PDE - Final Project

Oxana USHAKOVA

July 25, 2016

Contents

Notations

- S - Stock price, also called Spot price (or any underlying asset)
- $V(S,t)$ - value of an option, depending on time and spot price
- K - Strike price
- r - risk-free rate
- d - dividend yield
- μ - drift rate of S - the rate at which the average of S changes
- σ - volatility of the stock, standard deviation of $log(S)$ - return on stock
- T_0,T - initial and final time
- θ - long variance : as t tends to infinity, the expected value of ν tends to θ
- κ - the rate at which ν reverts to θ
- ξ - the volatility of volatility

Introduction

The main objective of this project is to learn what are stochastic PDE's and how to solve them with FEM. The numerical results are presented in Feel++ and FreeFem++ environments.

The paper can be divided in 4 parts. As the most common example of stochastic PDEs is the financial model Black-Scholes for option pricing, the first and the second chapters give the basics both in financial theory and stochastic calculus. The third and the forth chapters show the implementation of FEM in theory and in practice respectfully.

Chapter 1

Financial Market : basics

Computational finance is an interdisciplinary field which joins financial mathematics, stochastics, numerics and scientific computing. Its task is to estimate as accurately and efficiently as possible the risks that financial instruments generate. In this project we will focus on options.

1.1 Option as a financial instrument

An **Option** is a contract that gives the buyer the right, *but not the obligation*, to buy or sell an underlying asset (a stock, a bond, gold, other option, etc) at a specific price, called **Strike** price, on or before a certain date. An option is a security, just as stockes or bonds, it has its own price called **Premium**.

Every option contract has several parameters to be pre-set at $t = T_0$:

- Who buys (long), who sells (short)?
- What is the underlying asset ?
- What is the maturity T of the contract ?
- Does the contract give the right to buy (**call option**) or to sell(**put option**) ?
- What is the Strike price K ?
- What is the price of the option itself, i.e. premium?

  Obviously, the asymmetry of the option contract leads us to the question if we buy this contract it or we sell it. A buyer of an option contract is said to be in **long** positions, the seller - in **short** position.

  Let's take a look on potential gain diagram for Long Call Option:

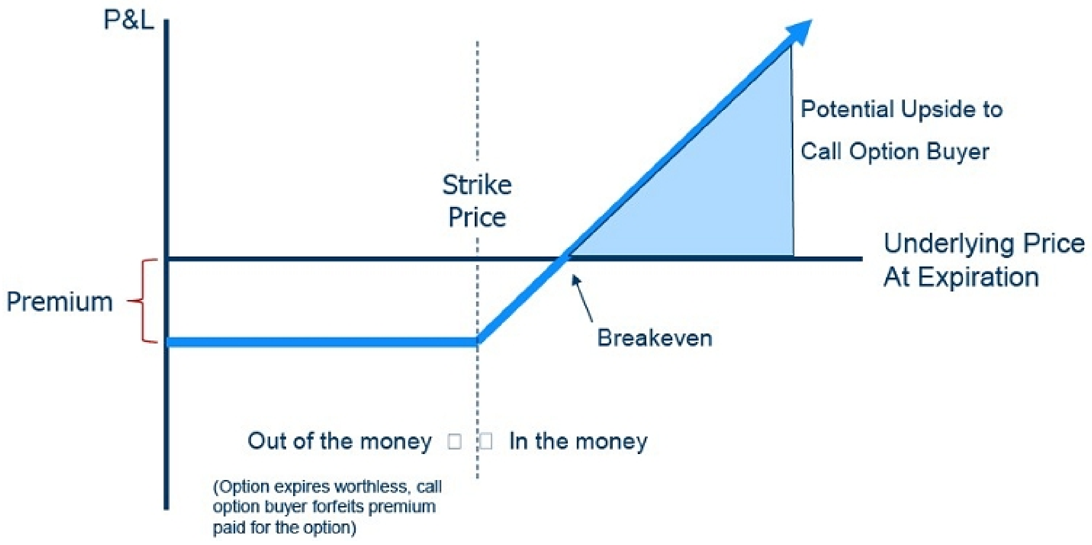


Figure 1.1: Long Call Option

As it comes from the figure 1, Long Call Option brings profit (« in the money ») , if S at maturity T is higher than the Breakeven point. So the payoff of this option is premium ignored):

- $S - K$ if $S(T) > K$
- 0 if $S(T) < K$

Here are all 4 possible payoffs:

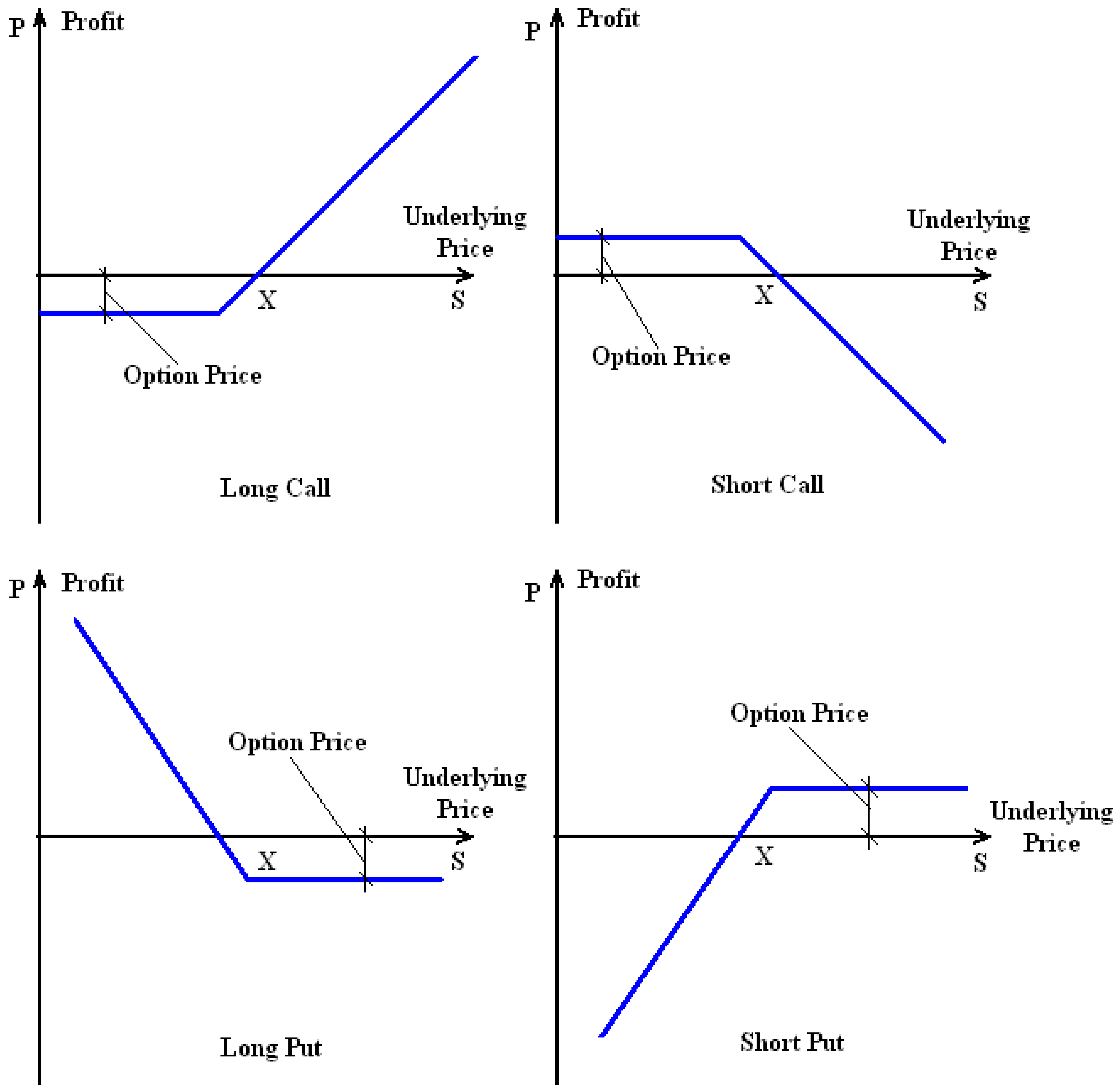


Figure 1.2: Possible payoffs

1.1.1 Put-Call Parity

The put–call parity defines a relationship between the price of a European call option and European put option, both with the identical strike price and maturity.
We consider two portfolios:

- one European call (C) and cash (Ke^{-rT})
- one European put (P) and one share (S_0)

At time T they both worth $max(S_T, K)$, hence their values should be equal today, i.e.:

$$C + Ke^{-rT} = P + S_0 \tag{1.1}$$

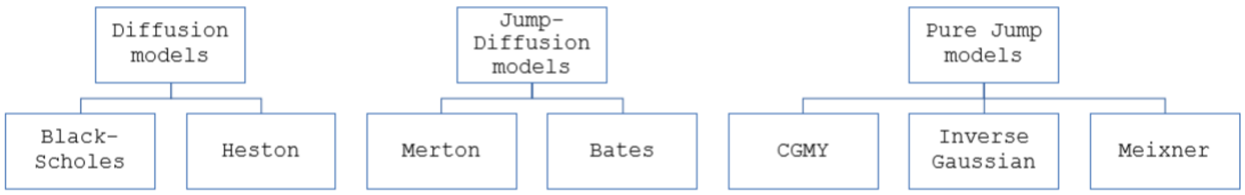
1.1.2 Vanilla vs Exotics

This type of of options is said to be **Vanilla** : the simplest version of all, without any optional extras, by analogy with the default ice cream flavour, vanilla. There exist a rich family of, so called **Exotic** options. Here we list only several of them:

- American option (Bermudian)
  can be executed not only at T,but on any time of life of the option (T_0, T);
  Bermudian option can be executed on a specific period during the life of the option, i.e. every second Monday, June, etc.
- Barrier option (Paris)
  can be activated for be executed only if the asset price touches (or not) a specific barrier;
  Paris barrier option can be activated for be executed only if the asset price satisfy the barrier condition for a certain period of time (i.e. 15min, 1 day, 30%, etc).
- Asian option
  its payoff is determined by the average underlying price over some pre-set period of time.
- Lookback option (Russian)
  its payoff depends not at S at final time T, $S(T)$, but on $max(S)$ over the life of the option.
  Russian lookback option is a special case of lookback : it has no pre-set expiration time, it's up to buyer of the option when to execute it. It's also called 'no regret' option.

1.2 Option Pricing models

1.2.1 Pricing Models



- **Diffusion models** have only a diffusion component, given by Wiener process : *Black-Scholes, Heston*;

  * *Black-Scholes*
    W, and consequently its increment dW,represents the only source of 'diffusion' uncertainty in the price history of the stock :
    $dS = \mu S dt + \sigma S dW$
  * *Heston*
    W, and consequently its increment dW, and ν, and consequently root from it, represent two sources of 'diffusion' uncertainty in the price history of the stock :
    $dS = \mu S dt + \sqrt{\nu} S dW$
    $d\nu = \kappa(\theta - \nu)dt + \xi\sqrt{\nu} dW$

- **Jump diffusion models** have both diffusion component, given by Wiener process, and jump component, given by compounded Poisson process :

  Rq: For financial applications, it is of little interest to have a process with a single possible jump size. The compound Poisson process is a generalization where the waiting times between jumps are exponential but the jump sizes can have an arbitrary distribution.

  ∗ *Merton*
  dW represents the source of 'diffusion' uncertainty and the last term represents is the source of 'jump' uncertainty (compound Poisson process with Gaussian jumps) in the price history of the stock. dW and ν represent the source of 'diffusion' uncertainty and the last term represents is the source of 'jump' uncertainty (compound Poisson process with Gaussian jumps) in the price history of the stock.
  It mixes Black-Scholes model Compounded Poisson process :
  $dS = \mu S dt + \sigma S dW + \sum_{n=1}^{N_t} Y_i$

  ∗ *Bates*
  dW and ν represent the source of 'diffusion' uncertainty and the last term represents is the source of 'jump' uncertainty (compound Poisson process with Gaussian jumps) in the price history of the stock.
  It mixes Black-Scholes model Compounded Poisson process : It mixes Merton and Heston models:

  $dS = \mu S dt + \sqrt{\nu} S dW + \sum_{n=1}^{N_t} Y_i$

  $d\nu = \kappa(\theta - \nu)dt + \xi\sqrt{\nu}dW$

- **Pure jump models** have no diffusion, just a random process :

  ∗ *CGMY*
  ∗ *NIG*

1.3 Greeks

Greeks are the quantities representing the sensitivity of the price of options to a change in underlying parameters. We list some of them, further in the project we will study them:

– **Delta** - the rate of change of the option price w.r.t. the price of the underlying asset:

$$\delta = \frac{\partial C}{\partial S} \tag{1.2}$$

– **Theta** - the rate of change of the portfolio price w.r.t. the time:

$$\theta = \frac{\partial \Pi}{\partial t} \tag{1.3}$$

– **Gamma** - the rate of change of the portfolio w.r.t. the price of the underlying asset:

$$\Gamma = \frac{\partial^2 \Pi}{\partial S^2} \tag{1.4}$$

– **Vega** - the rate of change of the portfolio price w.r.t. the volatility of the underlying asset:

$$\upsilon = \frac{\partial \Pi}{\partial \sigma} \tag{1.5}$$

– **Rho** - the rate of change of the portfolio price w.r.t. the interest rate:

$$\upsilon = \frac{\partial \Pi}{\partial r} \tag{1.6}$$

Chapter 2

Stochastics : basics

In this section we give concepts that will be used further when the existence and the unicity of a solution for a stochastic PDE will be discussed.

2.1 Stochastic process

A **stochastic process** in discrete time, denoted by $(X_t)_{t \in \mathcal{T}}$, is a sequence of random variables $X_0, X-1, ..X_T$ defined on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ with values in $\mathbb{R}$. For a given $\omega \in \Omega$, the sequence $X_t(w), t \in \mathcal{T}$ is called a **path** of the process (X_t), where $\mathcal{T}$ stands for time series (dates) : $\mathcal{T} = \{0, 1, ..T\}$

2.2 Filtrations

A **filtration** $\mathcal{F}$ on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ is a sequence of σ-algebras $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq ... \subseteq \mathcal{F}_T \subseteq \mathcal{A}$. A quadruple $(\Omega, \mathcal{A}, \mathbb{P}, \mathcal{F})$ is called a **filtered probability space**.
A stochastic process $(X_t)_{t \in \mathcal{T}}$ is sais to be **adapted to a filtration** $\mathcal{F}$ if for all $t \in \mathcal{T}$, the random variable X_t is $\mathcal{F}_t$ - measurable.

2.3 Martingales

Let $(\Omega, \mathcal{A}, \mathbb{P}, \mathcal{F}))$ be a filtered probability space. A stochastic process $X = (X_t)_{t \in \mathcal{T}}$ is called a **martingale** if

1. X is adapted to $\mathcal{F}$
2. $\mathbb{E}[|X_t|] < \inf \forall \in \mathcal{T}$
3. $\mathbb{E}[X_t | \mathcal{F}_s] = X_s \forall t \geq s$

Note, the following brownian martingale is used in the Black-Scholes theory:

$$(e^{\alpha B_t - \frac{\alpha^2}{2} t}) \tag{2.1}$$

and

$$\mathbb{E}[e^{\alpha B_t - \frac{\alpha^2}{2} t} | \mathcal{F}_f = e^{\alpha B_s - \frac{\alpha^2}{2} s}, 0 \leq s < t \tag{2.2}$$

2.4 Itô Calculus

The **Itô integral** is the central concept of Itô calculus. The integrand and the integrator are stochastic processes:

$$Y_t = \int_0^t X_s dB_s, \tag{2.3}$$

where X is a locally square integrable process adapted to the filtration generated by B, which is a Wiener process or Brownian motion. The result of the integration is another process.

If we assume that $\{\pi_n\}$ is a sequence of partitions of $[0,t]$ with min mesh size, then the integration by parts for Itô integral is given by :

$\int_0^t XdB = \lim_{n\to\infty} \sum_{t_{i-1},t_i\in\pi_n} X_{t_{i-1}}(B_{t_i} - B_{t_{i-1}})$.

X is square integrable, then its integral w.r.t. B can be defined and X is said to be B-integrable. As $\int_0^t (X - X_n)^2 ds \to 0$ in probability, the Itô integral becomes:

$\int_0^t XdB = \lim_{n\to\infty} \int_0^t X_n dB$

where we faal again on convergence in probability. This leads us to a very important property: **Itô isometry**:

$$\mathbb{E}[(\int_0^t X_s dB_s)^2] = \mathbb{E}[\int_0^t X_s^2 ds] \tag{2.4}$$

2.5 Itô's lemma

The most crucial notion for solving stochastic PDE's is the **Itô's formula (lemma)**.

Given a process X_t described by an SDE, the Itô formula tells us how another process $Y_t = f(t, X_t)$ that is given in terms of t and X_t is itself described by an SDE. The formula is so useful because it can be used to transform an SDE that is hard to solve (integrate) into another SDE that can be solved, and then transforming the solution back into a solution for the original equation.

Let X_t be a process given by the SDE: $dX_t = udt + vdB_t$. Let $f(t,x) \in C^2$ .Taylor expansion for $f(t,x)$ is given by :

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}dx + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}dx^2 ... \tag{2.5}$$

Now, we change $x \longrightarrow X_t$, $dx \longrightarrow \mu dt + \sigma dB_t$ and get :

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}(\mu dt + \sigma dB_t) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(\mu^2 dt^2 + 2\mu\sigma dt dB_t + \sigma^2 dB_t^2) + ... \tag{2.6}$$

Using $dt * dt = dt * dB_t = dB_t * dt = 0, dB_t * dB_t = dt$, we finally obtain we get the solution:

$$df = (\frac{\partial f}{\partial t} + \mu\frac{\partial f}{\partial x} + \frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2})dt + \sigma\frac{\partial f}{\partial x}dB_t \tag{2.7}$$

Or, more generally:
Let $f(t,x) \in C^2$, $Y_t = f(t, X_t)$. Then Y_t follows:

$$dY_t = \frac{\partial f}{\partial t}(t, X_t)dt + \frac{\partial f}{\partial x}(t, X_t)dB_t + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(t, X_t)(dB_t)^2 \tag{2.8}$$

2.5.1 Solving Black-Scholes PDE

Let's solve the Black-Scholes SDE:

$$dS_t = \mu S_t dt + \sigma S_t dB_t, X_0 > 0 \tag{2.9}$$

Let $f(t,x) = lnx$,$f \in C^2$ and $Y_t = lnS_t$. It follows that

$$dY_t = \frac{1}{S_t}dX_t + \frac{1}{2}(-\frac{1}{S_t^2})(dS_t)^2 = \mu dt + \sigma dB_t - \frac{1}{2}\frac{1}{X_t^2}\sigma^2 S_t^2 dt = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dB_t. \tag{2.10}$$

Now lets integrate it:

$Y_t - Y_0 = \int_0^T dY_t = (\mu - \frac{1}{2}\sigma^2) \int_0^T dt + \sigma \int_0^t dB_t = (\mu - \frac{1}{2}\sigma^2)T + \sigma B_t$

And we finally get the solution:

$$S_T = e^{Y_T} = e^{Y_0 + (\mu - \frac{1}{2}\sigma^2)T + \sigma dB_t} = S_0 e^{(\mu - \frac{1}{2}\sigma^2)T + \sigma dB_t} \tag{2.11}$$

Chapter 3

Black-Scholes PDE

3.1 Theoretical assumptions

The Black-Scholes model is one of the most important concepts in modern option pricing theory. Before introducing the underlying idea, we list the model's assumptions.

- The stock price follows the Geometric Brownian motion with μ and σ constant.
- The short selling of securities with full use of proceeds is permitted.(*No time limits or conditions on transaction*).
- No transaction costs or taxes. All securities are perfectly divisible.
- There are no dividends during the life of derivative. No arbitrage.
- Security trading is continuous.
- Risk-free rate is constant and the same of all maturities.

3.2 Wiener process and Geometric Brownian Motion

The Black-Scholes SDE describing the process for a stock price S_t is given by:

$$dS_t = \mu S_t dt + \sigma S_t dB_t \tag{3.1}$$

This process is called **Geometric Brownian Motion** - a continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion with drift.
We have already seen that its solution is given by:

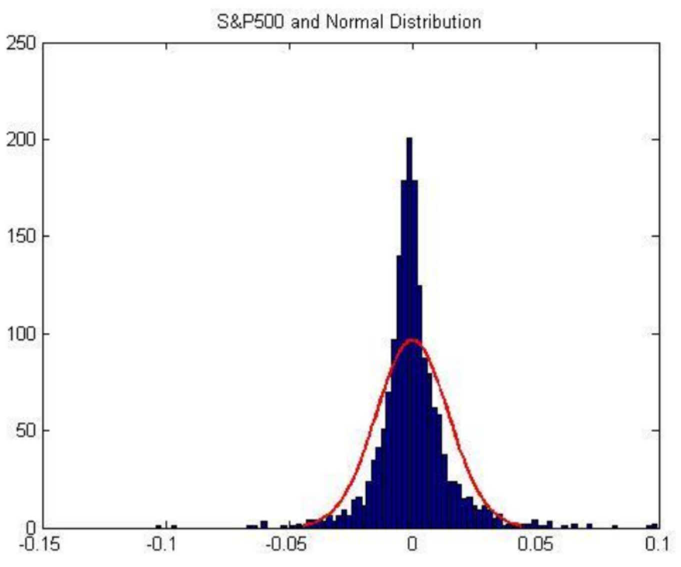
$$S_T = S_0 e^{(\mu - \frac{1}{2}\sigma^2)T + \sigma dB_t} \tag{3.2}$$

where the stock price S_t is log-normally distributed.
So why to use GBM and not the Brownian motion itself?

- The expected returns of GBM are independent of the value of the process (stock price), which agrees with what we would expect in reality.
- A GBM process only assumes positive values, just like real stock prices.
- A GBM process shows the same kind of 'roughness' in its paths as we see in real stock prices.
- Calculations with GBM processes are relatively easy.

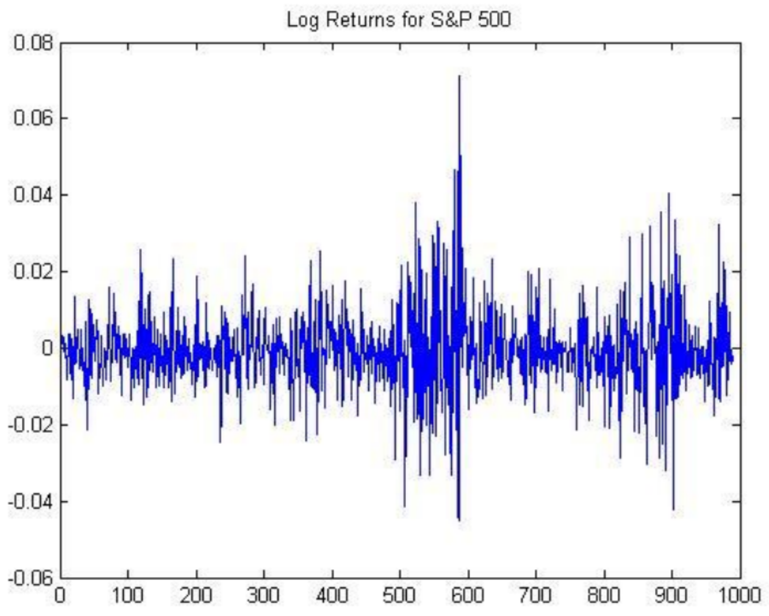
As the (Geometric) Brownian motion is the dynamic form of Normal distribution it is symmetric around its mean, with zero skewness and kurtosis equal to 3. However, empirical market data usually doesn't respect Normal Distribution properties.
Following figure show the imperfection of Black-Scholes model in fitting market data. We observe S&P 500 close prices from 3 Jan, 2005 to 7 Dec, 2011 (1748 ticks). Here the kurtosis is 12.0293 and skewness is 0.5281. The

higher kurtosis causes heavy tails, which means a higher chance of large price changes, i.e. jumps. The negative skewness shows the asymmetry.



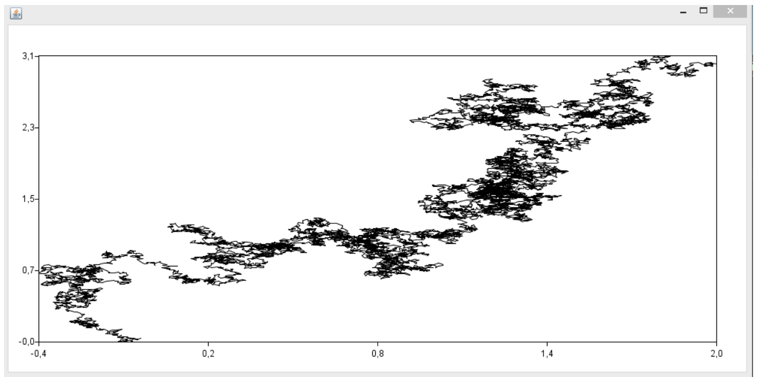
Another imperfection of Black-Scholes model is the volatility clustering: parameters of uncertainty change stochastically over time.

The best way to demonstrate the volatility clustering is plotting autocorrelations of squared log-returns. This method helps see if the heteroscedasticity took place. Heteroscedasticity is a violation of the constant error variance assumption. It occurs if different observations' errors have different variances. Obviously, in case of financial market series we take volatility as the source of error. The following figure shows high autocorrelations, i.e. our data is heteroscedastic, thus we have volatility clustering.



So, GBM has no jumps in it and its volatility is constant, that is why such models like Heston or Merton are much more popular in practice (see section 1.2). Here we give some illustrations:

Figure 3.1: Wiener and Wiener-Poisson process



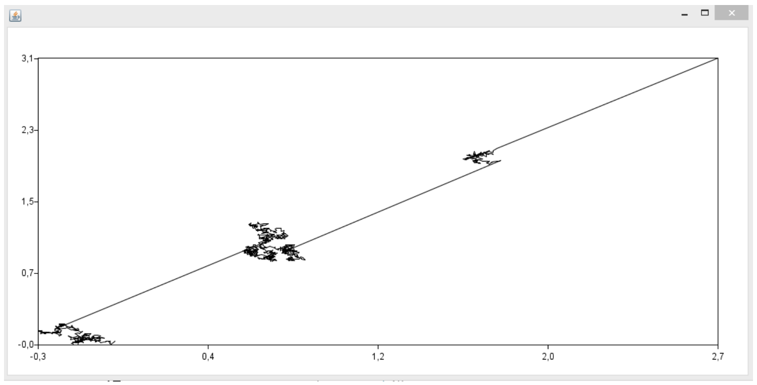
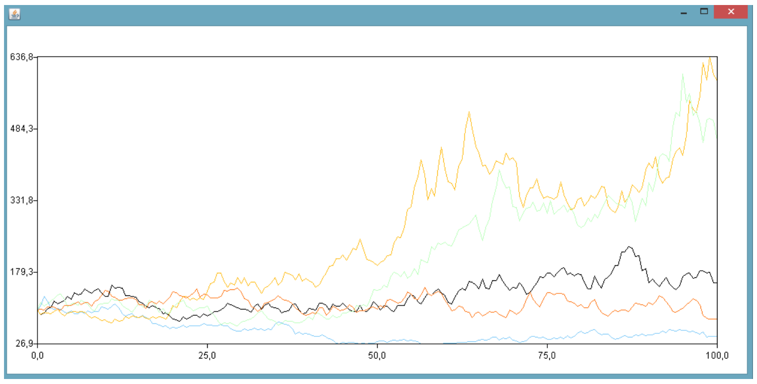
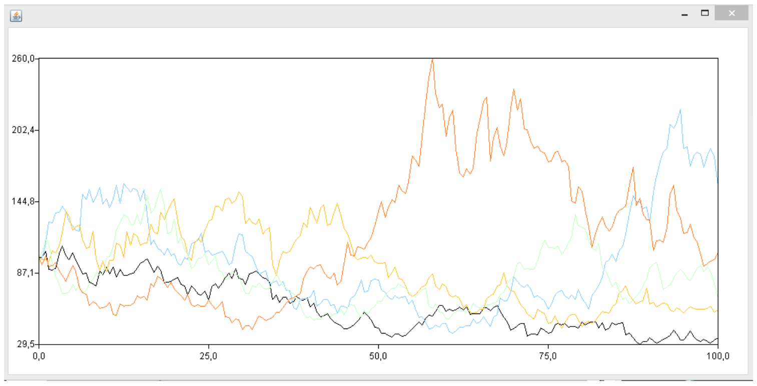


Figure 3.2: Black Scholes and Merton





For further computations we need to find mean and variance.Lets first find $\mathbb{E}[S_t]$ using martingales theory:

$$\mathbb{E}[S_t] = S_0 e^{\mu T}\mathbb{E}[e^{\sigma B_t - \frac{1}{2}\sigma^2 T}] = S_0 e^{T}. \tag{3.3}$$

since $\mathbb{E}[e^{\sigma B_T - \frac{1}{2}\sigma^2 T}] = e^{\sigma B_0} = 1$.
Now, lets find $\mathbb{E}[S_t]$ with moment generating function.We know that for a normally distributed variable $X \sim \mathcal{N}(\mu, \sigma^2)$ the moment generating function is given by $M_X(t) := \mathbb{E}[e^{tX}] = e^{\mu t + \frac{1}{2}\sigma^2 t^2}$.
When we apply this to:

$X = \sigma B_T - \frac{1}{2}\sigma^2 T \sim \mathcal{N}(-\frac{1}{2}\sigma^2 T, \sigma^2 T)$

we get:

- $\mathbb{E}[e^X] = M_X(1) = e^{-\frac{1}{2}\sigma^2 T + \frac{1}{2}\sigma T} = 1$
- $\mathbb{E}[(e^X)^2] = \mathbb{E}[e^{2X}] = M_X(2) = e^{-\sigma^2 T + 2\sigma T}$

For the second moment we get:

$\mathbb{E}[S_t^2] = S_0^2 e^{2\mu T}\mathbb{E}[e^{2X}] = S_0^2 e^{2\mu T} e^{\sigma^2 T}$

And finally the variance:

$\mathbb{V}[S_T] = \mathbb{E}[S_T^2] - (\mathbb{E}[S_T])^2 = S_0^2 e^{2\mu T}(e^{\sigma^2 T} - 1)$

3.3 Black-Scholes pricing formulas

Lets find the discounted expectation $e^{-rT}\mathbb{E}[max(S_T - K, 0)]$ for a Vanilla Call option C. Please note, that we need to take in account that $\mu = r - d$, where d corresponds to a dividend yield.

$C = e^{-rT}\mathbb{E}[max(S_0 e^{(r-d-\frac{\sigma^2}{2})T + \sigma B_T} - K, 0)]$

$= \frac{1}{\sqrt{2\pi}} \int_{\frac{\ln(\frac{K}{S_0}) - (r-d-\frac{\sigma^2}{2})T}{\sigma\sqrt{T}}}^{\infty} (S_0 e^{\sigma\sqrt{T}x - (d + \frac{\sigma^2}{2}T)} - e^{-rT}K) e^{-\frac{x^2}{2}} dx$

$= e^{-dT} S_0 \frac{1}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{-\frac{(x-\sigma\sqrt{T})^2}{2}} dx - e^{-rT} K \frac{1}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{-\frac{x^2}{2}} dx$

$= e^{-dT} S_0 (1 - \mathcal{N}(-d_2 - \sigma\sqrt{T})) - e^{-rT} K (1 - \mathcal{N}(-d_2))$

$= e^{-dT} S_0 \mathcal{N}(d_1) - e^{-rT} K \mathcal{N}(d_2))$

with

- $d_1 = \frac{\ln(\frac{S_0}{K}) + (r-d+\frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$
- $d_2 = \frac{\ln(\frac{S_0}{K}) + (r-d-\frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$

With similar computation we get a formula for Put option:

$P = e^{-rT} K \mathcal{N}(-d_2)) - e^{-dT} S_0 \mathcal{N}(-d_1)$

Chapter 4

Applying FEM

4.1 Derivation of Black-Scholes PDE

Black-Scholes equation is a linear parabolic PDE with non-constant coefficients and non-homogenous boundary conditions and, possibly, non-differentiable or discontinuous final conditions. Here we give its derivation:
We take the expression of a Geometric Browinan motion with $a(S,t)$ and $b(S,t)$ as coefficients:

$$dS = a(S,t)dt + b(S,t)dB \tag{4.1}$$

Then we take a function $V \in C^2$ and with Ito's lemma we get :

$$dV(S(t),t) = (\frac{\partial V}{\partial t} + a(S,t)\frac{\partial V}{\partial S} + \frac{b(S,t)^2}{2}\frac{\partial^2 V}{\partial S^2})dt + b(S,t)\frac{\partial V}{\partial S}dB \tag{4.2}$$

As the asset price follows GBM, let's do following changes:

- $a(S,t) = S\mu$
- $b(S,t) = S\sigma$

That give us:

$$dV(S(t),t) = (\frac{\partial V}{\partial t} + S\mu\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2})dt + S\sigma\frac{\partial V}{\partial S}dB \tag{4.3}$$

Now we consider a portfolio with an option V and "short" on some asset value ΔS - in other words, we use so called, *delta-hedge portfolio* - the portfolio which value remains unchanged when small changes occur in the value of the underlying asset.

$$\Pi = V - \Delta S \tag{4.4}$$

or with time inserted:

$$d\Pi = dV - \Delta dS \tag{4.5}$$

Let's insert the last PDE and the GBM in our portfolio:

$$d\Pi = ((\frac{\partial V}{\partial t} + S\mu\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2})dt + S\sigma\frac{\partial V}{\partial S}dB - \Delta(S\mu dt + S\sigma dB) \tag{4.6}$$

Now, in order to remove the random term dB, we fix $\Delta = \frac{\partial V}{\partial S}$:

$$d\Pi = (\frac{\partial V}{\partial t} + S\mu\frac{\partial V}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - S\mu\frac{\partial V}{\partial S})dt = (\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2})dt \tag{4.7}$$

Another purely financial concept, not discussed here, is *arbitrage free market*, that is expressed as:

$$d\Pi = r\Pi dt \tag{4.8}$$

This concept is one of assumptions given in 3.1.
So , when substituting the portfolio and $d\Pi$ in the no-arbitrage expression, we get:

$$(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2})dt = r(V - \frac{\partial V}{\partial S}S)dt \tag{4.9}$$

The final PDE comes out after we d1vide both sides by dt:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{4.10}$$

4.2 Boundary conditions

- Call Option
  When fixing final, initial and boundary conditions we use economical arguments. Final condition is known and well posed : at time T worth 0 or $S-K$. If $S=0$ then $C=0$. If $S \longrightarrow \infty$, the option value is the asset price corrected by the dividend minus the exercise price corrected by the case if the holder had invested his money on the bank, so $C(S,t) = e^{-dT}S_0 - e^{-rT}K$. So our final problem is:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0 \tag{4.11}$$

$$\begin{aligned} C(0,t) &= 0 \\ C(S,t) &= e^{-dT}S_0 - e^{-rT}K \text{ when } S \longrightarrow \infty \\ C(S,T) &= max(S-E,0) \end{aligned}$$

- Put Option
  The same logic gives us conditions for put option.

$$\frac{\partial P}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} - rS\frac{\partial P}{\partial S} + rP = 0 \tag{4.12}$$

$$\begin{aligned} P(0,t) &= e^{-rT}K \\ P(S,t) &= 0 \text{ when } S \longrightarrow \infty \\ P(S,T) &= max(E-S,0) \end{aligned}$$

4.3 Weak Formulation

Consider a vanilla put option with maturity T and payoff function u_0. Let u be the pricing function, i.e., the price of the option at time Tt and when the spot price is S is $u(S,t)$. The function u solves the initial SDE:

$$\frac{\partial u}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - rS\frac{\partial u}{\partial S} + ru = 0 \tag{4.13}$$

Now lets multiply it by a test function, that lives in Weighted Sobolev space : $\forall u \in V, V = \{v \in L^2(\mathbb{R}_+) : S\frac{\partial v}{\partial S} \in L^2(\mathbb{R}_+)\}$ and integrate the whole expression. As always , we apply the integration by parts and obtain:

$$\frac{d}{dt}\left(\int_{\mathbb{R}+} u(S,t)w(S)dS\right) + a_t(v,w) = 0, \tag{4.14}$$

where a_t is a bilinear form defined as:

$$a_t(v,w) = \int_{\mathbb{R}+}\left(\frac{1}{2}S^2\sigma^2(S,t)\frac{\partial v}{\partial S}\frac{\partial w}{\partial S} + r(t)vw\right)ds + \int_{\mathbb{R}+}\left(-r(t) + \sigma^2(S,t) + S\sigma(S,t)\frac{\partial \sigma}{\partial S}(S,t)\right)S\frac{\partial v}{\partial S}wdS. \tag{4.15}$$

Let's check if u_0 is a unique solution.
Rq This proof is not full.

- As the volatility is always positive and bounded, we can conclude that a_t is continuous on V, càd there exists a positive constant M, such that for all $v, w \in V$,

$$|a_t(v,w)| \leq M|v|_V|w|_V. \tag{4.16}$$

– To prove the coercivity we use **Gârding's inequity**:

$$a_t(v,v) \geq C_1||v||_V^2 - C_2||v||_{L^2}^2. \tag{4.17}$$

So, under these conditions we may say that by Lax-Milgram if $u_0 \in L^2$ then it is the unique solution and we can write the weak formulation:

$$\begin{aligned} \text{Find } u \in \mathcal{C}^0([0;T]), u \in L^2 \cap V \\ \forall v \in V, (\frac{\partial u}{\partial t}(t), v) + a_t(u(t), v) = 0. \\ u(0,t) = 0 \\ u(S,t) = e^{-dT}S_0 - e^{-rT}K \text{ when } S \longrightarrow \infty \\ u(S,T) = max(S-E, 0) \end{aligned}$$

4.4 Black-Scholes and Heat equation

4.5 Existence and Uniqueness

Let $T > 0$. Let $b(.,.) : [0,T] \times \mathbb{R}^n \longrightarrow \mathbb{R}^n$ and $\sigma(.,.) : [0,T] \times \mathbb{R}^n \longrightarrow \mathbb{R}^{n \times m}$ be measurable functions, satisfying two following properties:

$$|b(t,x)| + |\sigma(t,x)| \leq C(1 + |x|), \tag{4.18}$$

with $x \in \mathbb{R}, t \in [0,T]$, for some constant C, (where $|\sigma|^2 = \sum |\sigma_{ij}|^2$) and

$$|b(t,x) - b(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq D|x-y|, \tag{4.19}$$

with $x, y \in \mathbb{R}, t \in [0,T]$, for some constant D.
Let Z be a random variable which is independent of the σ -algebra $\mathbb{F}_\infty^{(m)}$ generated by $B_s(.)$, $s > 0$, such that $\mathbb{E}[|Z|^2] < \infty$.
Then the stochastic SDE $dX_t = b(t,X_t)dt + \sigma(t,X_t)dB_t$ with $X_0 = Z$ has a unique solution X$_t(w) withthepropertythat X_t(w$
and $B_s(.)$, $s < t$ and $\mathcal{E}[\int_0^T |X_t|^2 dt] < \infty$

4.6 Mesh Adaptation and Delaunay triangulation

Mesh adaptation is an important tool in problems with free boundary. The procedure is done w.r.t. Delanay algorithm and keeps the error of interpolation bounded by:

$$||u - u_h|| < C||\nabla(\nabla u)h^2 \tag{4.20}$$

where $\nabla(\nabla u)$ is a Hessian matrix of u.
The, so called, Delaunay triangulation helps to create a "good" mesh : no obtuse triangles, neighbor triangles have more or less the same size.
In other words, the Delaunay triangulation create a mesh where for each edge the circle circumscribing one triangle does not contain the fourth vertex.

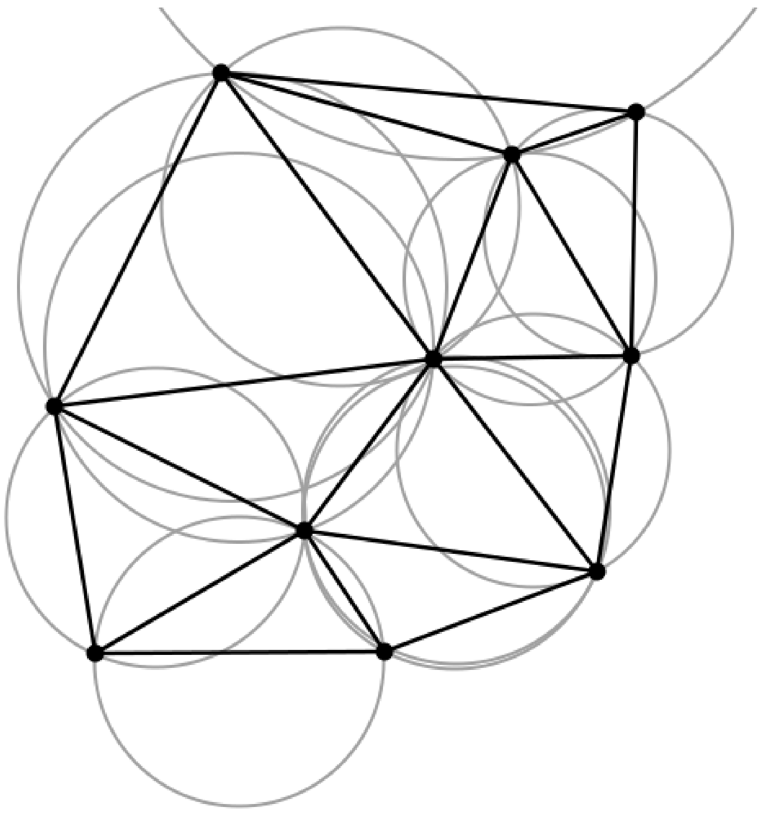


Figure 4.1: Delaunay triangulation

In freefem++ the mesh adaptation is easily done with **adaptmesh** command.

4.7 Numerical results

We solve a 2D PDE for an american put with $= 40, r = 5\%$.

4.7.1 FreeFem++

BlackScholes 1D

- Vanilla put, $K = 100$

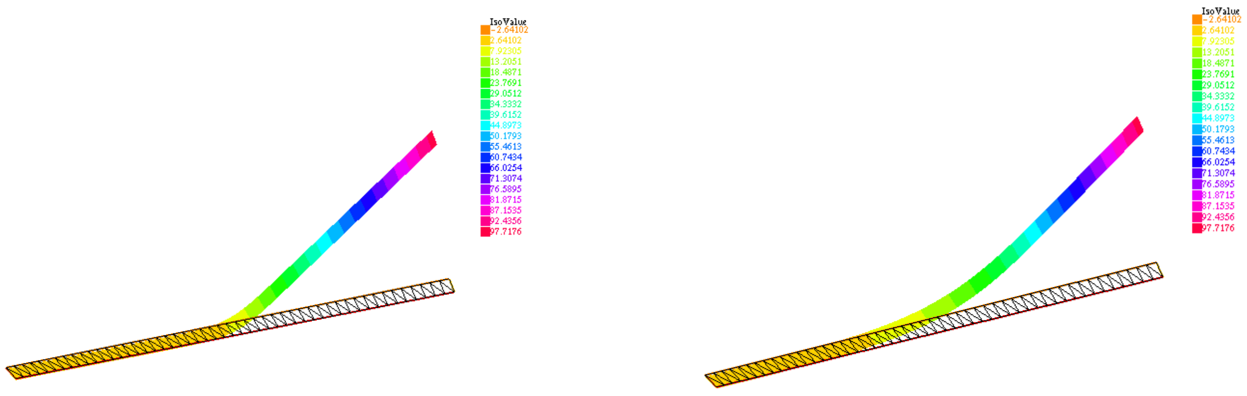


Figure 4.2: $\sigma = 0.1$ and $\sigma = 0.3$

– Delta for Vanilla put, $K = 100$

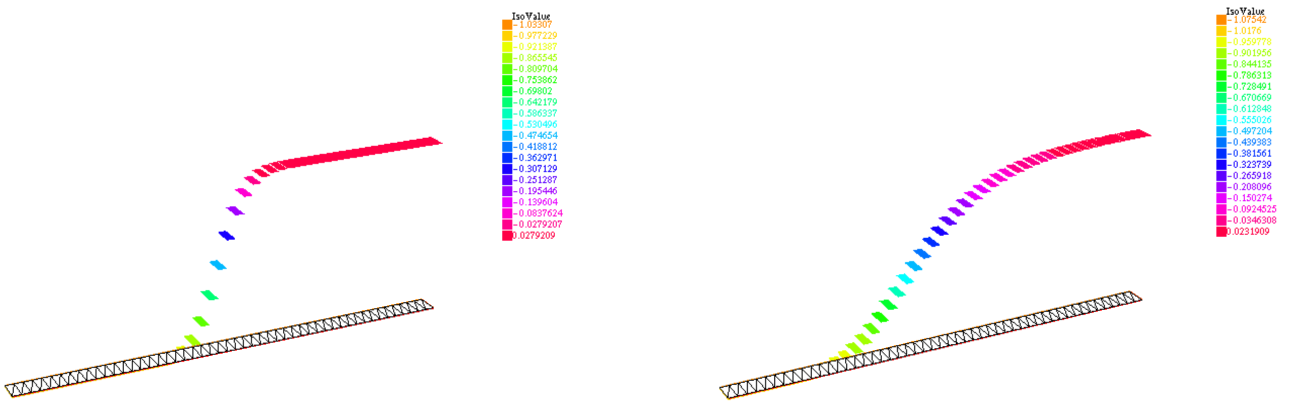


Figure 4.3: Delta for $\sigma = 0.1$ and $\sigma = 0.3$

– Barier put, $K = 100$

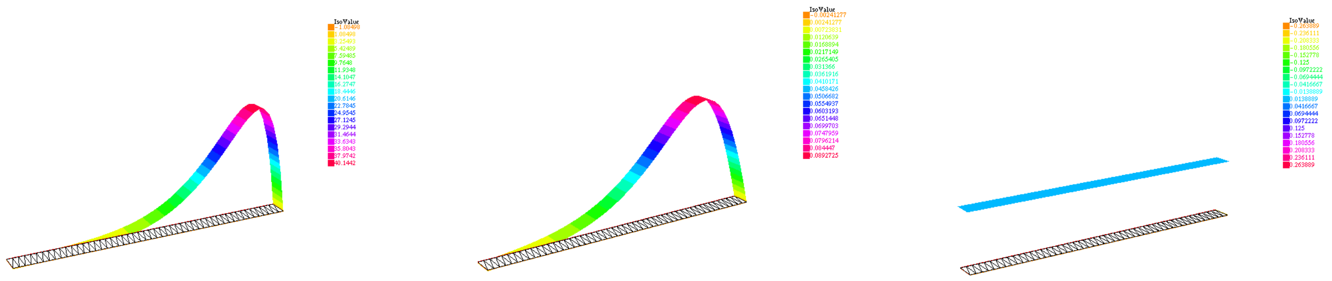


Figure 4.4: barriers $= 30, 90, 100$

– Asian put, $K = 100$

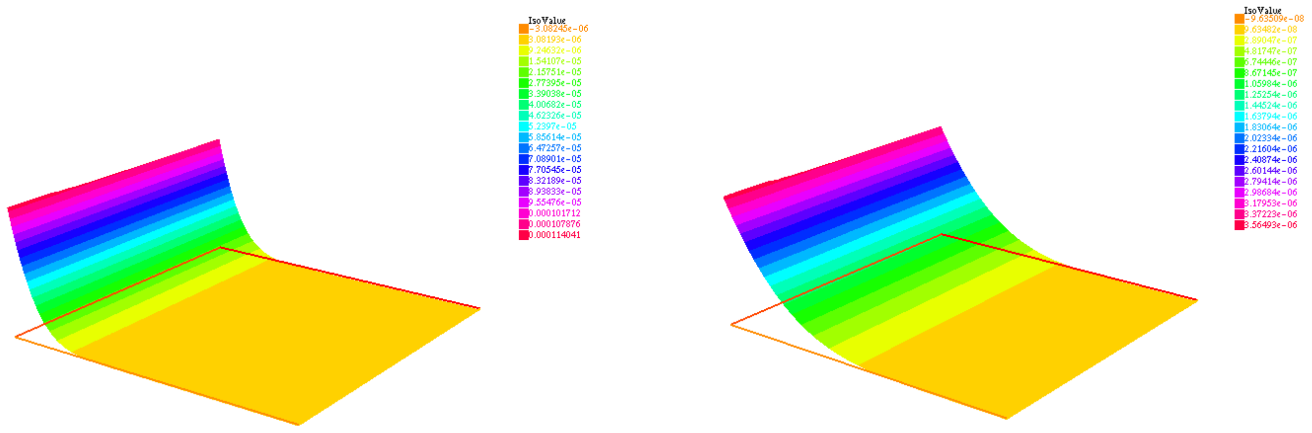


Figure 4.5: $\mu = 0.1$ and $\mu = 0.3$

BlackScholes 2D

– Classic asymmetric data

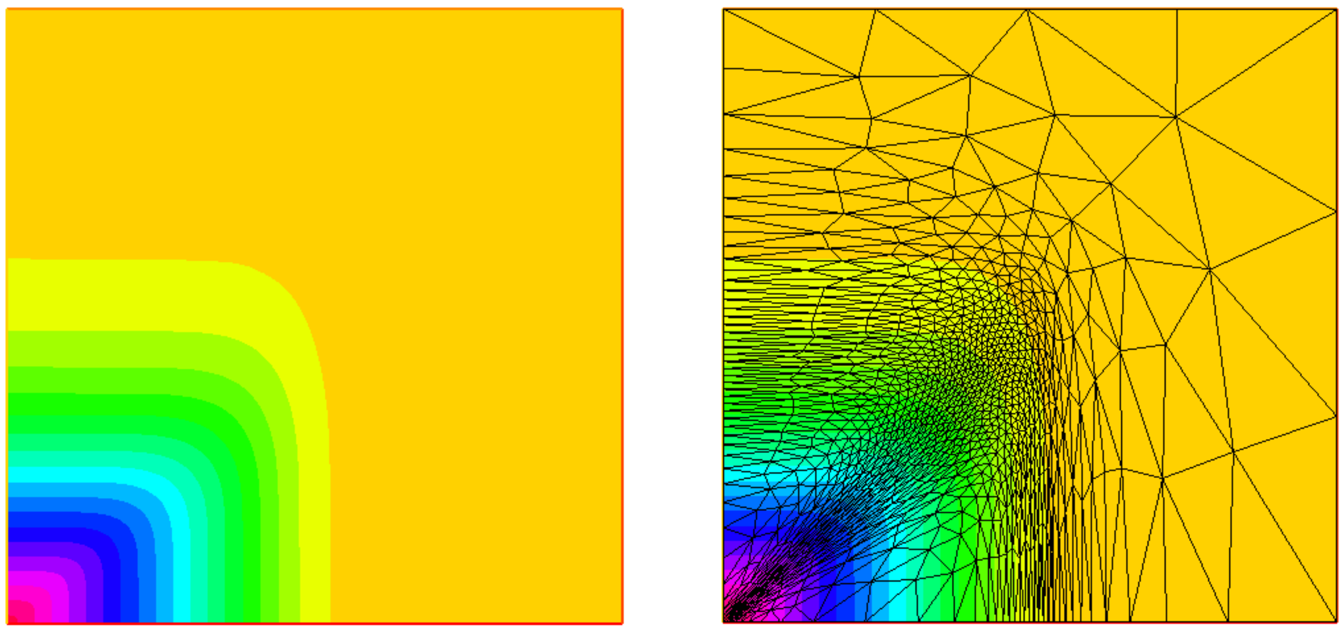


Figure 4.6: PDE : $\sigma_x = 0.1, \sigma_y = 0.3, \rho = 0.3$

– Low volatility with high correlation

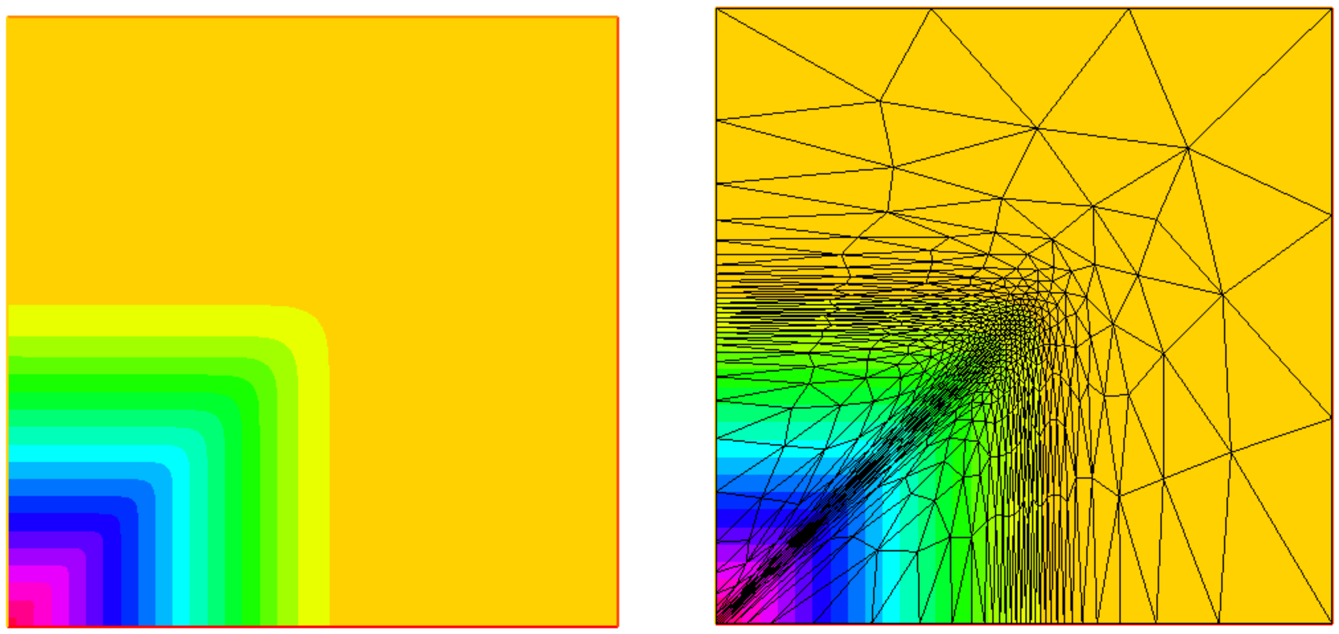


Figure 4.7: PDE : $\sigma_x = 0.1, \sigma_y = 0.1, \rho = 0.6$

– High volatility but low correlation

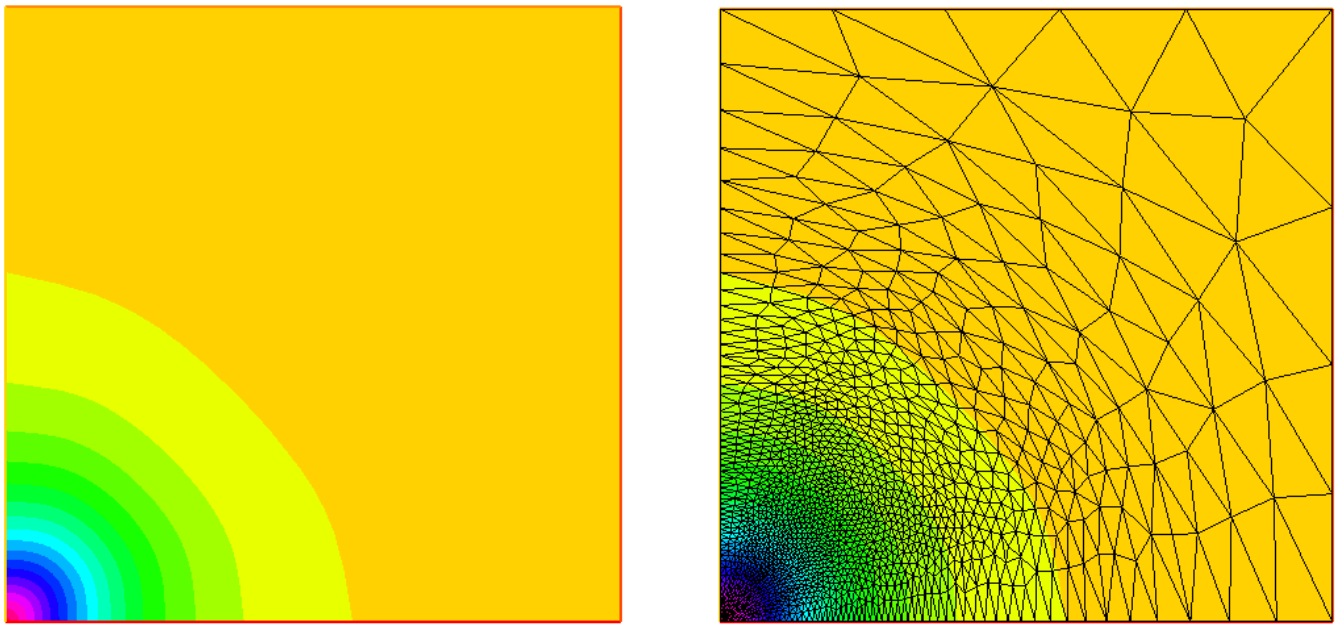


Figure 4.8: PDE : $\sigma_x = 0.6, \sigma_y = 0.6, \rho = 0.3$

– High volatility with high correlation

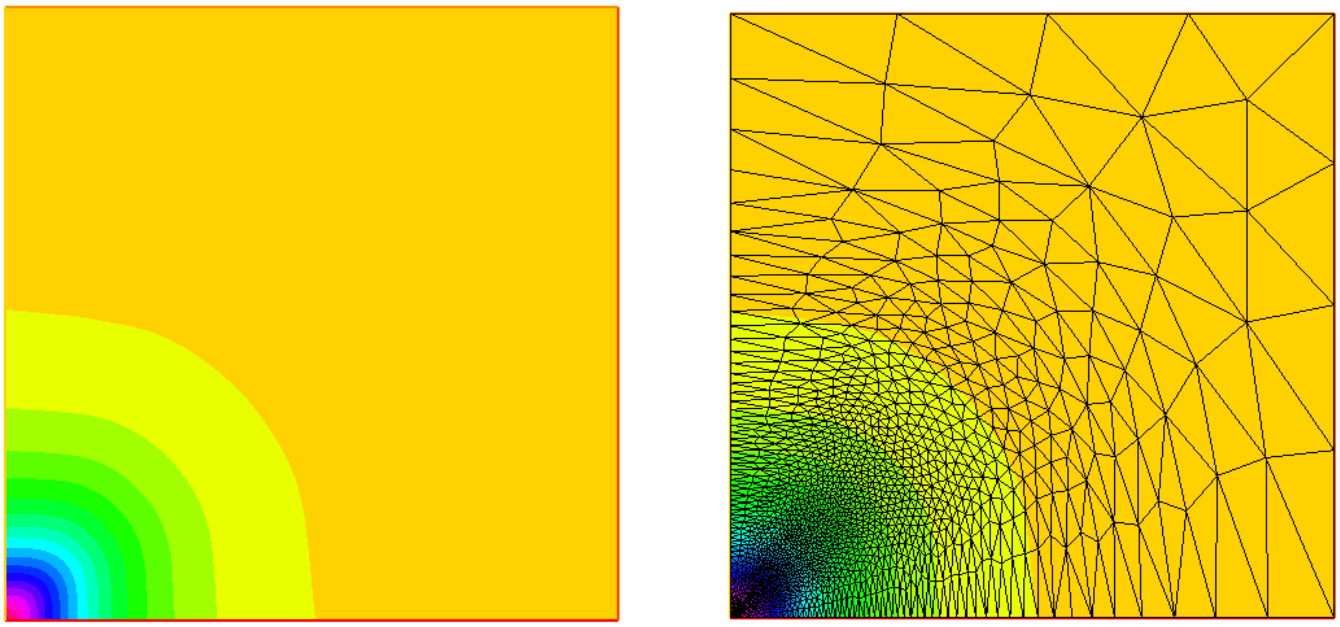


Figure 4.9: PDE :$\sigma_x = 0.6, \sigma_y = 0.6, \rho = 0.6$

4.7.2 Fell++

No export:

4.7.3 Finite Difference with C++

4.7.4 Greeks with C++ and Octave

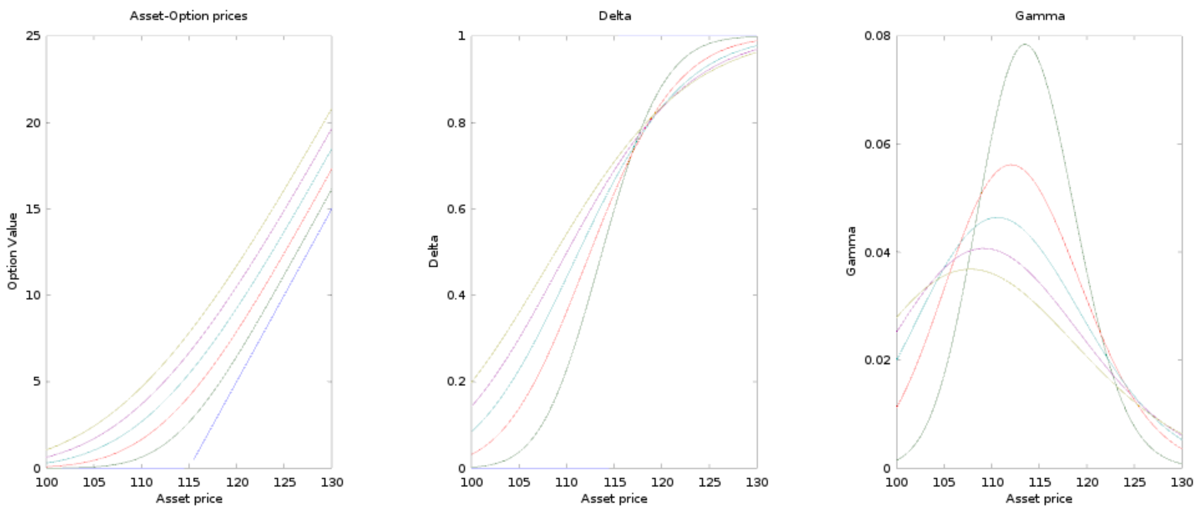
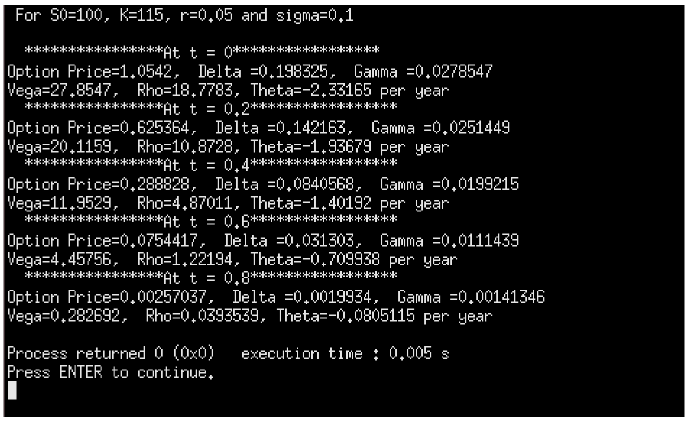


Figure 4.10: Greeks



```
 For S0=100, K=115, r=0.05 and sigma=0.1

  ****************At t = 0******************
Option Price=1.0542,  Delta =0.198325,  Gamma =0.0278547
Vega=27.8547,  Rho=18.7783, Theta=-2.33165 per year
  ****************At t = 0.2******************
Option Price=0.625364,  Delta =0.142163,  Gamma =0.0251449
Vega=20.1159,  Rho=10.8728, Theta=-1.93679 per year
  ****************At t = 0.4******************
Option Price=0.288828,  Delta =0.0840568,  Gamma =0.0199215
Vega=11.9529,  Rho=4.87011, Theta=-1.40192 per year
  ****************At t = 0.6******************
Option Price=0.0754417,  Delta =0.031303,  Gamma =0.0111439
Vega=4.45756,  Rho=1.22194, Theta=-0.709938 per year
  ****************At t = 0.8******************
Option Price=0.00257037,  Delta =0.0019934,  Gamma =0.00141346
Vega=0.282692,  Rho=0.0393539, Theta=-0.0805115 per year

Process returned 0 (0x0)   execution time : 0.005 s
Press ENTER to continue.
```

Figure 4.11: Greeks

Annexe

FreeFem++ code

1D case

```
//BlackSholes1D.edp
int Nx = 50, L=200, LL = 10,n;
real T=1, sigma = 0.3, r=0.05, K =100, dt = 0.01;
mesh th = square (Nx, 1, [L*x, LL*y]);
fespace  Vh(th, P1, periodic = [[1,x],[3,x]]);
fespace Vhdc(th,P1dc);

Vh u = max(K-x, 0.), v, uold;

problem BS(u, v, init=n)=
        int2d(th) (u*v*(r+1/dt)
        -x*(r-sigma^2/2)*dx(u)*v
        +dx(u)*dx(v)*(x*sigma)^2/2)
        -int2d(th)(uold*v/dt) + on (2, u=0);

for (n=0; n*dt <=T; n++)
{
uold=u;
BS;
}
Vhdc dxu=dx(u);
plot(u, th , value=1);
```

2D case

```
// file BlackScholes2D.edp
int m=30,L=80,LL=80, j=100;
real sigx=1., sigy=1., rho=0., r=0.05, K=40, dt=0.01;
mesh th=square(m,m,[L*x,LL*y]);
fespace Vh(th,P1);

Vh u=max(K-max(x,y),0.);
Vh xveloc, yveloc, v,uold;

for (int n=0; n*dt <= 1.0; n++)
{
if(j>20)  { th = adaptmesh(th,u,verbosity=1,abserror=1,nbjacoby=2,
err=0.001, nbvx=5000, omega=1.8, ratio=1.8, nbsmooth=3,
splitpbedge=1, maxsubdiv=5,rescaling=1) ;
j=0;
xveloc = -x*r+x*sigx^2+x*rho*sigx*sigy/2; //discount factor
yveloc = -y*r+y*sigy^2+y*rho*sigx*sigy/2;

u=u;

};
uold=u;
solve eq1(u,v,init=j,solver=LU) = int2d(th)( u*v*(r+1/dt)
+ dx(u)*dx(v)*(x*sigx)^2/2 + dy(u)*dy(v)*(y*sigy)^2/2
+ (dy(u)*dx(v) + dx(u)*dy(v))*rho*sigx*sigy*x*y/2)
- int2d(th)(v*convect([xveloc,yveloc],dt,uold)/dt) + on(2,3,u=0);
j=j+1;
};
plot(u, fill=1,wait=1,value=1);
```

Exotics: Barrier

```
//BlackSholesBarrier.edp
int Nx = 50, L=150, LL = 10,n;
real T=1, sigma = 0.3, r=0.05, K =100, dt = 0.01, b=50;
//mesh rules the payoff
mesh th = square (Nx, 1, [b + L*x, LL*y]);
fespace Vh(th, P1, periodic = [[1,x],[3,x]]);

Vh u = max(K-x, 0.), v, uold;

problem BS(u, v, init=n)=
        int2d(th) (u*v*(r+1/dt)
        -x*(r-sigma^2/2)*dx(u)*v
        +dx(u)*dx(v)*(x*sigma)^2/2)
        -int2d(th)(uold*v/dt) + on (2,4, u=0);

for (n=0; n*dt <=T; n++)
{
uold=u;
BS;
}

plot(u, th , value=1);
```

Exotics: Asian

```
//BlackSholesAsian.edp
int N = 25, L=250,Nmax = 30,n=0;
real verbo = 0, T=4, mu = 0.3, r=0.03, K =100, t, dt = T/Nmax;

mesh th = square (N, N, [ L*x, L*y]);
fespace Vh(th, P2);



func u1= (mu^2-r)*x;
func u2 = (y-x)/(T-t);

real g1,g2,g3;

Vh u = max(y-K, 0.), v, uold;
```

```



//explain <,>
problem BS(u, v, init=n)=
        int2d(th) (u*v*(r+1/dt)
        +dx(u)*dx(v)*x^2*mu^2/2)
        -int2d(th) (convect ([u1,u2], -dt,uold)*v/dt)
        -int1d(th,2)((g1*(y<g3) + g2*(y>=g3))*v);

for (t=0; t<T-2; t+=dt)
{
//artificial boundaties
g1 = 2*t*exp(-t*r)/T/(mu*L)^2;
g2 = 2*(1-exp(-t*r))/T/r/(mu*L)^2;
g3 = K*T/(T-t+0.001);
BS; n=1;
uold = u;
plot(u,  wait=0, value=1,  dim = 3, fill=1);
};
```

Feel++ code

1D case

BlackScholes.cpp

```
#include <feel/feel.hpp>

using namespace Feel;
inline
po::options_description
makeOptions()
{
po::options_description blackscholesoptions ("BS options");
blackscholesoptions.add_options()
        ("T", po::value<double>()->default_value(1.), "Time to maturity")
        ("K", po::value<double>()->default_value(1.), "Strike price")
        ("r", po::value<double>()->default_value(1.),"Interest rate")
        ("sigmax", po::value<double>()->default_value(1.),"sigma x")
        ("dt", po::value<double>()->default_value(1.),"dt")
        ;
        return  blackscholesoptions.add( backend_options("bs1d"));
}

int main (int argc, char* argv[])
{

        Environment env(_argc = argc, _argv=argv,
        _desc=makeOptions(),
        _about=about(_name="mymesh",
                    _author="Feel++ Consortium",
                    _email="feelpp-devel@feelpp.org"));

//loading a created mesh

        auto mesh = loadMesh(_mesh=new Mesh<Simplex<2>>);
```

```

//mesh adaptation - TODO

//determine space

        auto Xh = Pch<1>(mesh);

//initialisation of elements

    //    auto u=Xh->element("u");
        auto uold=Xh->element("uold");
        auto v = Xh->element("v");

//initialisation of parameters

        double T=doption(_name="T"); //Time to maturiry
        double K=doption(_name="K"); //Strike price
        double r=doption(_name="r"); //Interest rate
        double sigmax=doption(_name="sigmax"); //Volatility for option A
        double dt = doption(_name="dt"); //Time step

//additional functions uold!!!!

        auto u=max(K- Px(),0);

//initialisation of forms
        auto l = form1(_test=Xh);
        auto a = form2(_trial=Xh, _test=Xh);

//export
        auto e = exporter(_mesh = mesh);

//iteration


        for (double t=dt; t<T; t+=dt){
        uold = u;
        l.zero();
        a=integrate(_range=elements(mesh), _expr= ((idt(u)*id(v)*(r+(1/dt)))+
            dxt(u)*dx(v)*(sigmax*Px())*(sigmax*Px())/2 -Px()*(r-(sigmax*sigmax
            )/2)dxt(u)id(v);
        a.solve(_solution=u, _rhs=l, _name="bs1d");
        e->step(t)->add("u",u);
  e->save();
};

}
```

BlackScholes.cfg

```
T =0.5
K=50.
dt = 0.02
r = 0.5
sigmax = 0.2


[gmsh]
filename=square1d.geo
```

square1d.geo

```
h=0.2;
Point(1) = {0,0,0,h};
Point(2) = {0,1,0,h};
Point(3) = {1,1,0,h};
Point(4) = {1,0,0,h};
Line(1) = {1,2};
Line(2) = {2,3};
Line(3) = {3,4};
Line(4) = {4,1};
Line Loop(4) = { 1, 2, 3, 4};
Plane Surface(4) = {4};
Physical Line("in") = {1, 2, 4};
Physical Line("out") = {3};
Physical Surface("Omega") = {4};
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)

if (${CMAKE_SOURCE_DIR} STREQUAL ${CMAKE_CURRENT_SOURCE_DIR})
        find_package(Feel++
                PATHS $ENV{FEELPP_DIR}/share/feel/cmake/modules
                /usr/share/feel/cmake/modules
                /usr/local/share/feel/cmake modules
                opt/share/feel/cmake/modules
        )
        if(NOT FEELPP_FOUND)
                message(FATAL_ERROR "Feel++ was not found on your system.
                    Make sure to install it and specify the FEELPP_DIR to
                    reference the illation directory.")
        endif()
endif()
feelpp_add_application(
 BS1d
  SRCS BlackScholes1d.cpp
  GEO square1d.geo
  DEFS A_DEF=2
  CFG BlackScholes1d.cfg )
```

2D case

BlackScholes.cpp

```
#include <feel/feel.hpp>

using namespace Feel;
inline
po::options_description
makeOptions()
{
po::options_description blackscholesoptions ("BS options");
blackscholesoptions.add_options()
        ("T", po::value<double>()->default_value(1.), "Time to maturity")
        ("K", po::value<double>()->default_value(1.), "Strike price")
        ("r", po::value<double>()->default_value(1.),"Interest rate")
        ("sigmax", po::value<double>()->default_value(1.),"sigma x")
        ("sigmay", po::value<double>()->default_value(1.),"sigma y")
        ("rho", po::value<double>()->default_value(1.),"rho")
        ("mu", po::value<double>()->default_value(1.),"mu")
        ("dt", po::value<double>()->default_value(1.),"dt")
        ;
        return  blackscholesoptions.add( backend_options("bs"));
}

int main (int argc, char* argv[])
{

        Environment env(_argc = argc, _argv=argv,
        _desc=makeOptions(),
        _about=about(_name="mymesh",
                     _author="Feel++ Consortium",
                     _email="feelpp-devel@feelpp.org"));

//loading a created mesh

        auto mesh = loadMesh(_mesh=new Mesh<Simplex<2>>);

//mesh adaptation - TODO

//determine space

        auto Xh = Pch<1>(mesh);

//initialisation of elements

     //   auto u=Xh->element("u");
        auto uold=Xh->element("uold");
        auto v = Xh->element("v");

//initialisation of parameters

        double T=doption(_name="T"); //Time to maturiry
        double K=doption(_name="K"); //Strike price
        double r=doption(_name="r"); //Interest rate
        double sigmax=doption(_name="sigmax"); //Volatility for option A
        double sigmay=doption(_name="sigmay"); //Volatility for option B
        double rho = doption(_name="rho"); //Correlation between A and B
        double mu = doption(_name="mu"); //Drift rate
        double dt = doption(_name="dt"); //Time step

//additional functions uold!!!!

        auto u=max(K-max( Px(), Py() ),0);
        auto xvel = -Px()*r + Px()*sigmax*sigmax+Px()*rho*sigmax*sigmay/2;
        auto yvel = -Py()*r + Py()*sigmay*sigmay+Py()*rho*sigmax*sigmay/2;

//initialisation of forms
        auto l = form1(_test=Xh);
        auto a = form2(_trial=Xh, _test=Xh);

//export
    //    auto e = exporter(_mesh = mesh);



```

```
//iteration

        for (double t=dt; t<T; t+=dt){
        l.zero();
        uold = u;
        a=integrate(_range=elements(mesh),_expr =( (gradt(u)*vec(xvel,yvel))*
            id(v)));
        a+=integrate(_range=elements(mesh), _expr= ((idt(u)*id(v)*(r+(1/dt)))
            +dxt(u)*dx(v)*(sigmax*Px())*(sigmax*Px())/2+dyt(u)*dy(v)*(sigmay*
            Py())*(sigmay*Py())/2+(dyt(u)*dx(v) + dxt(u)*dy(v))*rho*sigmax*
            sigmay*Px()*Py()/2) );

        a+=on(_range=markedfaces(mesh, "out"), _rhs=l, _element=u,_expr=cst
            (0) );
        a+=on(_range=markedfaces(mesh,"in"), _rhs=l, _element=u,_expr=uold);
        a.solve(_solution=u, _rhs=l, _name="bs");

        e->step(t)->add("u",u);
        e->save();
};
}
```

BlackScholes.cfg

```
T =0.5
K=50.
dt = 0.02
r = 0.5
sigmax = 0.2
sigmay=0.3
rho = 0.3
mu = 0.2

[gmsh]
filename=square.geo
```

square.geo

```
h=0.2;
Point(1) = {0,0,0,h};
Point(2) = {0,0.5, h/2};
Point(3) = {0,1,0,h};
Point(4) = {1,1,0,h};
Point(5) = {1,0,0,h};
Point(6) = {0.5,0,0,h/2 };
Line(1) = {1,2};
Line(2) = {2,3};
Line(3) = {3,4};
Line(4) = {4,5};
Line(5) = {5,6};
Line(6) = {6,1};
Line Loop(6) = { 1, 2, 3, 4,5,6};
Plane Surface(6) = {6};
Physical Line("in") = {1, 2, 6, 5};
Physical Line("out") = {3,4};
Physical Surface("Omega") = {6};
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
```

```

if  (${CMAKE_SOURCE_DIR} STREQUAL ${CMAKE_CURRENT_SOURCE_DIR})
        find_package(Feel++
                PATHS $ENV{FEELPP_DIR}/share/feel/cmake/modules
                /usr/share/feel/cmake/modules
                /usr/local/share/feel/cmake modules
                opt/share/feel/cmake/modules
        )
        if(NOT FEELPP_FOUND)
                message(FATAL_ERROR "Feel++ was not found on your system.
                    Make sure to install it and specify the FEELPP_DIR to
                    reference the illation directory.")
        endif()
endif()
feelpp_add_application(
 BS
  SRCS BlackScholes.cpp
  GEO square.geo
  DEFS A_DEF=2
  CFG BlackScholes.cfg )
```

4.7.5 C++

Greeks

```
#include <vector>
#include <stdio.h>
#include <math.h>
#include <iostream>

using namespace std;
double f(double x) {
        double pi =  4.0*atan(1.0);
        return exp(-x*x*0.5)/sqrt(2*pi);
}


// Boole's Rule
double Boole(double StartPoint, double EndPoint, int n) {
        vector<double> X(n+1, 0.0);
        vector<double> Y(n+1, 0.0);
        double delta_x = (EndPoint - StartPoint)/double(n);
        for (int i=0; i<=n; i++) {
                X[i] = StartPoint + i*delta_x;
                Y[i] = f(X[i]);
        }
        double sum = 0;
        for (int t=0; t<=(n-1)/4; t++) {
                int ind = 4*t;
            sum += (1/45.0)*(14*Y[ind] + 64*Y[ind+1] + 24*Y[ind+2] + 64*Y[ind
                +3] + 14*Y[ind+4])*delta_x;
        }
        return sum;
}

// N(0,1) cdf by Boole's Rule
double N(double x) {
        return Boole(-10.0, x, 240);
```

```
}

// Black-Scholes Call Price
double BSPrice(double S, double K, double T, double r, double sigma, char
    OpType)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        double call = S*N(d) - exp(-r*T)*K*N(d - sigma*sqrt(T));
        if (OpType=='C')
                return call;
        else
                return call - S + K*exp(-r*T);
}
// Black-Scholes Delta
double BSDelta(double S, double K, double T, double r, double sigma, char
    OpType)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        if (OpType=='C')
                return N(d);
        else
                return N(d) - 1;
}

// Black-Scholes Gamma
double BSGamma(double S, double K, double T, double r, double sigma)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        return f(d) / S / sigma / sqrt(T);
}

// Black-Scholes Vega
double BSVega(double S, double K, double T, double r, double sigma)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        return S*f(d)*sqrt(T);
}

// Black-Scholes Rho
double BSRho(double S, double K, double T, double r, double sigma, char
    OpType)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        if (OpType=='C')
        return T*K*exp(-r*T)*N(d - sigma*sqrt(T));
        else
                return -T*K*exp(-r*T)*N(sigma*sqrt(T) - d);
}
// Black-Scholes Theta
double BSTheta(double S, double K, double T, double r, double sigma, char
    OpType)
{
        double d = (log(S/K) + T*(r + 0.5*sigma*sigma)) / (sigma*sqrt(T));
        if (OpType=='C')
                return -S*f(d)*sigma/2/sqrt(T) - r*K*exp(-r*T)*N(d - sigma*
                    sqrt(T));
        else
                return -S*f(d)*sigma/2/sqrt(T) + r*K*exp(-r*T)*N(sigma*sqrt(T
                    ) - d);
}

int main()
{
        double S = 100;         // Stock Price
        double K = 115.0;   // Strike Price
    double dt= 0.2;     // Time step
        double T = 1;           // Years to maturity
        double r = 0.05;        // Risk free interest rate
        double sigma = 0.10;// Yearly volatility
        char OpType = 'C';      // 'C'all or 'P'ut

        cout << " For S0=" <<S<<", K="<<K<<", r="<<r<<" and sigma="<<  sigma
           << endl;
        cout << "                                                  " << endl;
        for (double t=0; t<T; t=t+dt)
        {
    cout << "  ****************At t = " << t << "*****************" << endl;
    cout << "Option Price=" << BSPrice(S,K,T-t,r,sigma,OpType) <<",  Delta ="
        << BSDelta(S,K,T-t,r,sigma,OpType) <<",  Gamma =" << BSGamma(S,K,T-t,
       r,sigma)<< endl;
    cout << "Vega=" << BSVega(S,K,T-t,r,sigma)<<",  Rho=" << BSRho(S,K,T-t,r,
       sigma,OpType)<< ", Theta=" << BSTheta(S,K,T-t,r,sigma,OpType)<< " per
       year"<< endl;
        }
}
```

Explicit Finite Difference Method

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
using namespace std;
double explicitCallOption(double S0,double X,double T,double r,double sigma,
    int iMax,int jMax)
{
  // declare and initialise local variables (ds,dt)
  double S_max=2*X;
  double dS=S_max/jMax;
  double dt=T/iMax;
  // create storage for the stock price and option price (old and new)
  vector<double> S(jMax+1),vOld(jMax+1),vNew(jMax+1);
  // setup and initialise the stock price
  for(int j=0;j<=jMax;j++)
  {
    S[j] = j*dS;
  }
  // setup and initialise the final conditions on the option price
  for(int j=0;j<=jMax;j++)
  {
    vOld[j] = max(S[j]-X,0.);
    vNew[j] = max(S[j]-X,0.);
  }
  // loop through time levels, setting the option price at each grid point,
      and also on the boundaries
  for(int i=iMax-1;i>=0;i--)
  {
    // apply boundary condition at S=0
    vNew[0] = 0.;
    for(int j=1;j<=jMax-1;j++)
    {
      double A,B,C;
      A=0.5*sigma*sigma*j*j*dt+0.5*r*j*dt;
      B=1.-sigma*sigma*j*j*dt;
      C=0.5*sigma*sigma*j*j*dt-0.5*r*j*dt;
      vNew[j] = 1./(1.+r*dt)*(A*vOld[j+1]+B*vOld[j]+C*vOld[j-1]);
    }
    // apply boundary condition at S=S_max
    vNew[jMax] = S[jMax] - X*exp(-r*(T-i*dt));
    // set old values to new
    vOld=vNew;
  }
  // get j* such that S_0 \in [ j*dS , (j*+1)dS ]
  int jstar;
  jstar = S0/dS;
  double sum=0.;
  // Lagrange2 polynomial interpolation
  sum = sum + (S0 - S[jstar+1])/(S[jstar]-S[jstar+1])*vNew[jstar];
  sum = sum + (S0 - S[jstar])/(S[jstar+1]-S[jstar])*vNew[jstar+1];
  return sum;
}

int main()
{

  double S0=1.639,X=2.,T=1.,r=0.05,sigma=0.4;

  int iMax=4,jMax=4;
  cout << explicitCallOption(S0,X,T,r,sigma,iMax,jMax) << endl;

}
```