
start <- '2015-08-23'
end = '2019-08-23'
tickers <- c('JPM', 'MSFT')
prices <- GetPrices(tickers, start,end)

# get the cc returns and vectors of average returns and volaitiliy
returns.mat <- as.matrix(na.omit(diff(log(prices))))
mean.vec <- as.numeric(colMeans(returns.mat))
sigma.vec <- as.numeric(sqrt(apply(returns.mat, 2, var)))
prices.vec <- as.numeric(prices[nrow(prices)])
cor.mat <-as.matrix(cor(returns.mat))

paths <- CorrelatedGBM(N, prices.vec , mean.vec, sigma.vec, cor.mat)

colors <- c('darkblue', 'darkgreen', 'darkgoldenrod')
plot(t, paths[,1], type = 'l', ylim = c(50, max(paths)), xlab = "Year",
     ylab = "Price", main = "Simulated␣Asset␣Prices", col = colors[1])
for (i in 2:ncol(paths)) {
  lines(t, paths[, i], col = colors[i])
}
legend(x = 0., y = 80, c('JPM', 'MSFT'), lty = c(1,1,1), col = colors, cex
    = 0.9)


BSMulti <- function( N, T=1, S0=1, r=0.01, z,f, M=1000){
  require(tcltk)
  X <- numeric(M)
  Z <- numeric(M)
  pb <- tkProgressBar(min = 1, max = M)
  for(i in 1:M){
    X[i] <- z( CorrelatedGBM(N, prices.vec , mean.vec, sigma.vec, cor.mat))
    setTkProgressBar(pb, i)
```

```
  }
  for(i in 1:M){
    Z[i] <- f(X[i])
  }
  close(pb)
  exp(-r*T)*mean(Z)
}
z <- function(x) sapply(x, function(x) max(x))
f <- function(x) sapply(x, function(x) max(x-K,0))

set.seed(123)
K<-100
r=0.0199
T=1
BSMulti(S0=last(prices),z=z,f=f,N=N,M=1000,r=r)


###############COPULA & BASKET##############
require(copula)
require(quantmod)
getSymbols("JPM", from="2015-01-01", to="2019-08-23")
attr(JPM, "src")
S <- JPM [,"JPM.Close"]
X <- na.omit(diff(log(S)))
A <- as.numeric(X)

getSymbols("MSFT", from="2015-01-01", to="2019-08-23")
attr(MSFT, "src")
Z <- MSFT [,"MSFT.Close"]
W <- na.omit(diff(log(Z)))
B <- as.numeric(W)

dataset = cbind(A,B)
dataset = as.data.frame(dataset)
prices_ <- cbind(S,Z)

#estimate Kendall tau from data, Kendall Tau measure the association of two
    time series.
tau <- cor.test(x=A, y=B, method = 'kendall')$estimate

#Generator as a function of t
t <- seq(0, 2, length.out = 257) # evaluation points

psi. <- cbind(Pi = exp(-t), # Pi generator
              C  = copClayton@psi(t, theta = iTau(claytonCopula(), tau)),
              F  = copFrank@psi  (t, theta = iTau(frankCopula(),   tau)),
              GH = copGumbel@psi (t, theta = iTau(gumbelCopula(),  tau))
)
plot(t, type = "l", lwd = 2,
     xlim = range(t), ylim = range(psi.), col = 1, ylab = "",
```

```
     xlab = quote(psi(t)~"as a function of t"))
for(j in 2:ncol(psi.)) lines(t, psi.[,j], col = j, lwd = 2)
legend("topright", bty = "n", lty = 1, lwd = 2, col = 2:ncol(psi.),
       legend = c( "Clayton", "Frank",
                   "Gumbel"))

#From Tau to Theta manually
gumbel.tau2theta  = function(tau){
  1/(1 - tau)
}

clayton.tau2theta = function(tau){
  2 * tau / (1-tau)
}

normal.tau2theta  = function(tau){
  sin(tau * pi/2)
}


means = colMeans(dataset)
sds   = sapply(dataset, sd)
dataset2 <- ordered(dataset$A)


params.margins = list(list(mean = means[[1]], sd = sds[[1]]),
                      list(mean = means[[2]], sd = sds[[2]]))
# marginal distributions
marg = rep("norm", length(params.margins))

gumbel.cop = gumbelCopula(gumbel.tau2theta(tau))
out.gumbel        = mvdc(gumbel.cop, marg, params.margins)

normal.cop = normalCopula(normal.tau2theta(tau))
out.normal        = mvdc(normal.cop, marg, params.margins)

clayton.cop = claytonCopula(clayton.tau2theta(tau))
out.clayton        = mvdc(clayton.cop, marg, params.margins)


frank.cop =frankCopula(iTau(frankCopula(),tau))
out.frank        = mvdc(frank.cop, marg, params.margins)

# Create perspective plots of densities
persp(frank.cop, dCopula, phi = 20, theta = 20, ticktype = "detailed", ylab
    = "",
      xlab = "Frank Copula", zlab = "", shade = 0.1)
persp(gumbel.cop, dCopula, phi = 20, theta = 20, ticktype = "detailed",
    ylab = "",
      xlab = "Gumbel Copula", zlab = "", shade = 0.1)
```

```
persp(clayton.cop, dCopula, phi = 20, theta = 20, ticktype = "detailed",
    ylab = "",
      xlab = "Clayton␣Copula", zlab = "", shade = 0.1)

### rClayton MO algorithm
    ###################################################################


rClaytonMO <- function(u, theta)
{
  if(!is.matrix(u)) u <- rbind(u)

  dim. <- dim(u)
  n <- dim.[1]
  d <- dim.[2] - 1
  V <- qgamma(u[,d+1], shape=1/theta) # n-vector, frailty component
  E <- qexp(u[,seq_len(d)]) # (n,d)-matrix

  (1 + E/matrix(rep(V, d), ncol=d))^(-1/theta) # (n,d)-matrix
}

######################rGBM#################################
rGeoBM <- function(u, S0, mu, sigma, T)
{
  stopifnot(0 < u, u < 1, length(mu) == (d <- length(S0)), mu >= 0,
            length(sigma) == d, sigma >= 0, T > 0)
  log.diff <- qnorm(u) * matrix(rep(sigma, each=n), ncol=d) # (n,d)-matrix;
       or t(t(qnorm(u))*sigma)
  log.drft <- (mu - sigma^2 / 2) * T # d-vector
  log. <- matrix(rep(log.drft, n), ncol=d, byrow=TRUE) + log.diff # (n,d)-
      matrix
  matrix(rep(S0, n), ncol=d, byrow=TRUE) * exp(log.) # S_t, t in 1,..,T; (n
      ,d)-matrix
}

###############PAYOFF########################################

payoff <- function(K, N, S0, S, type = c("call", "put"),
                   method = c("basket", "worst.of", "best.of"))
{
  stopifnot(K >= 0, N >= 0, S0 >= 0, S >= 0, length(S0) == ncol(S))
  type <- match.arg(type)
  method <- match.arg(method)
  perf <- switch(method,
                 "basket" = {
                   rowMeans(t(t(S)/S0))
                 },
                 "worst.of" = {
                   apply(t(t(S)/S0), 1, min)
                 },
```

```
                  "best.of" = {
                    apply(t(t(S)/S0), 1, max)
                  },
                  stop("Wrong ’method’"))
  N * pmax(0, if(type=="call") perf - K else K - perf)
}


### Define parameters
    ######################################################

n <- 1e5 # Monte Carlo sample size
d <- 2 # dimension

## Stochastic process parameters
sigma <- as.numeric(sds) # volatilities
r <- 0.0199 # continuously compounded short rate
S0 <- as.numeric(last(prices_)) # initial stocks’ levels
K <- 1 # option strike
N <- 1000 # option notional
T <- 1 # time horizon

## Copulas
## Clayton
family.C <- "Clayton"
th.C <- iTau(getAcop(family.C), tau) # corresponding parameter
clayton.cop <- onacopulaL(family.C, nacList=list(th.C, 1:d)) # Clayton
    copula
## t_3
family.t <- "t"
nu <- 3 # degrees of freedom
th.t <- iTau(ellipCopula(family.t, df=nu), tau) # corresponding parameter
t.cop <- ellipCopula(family.t, param=th.t, dim=d, df=nu) # define copula
    object


### Sampling
    ##################################################################
library(qrng)
## Uniform samples for CDM
set.seed(271)
U.CDM  <- matrix(runif(n*d), ncol=d) # pseudo
set.seed(271)
U.CDM. <- ghalton(n, d=d) # quasi

## Uniform samples for MO
set.seed(271)
U.MO  <- matrix(runif(n*(d+1)), ncol=d+1) # pseudo
set.seed(271)
U.MO. <- ghalton(n, d=d+1) # quasi
```

```
## t samples via CDM
U.t.CDM  <- cCopula(U.CDM,  cop=t.cop, inverse=TRUE) # pseudo
U.t.CDM. <- cCopula(U.CDM., cop=t.cop, inverse=TRUE) # quasi

## Clayton samples via CDM
U.C.CDM  <- cCopula(U.CDM,  cop=clayton.cop, inverse=TRUE) # pseudo
U.C.CDM. <- cCopula(U.CDM., cop=clayton.cop, inverse=TRUE) # quasi

## Clayton samples via MO
U.C.MO  <- rClaytonMO(U.MO,  theta=th.C) # pseudo
U.C.MO. <- rClaytonMO(U.MO., theta=th.C) # quasi


## Geometric Brownian Motion samples
S.t.CDM <- rGeoBM(U.t.CDM, S0=S0, mu=rep(r, d), sigma=sigma, T=T)
S.C.CDM <- rGeoBM(U.C.CDM, S0=S0, mu=rep(r, d), sigma=sigma, T=T)
S.C.MO  <- rGeoBM(U.C.MO,  S0=S0, mu=rep(r, d), sigma=sigma, T=T)

## Quasi-geometric Brownian Motion samples
S.t.CDM. <- rGeoBM(U.t.CDM., S0=S0, mu=rep(r, d), sigma=sigma, T=T)
S.C.CDM. <- rGeoBM(U.C.CDM., S0=S0, mu=rep(r, d), sigma=sigma, T=T)
S.C.MO.  <- rGeoBM(U.C.MO.,  S0=S0, mu=rep(r, d), sigma=sigma, T=T)


### 2.3 Functional Calculation
    ##################################################

erT <- exp(-r*T)

## Using pseudo-random samples

## Basket call
basket.t.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM))
basket.C.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM))
basket.C.MO  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO))

##Best-of
best.of.t.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM,method="best.of
    "))
best.of.C.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM,method="best.of
    "))
best.of.C.MO  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO,method="best.of"
    ))

## Worst of call
worst.of.t.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM, method="worst
    .of"))
worst.of.C.CDM <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM, method="worst
    .of"))
```

```
worst.of.C.MO  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO,  method="worst
    .of"))



## Using quasi-random samples

## Basket call
basket.t.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM.))
basket.C.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM.))
basket.C.MO.  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO.))

##Best-of
best.of.t.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM.,method="best.
    of"))
best.of.C.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM.,method="best.
    of"))
best.of.C.MO.  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO.,method="best.
    of"))


## Worst of call
worst.of.t.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.t.CDM., method="
    worst.of"))
worst.of.C.CDM. <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.CDM., method="
    worst.of"))
worst.of.C.MO.  <- erT * mean(payoff(K, N=N, S0=S0, S=S.C.MO.,  method="
    worst.of"))

### 2.4 Results
    ######################################################################


res <- array(, dim=c(3,2,2), dimnames=list(type=c("basket", "worst.of",'
    best.of'),
                                           copula=c("Clayton", paste0("t",
                                              nu)),
                                           method=c("CDM", "MO")))
res["basket",,]      <- matrix(c(basket.C.CDM., NA, basket.C.MO., basket.t.
    CDM.), ncol=2)
res["worst.of",,]    <- matrix(c(worst.of.C.CDM., NA, worst.of.C.MO., worst.
    of.t.CDM.), ncol=2)
res["best.of",,]      <- matrix(c(best.of.C.CDM., NA, best.of.C.MO., best.of
    .t.CDM.), ncol=2)
res


#####ANNs####
library(keras)
library(rsample)
```

```
sample<-sde::GBM(x=100,N=1000000,r = r,sigma = 0.8,T = 1)


#Generate Dataset
mydata <- NULL
mydata$Stock <- sample
mydata$Strike <- sample*runif(length(sample),min = 0.4,max=1)
mydata$Time <- runif(n=length(sample))
mydata$sigma <- runif(n=length(sample), min = 0.1, max = 0.8)
mydata$r <-runif(n=length(sample),min = 0.01, max=0.05)
mydata$BS <-  EU_call_bs(S = mydata$Stock, t = 0, T = mydata$Time, r =
    mydata$r, K = mydata$Strike, sigma = as.numeric(mydata$sigma));

mydata <- as.data.frame(mydata)

#Split dataset
split <- rsample::initial_split(mydata, prop = .7, strata='BS' )

train <- rsample::training(split)
test  <- rsample::testing(split)

# Create & standardize feature sets
# training features
train_x <- train %>% dplyr::select(-BS)
mean    <- colMeans(train_x)
std     <- apply(train_x, 2, sd)
train_x <- scale(train_x, center = mean, scale = std)
# testing features
test_x <- test %>% dplyr::select(-BS)
test_x <- scale(test_x, center = mean, scale = std)

# Create & transform response sets
train_y <- log(train$BS)
test_y  <- log(test$BS)


#MODEL 1
model_1 <- keras_model_sequential() %>%
  layer_dense(units =  200,activation = "relu", input_shape = ncol(train_x)
     ) %>%
  layer_dense(units = 130,activation = "relu") %>%
  layer_dense(units = 50,activation = "relu") %>%
  layer_dense(units = 1)  %>%

  # backpropagation
  compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae")
```

```
  )

learn_1 <- model_1 %>% fit(
  x = train_x,
  y = train_y,
  epochs = 45,
  batch_size = 256,
  validation_split = .2,
  verbose = TRUE,
)


#MODEL 2

model_2 <- keras_model_sequential() %>%
  layer_dense(units =  200,activation = "relu", input_shape = ncol(train_x)
      ,kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_batch_normalization() %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 130,activation = "relu",kernel_regularizer =
      regularizer_l2(0.001)) %>%
  layer_batch_normalization() %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 50,activation = "relu",kernel_regularizer =
      regularizer_l2(0.001)) %>%
  layer_batch_normalization() %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 1)  %>%

  compile(
    optimizer = "rmsprop", #optimizer_adam(lr = 0.01)
    loss = "mse",
    metrics = c("mae")
  )
learn_2 <- model_2 %>% fit(
  x = train_x,
  y = train_y,
  epochs = 45,
  batch_size = 256,
  validation_split = .2,
  verbose = TRUE,
  callbacks = list(
    #callback_early_stopping(patience = 10),
    callback_reduce_lr_on_plateau(patience = 5))
)

#Storing Prediction
risultati <- NULL
risultati$predcted_values_model_1<- model_1 %>% predict(test_x)
risultati$true_value <- test_y
```

```
risultati$predcted_values_model_2<- model_2 %>% predict(test_x)
risultati$pred_value_model_1_converted <- exp(risultati$predcted_values_
    model_1)
risultati$true_converted <- exp(risultati$true_value)
risultati$pred_value_model_2_converted <- exp(risultati$predcted_values_
    model_2)
risultati$S_K <- test[,1] / test[,2]
risultati$Err_model_1 <- abs(risultati$true_converted - risultati$pred_
    value_model_1_converted )
risultati$Err_model_2 <- abs(risultati$true_converted - risultati$pred_
    value_model_2_converted )
risultati <- as.data.frame(risultati)


plot(x=risultati$true_converted, y=risultati$pred_value_model_1_converted ,
     cex= 0.001, xlab='Actual', ylab='Predicted␣Model␣1')
plot(x=risultati$true_converted, y=risultati$pred_value_model_2_converted ,
     cex= 0.001, xlab='Actual', ylab='Predicted␣Model␣2')
plot(x=risultati$S_K, risultati$Err_model_1, xlab='S␣/␣K', ylab='|BS␣Price␣
    -␣Predicted␣model␣1|',cex=0.01)
plot(x=risultati$S_K, risultati$Err_model_2, xlab='S␣/␣K', ylab='|BS␣Price␣
    -␣Predicted␣model␣2|',cex=0.01)

learn_1
learn_2

(result_1<- model_1 %>% evaluate(test_x, test_y))
(result_2<- model_2 %>% evaluate(test_x, test_y))
```