A SEM-NI CODE FOR PRICING EUROPEAN OPTIONS WITH EXPONENTIAL LÉVY MODELS

Alessandro Montinari, Matr.787002
Introduction

The aim of this project is the implementation of a numerical code written in C++ to solve partial integro-differential equations (PIDE) deriving from a quantitative finance problem: option pricing with exponential Lévy models. In particular, we shall develop a spectral element method with numerical integration (SEM-NI) to solve a particular class of PIDE:

\[
\frac{\partial f}{\partial t} = L f + I f \quad \text{in} \quad (0, T] \times \Omega, \quad f = h \quad \text{in} \quad [0, T] \times \Omega^c
\]

Where the operators \( L \) and \( I \) are defined as:

\[
L f(t, S) = \mu(S)\partial_S f(t, S) + \partial_S(\sigma(S)\partial_S f(t, S)) - \gamma f(t, S)
\]

\[
I f(t, S) = \int_{\mathbb{R}} f(t, Se^y)\nu(y)dy
\]

and the spatial domain is an open interval: \( \Omega = (S_l, S_r) \subset \mathbb{R} \).

We will outline the advantages associated with the use of Gaussian mesh grids that permit precise evaluation of the integral operator \( I \) with low computational cost in comparison with the use of finite difference methods (widely used in computational finance) or finite element methods, in the particular case of the Merton model. We will also see that this method fails when used with singular Lévy measures, because of the low regularity of these measures.

- In Section 1 we present the financial problem and a particular case in which we can find an analytical solution for the pricing of European options.
- In Section 2 we present the stochastic framework and the main results. We also show the PIDE that we want to solve numerically with our C++ code.
- In Section 3 we analyze some techniques used to reduce the problem in a more tractable one, suitable to be solved with numerical methods. After that we present the spectral element method, with particular emphasis on the advantages of this method in comparison with other numerical methods used by practitioners.
- In Section 4 there is an accurate analysis of the C++ code.
- Section 5 is devoted to the application of the code to some classical problems: pricing of European, barrier and binary option with the Merton Jump Diffusion model and Variance Gamma model.
# Contents

1 Options and the Black & Scholes model 3

2 Lévy processes and Exponential Lévy models 5
   2.1 Some known processes ........................................... 6
   2.1.1 Merton Model ............................ 6
   2.1.2 Kou Model ................................. 6
   2.1.3 Variance Gamma .................................. 7
   2.2 Pricing with Exponential Lévy Models ................. 7

3 Numerical Methods for PIDEs 8
   3.1 Domain reduction and Asmussen-Rosinski approximation ........ 8
   3.2 Boundary Conditions ....................................... 9
   3.3 SEM-NI ........................................... 10
   3.4 Comparison with FDM and FEM ................................. 13
   3.5 Drawbacks of the use of SEM-NI with Lévy models .......... 13

4 C++ code for SEM-NI 15
   4.1 SemPideUtility.hpp ...................................... 15
   4.2 SemPideSolver class ...................................... 15
   4.2.1 solve member function ............................... 16
   4.3 Mesh class ........................................... 17
   4.4 BaseFunction class ....................................... 18
   4.5 BFLinearCombination ...................................... 19
   4.6 Integrator and GaussLegendre classes ..................... 20
   4.7 Problem class ........................................... 21
   4.8 How to use SemPide Library ................................ 21

5 Application to standard options 23
   5.1 Convergence Test for Merton Model .......................... 23
   5.2 Application to Up&Out and Binary Option ........................ 23
   5.3 Variance Gamma model .................................... 24

Conclusions 27

References 28
1 Options and the Black & Scholes model

Let us consider a financial market in which there is an asset (e.g. shares of a company) with price $S(t)$. An European call option is a contract that gives to the holder of the option the right but not the obligation to buy one share of the asset at an exercise date $T$ (also called maturity) at the price $E$. For example, suppose that we have an option with maturity $T=1$ year and with strike price $E=100$. If at $T$ the share price is $S(T) = 120 > E$ we exercise the option, obtaining a gain of $S(T) - E = 20$. If the share price falls to $S(T) = 80 < E$ we don’t exercise our right, so we do nothing, with a gain of 0. We can summarize saying that the European call option has a payoff given by the function:

$$\Psi(S(T)) = (S(T) - K)^+$$

Now we want to build a model to give a "fair price" to the option.

Let $(\Omega, \mathcal{F}, P)$ be a probability space with a filtration $\mathcal{F} = (\mathcal{F}_t)_{t \in [0,T]}$. Suppose that there are no transaction costs of trading and that the price $S(t)$ follows a Geometric Brownian Motion:

$$S(t) = S(0) + \int_0^t \mu S(u) du + \int_0^t \sigma S(u) dW(u)$$

where $\mu$ and $\sigma$ are real constants called respectively local mean rate of return and volatility, while $W(t)$ is a Wiener process on $(\Omega, \mathcal{F}, \mathbb{P})$. On the market there is also another financial asset called riskless asset (in contrast with $S(t)$, the risky asset) or money market account that follows:

$$B(t) = B(0) + \int_0^t r B(u) du = B(0)e^{rt}$$

A natural interpretation of a riskless asset is that it corresponds to a bank with short rate of interest $r$.

Definition 1.1. An arbitrage possibility is a portolio with value in time $V(t)$ such that:

$$V(0) = 0$$
$$P(V(T) \geq 0) = 1$$
$$P(V(T) > 0) > 0$$

We say that the market is arbitrage free if there are no arbitrage possibilities.

An arbitrage is essentially the possibility of making a positive amount of money out of nothing taking any risk. We will assume that the market is arbitrage free. Given this assumption we can enunciate the central theorem in mathematical finance: with this result we can definitively give a price to our call option.

Theorem 1.1 (Fundamental Theorem of Mathematical Finance). The market is arbitrage free if and only if there exists a probability measure $Q$ equivalent to $P$ such that the price of all financial assets in the market $\Pi(t)$ satisfies:

$$\mathbb{E}_Q[\frac{\Pi(T)}{B(T)} | \mathcal{F}_t] = \frac{\Pi(t)}{B(t)}$$

In other words, $\frac{\Pi(t)}{B(t)}$ is a martingale with respect to the filtration $\mathcal{F}$ under the probability $Q$.

Denoting the call price with $C(t)$, we have

$$C(t) = \mathbb{E}_Q^P \frac{B(t)}{B(T)} C(T) | \mathcal{F}_t] = e^{-r(T-t)} \mathbb{E}_Q^P [(S(T) - K)^+ | \mathcal{F}_t]$$

We remark that, under $Q$, $S(t)/B(t)$ is also a martingale, so using Girsanov theorem we can derive:

$$S(t) = S(0) + \int_0^t r S(u) du + \int_0^t \sigma S(u) dW^Q(u)$$
and using some properties of the stochastic integral we have:

\[ S(t) = S(0)e^{X(t)}, \quad X(t) = rt + \sigma W(t) \sim N((r - \frac{1}{2}\sigma^2)t, \sigma^2t) \]

Now we can derive with a simple argument (that is omitted, see [Bjork]) the famous Black and Scholes formula:

\[
C(0, S(0)) = S(0)N(d_1) - e^{-rT}Ee^{-rT}N(d_2) \tag{5}
\]

\[
d_1 = \frac{ln(S(0)/E) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T} \tag{6}
\]

in which \(N(\cdot)\) is the cumulative distribution function of a standard normal probability distribution.

Under some assumption on the market and on the assets dynamic we have derived a closed form solution to our problem. Unfortunately there are some empirical evidences that the Black & Scholes model is too simple to describe a financial market. These empirical facts called \textit{stylized facts} can be summarized in:

- the price \(S(t)\) is discontinous (in contrast with B&S model in which the price is a continuous process)
- heavy tails in the distribution of log returns
- asymmetric distribution of log returns
- markets are incomplete
- log returns are not independent
- there are transaction costs of trading

In this project we shall analyze an extension to the B&S model that permits to solve some of these problems. Due to some mathematical difficulties, in many case we will not be able to find an analytical solution like in the B&S model, so we will need numerical methods. In the next section we present the Exponential Lévy model and the PIDE associated with that.
2 Lévy processes and Exponential Lévy models

Definition 2.1. A stochastic process $X = (X(t))_{t \geq 0}$ with cadlag trajectories\(^1\) on a filtered space $(\Omega, \mathcal{F}, \mathbb{F} = (\mathcal{F}_t)_{t \geq 0}, P)$ is a Lévy process if

- $X(0) = 0$
- $X$ has stationary and independent increments
- $X$ is continuous in probability

Let now state some useful theorems and properties on Lévy processes.

First we define the Lévy Measure:

$$\nu(A) = \mathbb{E}[\# \{ t \in [0, 1] : \Delta X(t) \neq 0, \Delta X(t) \in A \}], \ A \in \mathcal{B}(\mathbb{R})$$

the Lévy measure of a process counts the mean number of jumps occurring in the time interval $[0, 1]$.

Theorem 2.1 (Lévy-Ito decomposition). Given $X = (X(t))_{t \geq 0}$, a Lévy process with Lévy measure $\nu$, then

1. \[ \int_{\mathbb{R}} \min(1, x^2) \nu(dx) < \infty \]

2. \[ X(t) = \gamma t + \sigma W(t) + X^1(t) + \lim_{\epsilon \to 0} \tilde{X}^\epsilon(t) \]

where $W(t)$ is a Wiener process, $X^1(t)$ is a compound Poisson with jumps whose size is bigger than 1, and $\tilde{X}^\epsilon(t)$ is a compensated compound Poisson with jumps whose size belongs to the interval $(\epsilon, 1]$. Furthermore $W(t)$, $X^1(t), \tilde{X}^\epsilon(t)$ are independent.

The Lévy-Ito decomposition entails that for every Lévy process there exist two constant $\gamma, \sigma$ and a positive measure $\nu$ that uniquely determine its distribution. The triplet $(\gamma, \sigma, \nu)$ is called characteristic triplet of the process $X$. We stress that $W, X^1, \tilde{X}^\epsilon$ are simple processes well known in stochastic analysis. When the process has only a finite number of jump in any finite interval $[0, t]$, it can be decomposed in only 3 simple terms:

$$X(t) = \gamma t + \sigma W(t) + \sum_{n=0}^{N(t)} Z_n$$

where $N(t)$ is a Poisson process and $(Z_n)_{n \in \mathbb{N}}$ is a sequence of IID variables with known law. A process of this kind is called Jump Diffusion. Note that in general a Lévy process has only a finite number of large jumps, i.e.

$$\forall \epsilon > 0 \quad \# \{ \text{jumps with size} > \epsilon \} < \infty$$

and that the total number of jumps is at most countably. This can, in some sense, justify the Lévy-Ito decomposition: a Lévy process is the sum of a continuous process (the drift term $\gamma$ and the Wiener process), a process with a finite number of large jumps and a process with infinite(at most countably) small bounded jumps. This property will be useful in the sequel, when we will need to approximate a Lévy process to solve numerically the PIDE associated with the call option price.

Note that property (1) in the Lévy-Ito decomposition theorem doesn’t assure that the Lévy measure is finite in $\mathbb{R}$, in particular we don’t know if integrable function with respect to the Lebesgue measure are also integrable with respect to the Lévy measure. $\nu$ has a singularity near zero of order two:

\(^1\)A functions in $[0,T]$ is said to be cadlag if it is right continuous with left limit on $[0,T]$
this may cause some numerical instabilities. However, in same case, things are not so bad: when a process has a jump part with only finite variation its Lévy measure satisfies:

\[ \int_{|x|<1} |x| \nu(dx) < \infty \]

In the Jump Diffusion case we have also \( \nu(\mathbb{R}) < \infty \).

The following proposition characterizes uniquely a Lévy process using its characteristic function.

**Theorem 2.2** (Lévy-Khinchin representation theorem). Let \((X(t))_{t \geq 0}\) be a Lévy process with triple \((\gamma, \sigma, \nu)\). Then

\[ \mathbb{E}[e^{iuX(t)}] = e^{t \Psi_X(u)} \]

with

\[ \Psi(u) = i\gamma u - \frac{\sigma^2 u^2}{2} + \int_{\mathbb{R}} (e^{iux} - 1 - iuxI_{|x| \leq 1}) \nu(dx) \]

The function \( \Psi(u) \) is called the **characteristic exponent of** \( X \), and sometime is written as \( \Psi_X(u) \).

The proof of this theorem is an immediate consequence of the Lévy-Ito decomposition theorem.

To conclude this section we point out that a process of the form \( Y(t) = e^{X(t)} \), where \((X(t))_{t \geq 0}\) is a Lévy process with characteristic exponent given by \( \Psi_X(u) \), \((Y(t))_{t \geq 0}\), is a Martingale if and only if

\[ \Psi_X(-i) = 0, \quad \int_{|x|>1} e^x \nu(dx) < \infty \]

**2.1 Some known processes**

In this section we present some Lévy processes used in Mathematical Finance. Some of them will be used in the sequel to test our C++ code.

**2.1.1 Merton Model**

In the Merton model log-prices are modeled using a Jump Diffusion process of the form:

\[ X(t) = \gamma t + \sigma W(t) + \sum_{n=0}^{N(t)} Z_n \]

where \( \gamma \in \mathbb{R} \), \(N(t)\) is a Poisson process with intensity \( \lambda \) and \((Z_n)_{n \in \mathbb{N}}\) is a sequence of IID standard normal random variables: \( Z_n \sim N(\mu, \delta^2) \). It is rather simple to show that in this case the Lévy measure admits a density with respect to the Lebesgue measure with this form:

\[ \nu(x) = \frac{\lambda}{\sqrt{2\pi\delta^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\delta^2} \right\} \]

For this model the probability density function of \( X(t) \) has a closed form:

\[ p_t(x) = e^{-\lambda t} \sum_{k=0}^{\infty} \frac{(\lambda t)^k}{k!} \exp\left\{ -\frac{(x-\gamma t-k\mu)^2}{2\delta^2} \right\} \]

that is simply a series of normal densities. For this reason we can explicitly solve the problem of pricing European options with an exponentially dumped series expansion of Black & Scholes prices. We will use this formula to test the convergence of numerical methods.

**2.1.2 Kou Model**

Kou model is based on a Jump Diffusion process like the Merton model, but the \( Z_n \) variables have double exponential distribution. It can be proved that in this case the Lévy measure has the form:

\[ \nu(x) = p \lambda_+ e^{-\lambda_+ x} 1_{x>0} + (1-p) \lambda_- e^{-\lambda_- x} 1_{x<0} \]

where \( \lambda_+, \lambda_- > 0 \) and \( p \in [0, 1] \) is the probability for a jump to be positive.

For this kind of model closed form doesn’t exist for log-price probability density functions and European option prices. For this reason we must rely on numerical methods.
2.1.3 Variance Gamma

Variance Gamma is a particular model where we have a subordinated Brownian Motion, namely

\[ X(t) = \gamma t + \tilde{B}(S(t)) \]

where

\[ \tilde{B}(t) = \mu t + \sigma W(t) \]

The idea consists in considering a process of the type Brownian motion + drift and change the flowing time using a stochastic clock described by a positive increasing process \( S(t) \) called subordinator. While for the preceding processes the Lévy measure gives only a finite number of jump in each finite subinterval \([0,t]\), for the Variance Gamma model we have infinite activity. However this model has finite variation. Lévy density is given by

\[ \nu(x) = \frac{1}{k|x|} e^{Ax-B|x|} \]

with

\[ A = \frac{\theta \sigma^2}{2}, \quad B = \sqrt{\theta^2 + 2 \sigma^2} \frac{\sigma^2}{k} \]

Note that, as we pointed out before, \( \nu \) has a singularity of first order (because has finite variation) in \( x = 0 \).

2.2 Pricing with Exponential Lévy Models

From this moment we will consider only the equivalent martingale measure \( Q \). Let us consider a market model in which the price of the risky asset is described by an exponential Lévy model:

\[ S(t) = S(0) e^{rt + X(t)} \]

where \((X(t))_{t \geq 0}\) is a Lévy process with triple \((\gamma, \sigma, \nu)\) and such that:

\[ \int_{|x|>1} e^{2x} \nu(dx) < \infty \]

with also \( \Psi_X(-i) = 0 \). Thanks to the first condition \( S(t) \) we have that the first and the second moments of \( S(t) \) are finite. The second condition instead implies that \( S(t) \) is a martingale under \( Q \).

If we consider a derivative with payoff given by the function \( \Phi(S) \), from the Fundamental Theorem of Mathematical Finance (1.1) we know that:

\[ C(t, S) = e^{-r(T-t)} E_Q[\Phi(S(T)) | S(t) = S] \]

The following proposition characterizes the price \( \Pi(t, S) \) in term of a linear Partial Integro-Differential Equation (PIDE) of the second order:

**Theorem 2.3.** Let \( \Phi(S) \) be Lipschitz and suppose that \( \sigma > 0 \). Then the function

\[ C : [0, T] \times [0, \infty) \to \mathbb{R} \]

\( (t, S) \to C(t, S) = e^{-r(T-t)} E_Q[\Phi(S(T)) | S(t) = S] \)

is continuous on \([0, T] \times [0, \infty]\) and \( C^{1,2} \) on \([0, T] \times [0, \infty]\) and satisfy the equation:

\[ \frac{\partial C}{\partial t}(t, S) + rS \frac{\partial C}{\partial S}(t, S) + \frac{\sigma^2 S^2}{2} \frac{\partial^2 C}{\partial S^2}(t, S) - rC(t, S) \]

\[ + \int_{\mathbb{R}} [C(t, Se^y) - C(t, S) - S(e^y - 1) \frac{\partial C}{\partial S}(t, S)] \nu(dy) = 0 \] (7)

on \([0, T] \times [0, \infty]\) with final condition

\[ C(T, S) = \Psi(S) \quad \forall S > 0 \]

In the next section we will deal with a numerical method to solve this equation.
3 Numerical Methods for PIDEs

This section aims to find a numerical method to solve the following PIDE:

\[
\frac{\partial C}{\partial t}(t, S) + rS \frac{\partial C}{\partial S}(t, S) + \frac{\sigma^2 S^2}{2} \frac{\partial^2 C}{\partial S^2}(t, S) - rC(t, S) + \int_\mathbb{R} [C(t, Se^y) - C(t, S) - S(e^y - 1) \frac{\partial C}{\partial S}(t, S)] \nu(y) dy = 0
\]

(8)

on \([0, T] \times [0, \infty]\) with terminal condition:

\[C(T, S) = \Psi(S) \quad \forall S > 0\]

3.1 Domain reduction and Asmussen-Rosinski approximation

The PIDE (8) has unbounded domain in the variable \(S\), so the first thing to do is to truncate the domain and give some appropriate boundary conditions. We must stress that the integro-differential operator associated with (8) is of non local type, in fact the function \(C(t, S)\) appears in the equation in every points of the spatial domain \(S \in [0, \infty)\). For this reason, using a truncated domain, we need a condition on every point of the complementary of \(\mathbb{R}\). Let us consider a closed interval \([S_{\text{min}}, S_{\text{max}}]\) and the equation (8) on \([0, T] \times [S_{\text{min}}, S_{\text{max}}]\) with boundary global condition:

\[C(t, S) = C_l(t, S), \quad 0 \leq S \leq S_{\text{min}}\]

\[C(t, S) = C_r(t, S), \quad S \geq S_{\text{max}}\]

\[C(T, S) = \Psi(S), \quad \forall S > 0\]

A priori with don’t need the truncation on the left side of the domain. Indeed, we will see in the following that in the neighbor of \(S = 0\) we have some numerical instabilities due to our numerical scheme. However we have always a good proxy for the left boundary condition. We will find appropriate functions \(C_l\) and \(C_r\) using financial and economical arguments. Let us see what we can do with the integral term. Suppose for the moment that the Lévy measure is of finite activity (\(\nu(\mathbb{R}) < \infty\)), so we can divide the three term without problems deriving from the singularity of \(\nu\):

\[\int_\mathbb{R} [C(t, Se^y) - C(t, S) - S(e^y - 1) \frac{\partial C}{\partial S}(t, S)] \nu(y) dy =\]

\[\int_\mathbb{R} C(t, Se^y) \nu(y) dy - C(t, S) \int_\mathbb{R} \nu(y) dy - S \int_\mathbb{R} \frac{\partial C}{\partial S}(t, S) \nu(y) dy =\]

\[\int_0^{\infty} \frac{C(t, Z)}{Z} \nu(\log(\frac{Z}{S})) dZ + \lambda C(t, S) + \alpha S \frac{\partial C}{\partial S}(t, S)\]

(9)

with \(\lambda = \int_\mathbb{R} \nu(y) dy\) and \(\alpha = \int_\mathbb{R} (e^y - 1) \nu(y) dy\). For a moment we must focus on the change of variable in (9).

In the general form, we need two mesh to discretize the problem: one for the differential operator (on \([0, \infty)\)) and one for the integral operator (that is on \(\mathbb{R}\)): the two grid never coincide! This could be very troublesome cause we need expensive interpolation to find and transfer values of the function from one grid to another. Furthermore this procedure is to be iterated in every points of the differential grid because in the integral term we have \(C(t, Se^y)\) which depends on \(S\). This change of variable solve the two problem in one shot: we have the same domain for each operator (the solution outside \([S_{\text{min}}, S_{\text{max}}]\) is given by \(C_l\) and \(C_r\) and we don’t have any dependence from \(S\) for the function appearing in the integral operator. The cost to pay is the numerical instability in the neighbor of \(S = 0\). So we have:

\[\int_0^{\infty} \frac{C(t, Z)}{Z} \nu(\log(\frac{Z}{S})) dZ = \int_{S_{\text{min}}}^{S_{\text{max}}} \frac{C(t, Z)}{Z} \nu(\log(\frac{Z}{S})) dZ + \]

\[+ \int_{S_{\text{min}}}^{S_{\text{max}}} \frac{C_l(t, Z)}{Z} \nu(\log(\frac{Z}{S})) dZ + \int_{S_{\text{max}}}^{\infty} \frac{C_r(t, Z)}{Z} \nu(\log(\frac{Z}{S})) dZ\]
Finally the equation (8) takes the form:
\[
\frac{\partial C}{\partial t}(t, S) + (r - \alpha)S \frac{\partial C}{\partial S}(t, S) + \frac{\sigma^2 S^2}{2} \frac{\partial^2 C}{\partial S^2}(t, S) - (r + \lambda)C(t, S) \\
+ \int_{S_{\min}}^{S_{\max}} \frac{C(t, Z)}{Z} \nu(\log(Z/S))dZ + f(t, S) = 0, \quad (t, S) \in [0, T] \times [S_{\min}, S_{\max}]
\]

\[
C(t, S) = C_t(t, S), \quad 0 \leq S \leq S_{\min} \\
C(t, S) = C_r(t, S), \quad S \geq S_{\max} \\
C(T, S) = \Psi(S), \quad \forall S > 0
\]

\[
f(t, S) = \int_{0}^{S_{\min}} \frac{C_l(t, Z)}{Z} \nu(\log(Z/S))dZ + \int_{S_{\max}}^{\infty} \frac{C_r(t, Z)}{Z} \nu(\log(Z/S))dZ
\]

In this way we solved some problems occurring when we have to rely on numerical methods. We have left the question about the separation of the three terms in the integral operator: in general this operation is forbidden because the terms left have no meaning due to the singularity in \(x = 0\).

We must return to the Lévy-Ito Decomposition 2.1:
\[
X(t) = \gamma t + \sigma W(t) + X^l(t) + \lim_{\epsilon \to 0} \tilde{X}^\epsilon(t)
\]

Problems arise from \(\lim_{\epsilon \to 0} \tilde{X}^\epsilon(t)\) that has an infinite number of small jumps and its Lévy measure is:
\[
\tilde{\nu}(x) = \nu(x)1_{|x| < 1}
\]

The idea is to approximate the process \(X(t)\) with a Jump Diffusion with a truncation in the limit and using a finite, small, \(\epsilon\). The resulting process
\[
X^\epsilon(t) = \gamma t + \sigma W(t) + X^l(t) + \tilde{X}^\epsilon(t)
\]

has Lévy measure given by
\[
\nu^\epsilon(x) = \nu(x)1_{|x| > \epsilon}
\]

Some theorem, that we don’t quote here for briefness, states that in most case this approximation is really rough. Asmussen and Rosinski found that, under some simple hypothesis, we have a better approximation using the process:
\[
X^\epsilon(t) = \gamma t + (\sigma + \sigma_\epsilon) W(t) + X^l(t) + \tilde{X}^\epsilon(t)
\]

with
\[
\sigma_\epsilon^2 = \int_{|x| < \epsilon} x^2 \nu(x)dx
\]

The idea (justified by theory) under this result is that small jumps are often similar to a sort of white noise. So we can substitute in the PIDE \((\sigma + \sigma_\epsilon)\) in place of \(\sigma\) and use the truncated Lévy measure \(\nu^\epsilon(x) = \nu(x)1_{|x| > \epsilon}\) instead of \(\nu(x)\). We will use Asmussen-Rosiski approximation with the Variance Gamma model.

### 3.2 Boundary Conditions

Boundary conditions are chosen according to the type of option we consider. Let us consider some cases:

- In the case of a **European call** \(\Phi(S) = (S - E)^+\). If in a particular moment we have \(S(t) << E\), probably we will have \(S\) smaller than \(E\) at maturity. For this reason we can choose on the left hand side 0:
  \[
  C_l(t, S) = 0
  \]
If $S$ is enough over $E$ ($S(t) > E$), probably we will have the same situation at maturity. At maturity we will have a gain of $\Phi(S(T)) = S(T) - E$. An option with such payoff has price given by $S(t) - Ee^{-r(T-t)}$. For this reason we have

$$C_r(t, S) = S - Ee^{-r(T-t)}$$

- If we consider an **European put** we have $\Phi(S) = (E - S)^+$. With the same reasoning we obtain:

$$C_l(t, S) = Ee^{-r(T-t)} - S$$

$$C_r(t, S) = 0$$

- A **Barrier option** of type **out** is an option that has no value at maturity if the underlying hits a barrier $B$ at any time before maturity. If $E > B$ the option is said **down & out**, if $B > E$ is said **up & out**, otherwise if we have two barrier is said **double knock out**. Barrier option could be call or put with different payoff function. Using some results from stochastic analysis it can be proved that the related PIDE has a bounded domain in the barrier with homogeneous Dirichlet boundary conditions.

For example if we take a down & out call we have:

$$S_{min} = B$$

$$\Phi(S) = (S - E)^+1_{S>B}$$

$$C_l(t, S) = 0$$

$$C_r(t, S) = S - Ee^{-r(T-t)}$$

### 3.3 SEM-NI

To present the spectral element method we generalize a little the equation (10). We assume that boundary condition are given by (11).

$$\frac{\partial C}{\partial t} (t, S) + \mu(S) \frac{\partial C}{\partial S} (t, S) + \frac{\partial}{\partial S} (\sigma(S) \frac{\partial C}{\partial S} (t, S)) - \gamma C(t, S)$$

$$+ \int_{S_{min}}^{S_{max}} \frac{C(t, Z)}{Z} \nu(\log(\frac{Z}{S}))dZ + f(t, S) = 0, \quad (t, S) \in [0, T] \times [S_{min}, S_{max}]$$

$$f(t, S) = \int_{0}^{S_{min}} \frac{C_l(t, Z)}{Z} \nu(\log(\frac{Z}{S}))dZ + \int_{S_{max}}^{\infty} \frac{C_r(t, Z)}{Z} \nu(\log(\frac{Z}{S}))dZ$$

Obviously to return to the first equation (12) we have to choose appropriate coefficients $\mu, \sigma, \gamma$. For a moment we consider the equation with homogeneous boundary conditions. First we multiply the equation (12) for a test function $v = v(S) \in V \equiv H^1_0(S_{min}, S_{max})$ and we integrate on $(S_{min}, S_{max})$. We are looking for a function $t \rightarrow C(t, S) \in V$ such that

$$\int_{S_{min}}^{S_{max}} \frac{\partial C(t)}{\partial t} vdS + a(C(t), v) + i(C(t), v) + F(v, t) = 0$$

where we have

$$a(C(t), v) = \int_{S_{min}}^{S_{max}} \mu C(t) \frac{\partial C(t)}{\partial S} vdS - \int_{S_{min}}^{S_{max}} \sigma \frac{\partial C(t)}{\partial S} \partial_v dS - \int_{S_{min}}^{S_{max}} \gamma C(t) v dS$$

$$i(C(t), v) = \int_{S_{min}}^{S_{max}} \left( \int_{S_{min}}^{S_{max}} \frac{C(t, Z)}{Z} \nu(\log(\frac{Z}{S}))dZ \right) v(S) dS$$

$$F(v, t) = \int_{S_{min}}^{S_{max}} f(t)vdS$$

10
Now we divide the domain \( \Omega = [S_{\min}, S_{\max}] \) in \( N \) elements \( \Omega = \bigcup_{k=1}^{N} \Omega_k \). Let

\[
V_N = \{ v \in C^0(\bar{\Omega}) : v|_{\Omega_k} = \Phi_k \in \mathbb{Q}^{n_k}, v(S_{\min}) = v(S_{\max}) = 0 \}
\]

\[
\mathbb{Q}^{n_k} = \{ p = \sum_{k=0}^{n_k} \alpha_k x^k, \alpha_k \in \mathbb{R} \}
\]

In simple worlds we approximate the space \( V \) with the space of continuous piecewise polynomial functions. Note that we may have different degree in each element. To be more precise, in each element we consider Gauss-Lobatto-Legendre nodes \( x_{k,j}, k = 1, \ldots, N, j = 1, \ldots, n_k \) and we choose as \( \Phi_k \) Lagrange polynomials on \( x_{k,j} \).

We have to find a function \( t \to u_N(t) \in V_N \) such that

\[
\int_{S_{\min}}^{S_{\max}} \frac{\partial u_N(t)}{\partial t} v_N dS + a(u_N(t), v_N) + i(u_N(t), v_N) + F(v_N, t) = 0, \quad v_N \in V_N
\]

The problem now is that in most cases we don’t have a closed form for the operators

\[
a(u_N(t), v_N(t)), \quad i(u_N(t), v_N(t)) \quad \text{and for } F(v_N, t) \text{ so we choose to approximate the integral in (13,14,15) with gaussian quadrature. Formulae for this kind of approximation are well known in literature, so we omit them here. We focus our attention on the integral operator:}
\]

\[
i(\Phi_l, \Phi_m) = \int_{S_{\min}}^{S_{\max}} \left( \int_{S_{\min}}^{S_{\max}} \frac{\Phi_l(Z)}{Z} \nu(\log(Z/S)) dZ \right) \Phi_m(S) dS
\]

where \( \Phi_l, \Phi_m \) are local Lagrange polynomials on Gauss-Lobatto-Legendre on the respective elements. Because of the local nature of the base functions we have

\[
i(\Phi_l, \Phi_m) = \int_{\Omega_l} \left( \int_{\Omega_j} \frac{\Phi_l(Z)}{Z} \nu(\log(Z/S)) dZ \right) \Phi_m(S) dS
\]

if \( \Phi_l \) has support on \( \Omega_l \) and \( \Phi_m \) on \( \Omega_j \). We immediately note that, even if \( i \neq j \), \( i(\Phi_l, \Phi_m) \neq 0 \). This reflect the non local nature of the operator \( i(\cdot, \cdot) \): the corresponding matrix will be full in all
cases. Noting that $\Phi_l$ is a Lagrange polynomial on the Gauss-Lobatto-Legendre nodes in $\Omega$, using Gauss Quadrature with the same nodes gives us:

$$
i(\Phi_l, \Phi_m) = \int_{\Omega} \{ \frac{1}{Z^j} \nu (\log(\frac{Z^j}{S})) \} \Phi_m(S) dS$$

where $\Phi_l(Z^j) = 1$. With the same reasoning for $\Phi_m$ we have

$$
i(\Phi_l, \Phi_m) = \frac{1}{Z^j} \nu (\log(\frac{Z^j}{S_m})) \} w_{Z^j} w_{S_m}$$

Note that $S^m, Z^j \in \Omega$. The use of Gauss quadrature simplify drastically the calculation of the matrix associated with the integral operator. Note that this is possible only because we have done the change of variables in (9) and we have the same mesh for the discretized domain $\Omega$ and the integral operator:

$$
i(C(t), \Phi_m) = \int_{S_{min}}^{S_{max}} \{ \int_{S_{min}}^{S_{max}} C(t, Z) \frac{1}{Z} \nu (\log(\frac{Z}{S})) dZ \} \Phi_m(S) dS$$

$$= \int_{S_{min}}^{S_{max}} \{ \int_{S_{min}}^{S_{max}} \sum_{i=1}^{M} C_i(t) \frac{1}{Z} \nu (\log(\frac{Z}{S})) dZ \} \Phi_m(S) dS$$

$$= \sum_{i=1}^{M} C_i(t) \int_{S_{min}}^{S_{max}} \{ \int_{S_{min}}^{S_{max}} \frac{1}{Z} \nu (\log(\frac{Z}{S})) dZ \} \Phi_m(S) dS$$

Where $C_i(t)$ are the same coefficients that appear in the differential operator $a(\cdot, \cdot)$. To completely discretize the equation we choose theta method for temporal discretization obtaining a linear system of the form:

$$\frac{1}{\Delta t} (M - \theta S - \theta I) u^{j+1} = \frac{1}{\Delta t} (M + (1 - \theta) S + (1 - \theta) I) u^j + M (\theta f^{j+1} + (1 - \theta) f^j)$$

where $M$ is the Mass matrix, $S$ is the Stiffness matrix, $I$ is the Integral matrix. The cost associated with the inversion of a dense matrix (because $I$ is dense) is high, however if $N$ is low $S$ is near to be dense so we don’t lose too much. When the number of elements is high we may prefer a splitting method where the linear system is:

$$\frac{1}{\Delta t} (M - \theta S) u^{j+1} = \frac{1}{\Delta t} (M + (1 - \theta) S + I) u^j + M (\theta f^{j+1} + (1 - \theta) f^j)$$

In our case we will use low number of elements (say 4-5) and high degree of the polynomials. To conclude we must take in consideration the boundary condition. Note that $f^j$ consider the solution outside the domain, so we have no problem with it. We consider the linear system associated with homogeneous Neumann condition where the base functions space is

$$V_N = \{ v \in C^0(\Omega) : v|_{\Omega_h} = \Phi_h \in \mathbb{Q}^{n_h} \}$$

and we impose the boundary condition directly on the matrix and on the constant term. Suppose that the linear system is $K \times K$:

$$\frac{1}{\Delta t} (M - \theta S - \theta I)_{1,j} = \delta_{j=1}$$

$$\frac{1}{\Delta t} (M - \theta S - \theta I)_{K,j} = \delta_{j=K}$$

If we write

$$b^j = \frac{1}{\Delta t} (M + (1 - \theta) S + (1 - \theta) I) u^j + M (\theta f^{j+1} + (1 - \theta) f^j)$$

then

$$b^j_1 = C_i(t_j, S_{min}), \ b^j_K = C_i(t_j, S_{max})$$
3.4 Comparison with FDM and FEM

The common practice in Finance is the use of Finite Difference Methods which are relatively simple to implement and well known in literature. These methods have the drawback that the convergence speed is low and to achieve high accuracy you need to increase a lot the computational cost.

In FDM we have a grid on which we discretize our solution approximating first and second order derivatives with finite differences. If we consider a simple \( \theta \) method we have

\[
\frac{C_{i,j}^{n+1} - C_{i,j}^n}{\Delta t} + \theta \left\{ \mu(S_i) \frac{C_{i+1,j}^{n+1} - C_{i-1,j}^{n+1}}{2\Delta S} \right. \\
+ \frac{1}{2} \sigma(S_{i+1}) C_{i+1,j}^{n+1} (\sigma(S_{i+1}) - \sigma(S_{i-1})) + \sigma(S_{i-1}) C_{i-1,j}^{n+1} - \gamma C_{i,j}^{n+1} \} \\
+ (1 - \theta) \left\{ \mu(S_i) \frac{C_{i,j}^{n+1} - C_{i,j}^{n-1}}{2\Delta S} \right. \\
+ \frac{1}{2} \sigma(S_{i+1}) C_{i,j}^{n+1} (\sigma(S_{i+1}) - \sigma(S_{i-1})) + \sigma(S_{i-1}) C_{i,j}^{n-1} - \gamma C_{i,j}^{n-1} \} \\
\left. + \int_{S_{\text{min}}}^{S_{\text{max}}} \frac{C_j(Z)}{Z} \nu(\log(Z/S)) dZ + f(t_j, S_i) = 0 \right.
\]

Note that the linear system associated with this kind of discretization is tridiagonal. For this reason we must treat explicitly the integral term. If we treat implicitly the integral term we have a dense matrix and we lose a lot of computational efficiency. Unfortunately, in this way, we lose the second order accuracy of the Crank Nicolson method.

We have to note that, in general, the integral term must be evaluated with sufficient accuracy or we lose second order convergence in space: here we have a tradeoff between grid flexibility and integral discretization accuracy. We want to raffinate the grid in the neighborhood of the strike price, were the initial condition has a discontinuity in the first derivatives, but if we do so, we don’t necessary gain accuracy in the discretization of the integral term. In this case we have two choises:

- Use a uniform grid: the evaluation of the integral term is done using simpson rule in the grid points. We don’t need interpolation. We lost in flexibility, but we don’t lose nothig in speed.

- Use a non uniform grid: this time we need to interpolate the solution values to use simpson rule or, if we want, gauss integration. Note that this is very expensive numerically for two reason: interpolation is expansive, and it has to be done an high number of times, one for each time step, since the method isn’t second order in time.

Another thing that slow down the method is the evaluation of the \( f(t,S) \) term. Using FDM we have to evaluate this term in a lot of grid point if compared with SEM-NI(because the FDM is only second order accurate in space) and a lot of time in space.

For this reasons the FDM is to be prefered only when we need a very low level of accuracy, instead for mid-high accuracy SEM-NI is preferable.

Only two worlds about Finite Element Methods. In the case of piecewise linear base functions the reasoning is the same as the FDM: for high accuracy the method is very time and memory consuming. When we use higher order polynomials we have some benefits in term of convergence but we loose the semplicity of FEM(piecewise linear) and FDM methods, besides if we want to recover the second order convergence in time we need to evaluate the integral matrix: without gauss grid we don’t have all the semplification shown above for the SEM-NI. Further, we have a dense matrix but we don’t have exponential convergence of SEM-NI.

3.5 Drawbacks of the use of SEM-NI with Lévy models

As we will see in the applications section, SEM-NI could fail when we have a singular Lévy measure. From the theory we know that Gaussian integration achieve high rate of convergence only when we have a very regular integrand function. In the case of the Merton model the Lévy measure has a \( C^\infty \) Gaussian density function so SEM-NI achieve exponential rate of convergence. Problems occur when we have a Lévy measure with non regular density as in the case of the Kou model.
and Variance Gamma model. In the case of Variance Gamma the Lévy measure is singular in 0 and we have seen that Asmussen Rosinski approximation permits to have a non singular measure. However the truncated measure is discontinuous! So Gaussian integration is not the best way to approximate the integral operator. In the Kou model the Lévy measure has finite activity but it is only continuous so we don’t have the same regularity as the Merton model even if the measure is not singular.
In conclusion we will see that with the Merton model this method is very performant but fails to achieve high rate of convergence even with finite activity models.
4  C++ code for SEM-NI

The C++ program for partial integro differential equation has been developed using only an external library: Eigen library for linear algebra. The code is structured in class objects, one related to each others, with a central class that is SemPideSolver, which has a method, named solve() that is used to solve the problem. In the following we will show in detail the classes interface. The code has also a reference manual generated using Doxygen. To develop this library we follow the lines given in the file LifeV Development Guidelines to have a clean code that can be simply expanded and or used in other libraries.

4.1 SemPideUtility.hpp

In this file we have some utility that we use in the rest of the library.

```cpp
namespace SEM_NI{

typedef Eigen::MatrixXd matrix_Type;
typedef Eigen::VectorXd vector_Type;
typedef Eigen::VectorXi intVector_Type;
typedef double real;
typedef int Int;
typedef unsigned int UInt;
enum class ODESolver_Type{ImplicitEuler,ExplicitEuler,CrankNicolson,RungeKutta};

double const gllpi=3.141592653589793;

void gll_generator(vector_Type &w,vector_Type &x,UInt N);
void gl_generator(vector_Type &w,vector_Type &x,UInt N);
}
```

First there are some typedef used to have some flexibility if we want to change the vector type used in the code or if we want to change precision for some reason. There is also an enum class that is used to decide which discretization we want to use to solve the problem. Finally there are two function, gll_generator, gl_generator that are used to generate Gauss-Legenedre-Lobatto and Gauss-Legendre nodes and weights to build the mesh and for the gauss quadrature.

4.2 SemPideSolver class

This is the core of the library. Let’s see the interface:

```cpp
namespace SEM_NI{

class SemPideSolver
{

public:

typedef Eigen::PartialPivLU<Eigen::MatrixXd> linearSolver_Type;
SemPideSolver(Problem const &myproblem,UInt n,UInt N,UInt Nt,real xMin,real xMax,real T);
SemPideSolver(Problem const &myproblem,ODESolver_Type myodesolver,
        Mesh const &mymesh,UInt Nt,real T);
void solve();

void printSolution(std::ostream &OUTPUT)const;
real evaluateSolution(real x)const;
const Mesh& getMesh() const{return M_mesh;};
const vector_Type& getSolution()const{return M_solution;};
}
```
private:
Problem M_problem;
Mesh M_mesh;
BaseFunctions M_baseFun;
Integrator<GaussLegendre> M_int;
ODESolver_Type M_odeSolver;
intVector_Type M_n;
UInt M_N;
UInt M_Nt;
real M_T;
vector_Type M_solution;
BFLinearCombination M_solutionEvaluator;

real findBound()const;
void localMatrix(matrix_Type &A,matrix_Type &B,matrix_Type &C,UInt interval)const;
void massStiffnessMatrices(matrix_Type &Mass,matrix_Type &Stiffness)const;
void integralMatrix(matrix_Type &Integral)const;
void externalIntegral(real t,real Bound,vector_Type &fx)const;
}

namespace SEM_NI

First we define the linear solver: we used the partial pivoting LU decomposition, built in the Eigen Library.

Then we have two constructor. The first is the user friendly constructor and it is used to solve the problem using a uniform mesh with uniform degree within each element. It takes only the problem object, the number of gauss nodes in each element, the number of element and the number of time steps and domain parameters (spatial and temporal). The second one, instead, takes the mesh object which can be personalized, allowing to use non uniform meshes with non uniform degree. It can also be specified the time discretization.

After that we have the method solve that simply solve the problem and store the solution in the variable M_solution.

The method printSolution simply print the solution and its first and second derivatives on the stream specified by OUTPUT. We can use this function in conjuction with Gnuplot to make some plots of the solution.

evaluateSolution simply return the value of the solution in the point x.

Finally we have two get method for the mesh and for the vector containing the solution coefficients.

Among the private members we have all the objects needed within the various methods. There are also some auxiliary methods used by solve:
findBound is used to find a bound for the integral in the \( f(t,S) \) term, which is evaluated using externalIntegral, in (12), because we have an unbounded interval.
massStiffnessMatrices and integralMatrix are used to build the matrix \( M,S,I \) in (16).
localMatrix is used to build the local matrix for each element.

4.2.1 solve member function

The member function solve is the function that actually solve the problem: so we decided to comment the code and the implementation.

void
SemPideSolver::solve()
{
const real Bound=findBound();
const real dt=M_T/M_Nt;
real theta;
switch (M_odeSolver) {
    case ODESolver_Type::ImplicitEuler:
        theta = 1;
        break;
    case ODESolver_Type::ExplicitEuler:
        theta = 0;
        break;
    case ODESolver_Type::CrankNicolson:
        default:
            theta = 0.5;
        }

    matrix_Type Mass = matrix_Type::Zero(M_mesh.getSize(), M_mesh.getSize());
    matrix_Type Stiffness = Mass;
    matrix_Type Integral = Mass;

    massStiffnessMatrices(Mass, Stiffness);
    integralMatrix(Integral);

    vector_Type b(M_mesh.getSize());
    vector_Type fxNew(M_mesh.getSize());
    vector_Type fxOld(M_mesh.getSize());

    for (Int i = 0; i < M_solution.size(); ++i)
        M_solution[i] = M_problem.initialCondition(M_mesh(i));

    externalIntegral(0, Bound, fxOld);
    linearSolver_Type pplu;
    pplu.compute(1/dt*Mass-theta*(Stiffness+Integral));

    for (UInt j = 1; j <= M_Nt; ++j)
    {
        externalIntegral(j*dt, Bound, fxNew);
        b = (1/dt*Mass+(1-theta)*(Stiffness+Integral))*M_solution +
            Mass*(theta*fxNew+(1-theta)*fxOld);
        fxOld = fxNew;
        b[0] = 1/dt*M_problem.leftBoundary(j*dt, M_mesh(0));
        b[b.size()-1] = 1/dt*M_problem.rightBoundary(j*dt, M_mesh(b.size()-1));
        M_solution = pplu.solve(b);
    }

    M_solutionEvaluator.setU(M_baseFun, M_solution);
    return;
}

First the function finds the bounds for the integral defined in the $f(t,S)$ term in (12) and the time step is calculated. The enum class M_odeSolver is used to decided the value of $\theta$ for Crank Nicolson. After that, we build the stiffness matrix and the integral matrix using the corresponding member functions. The initial condition is stored in M_solution and the LU decomposition is calculated. Also the first $f(t,S)$ term is calculated using externalIntegral. Then, there is a for loop that makes each time steps in the Crank Nicolson method, in which is calculated the value of $f(t,S)$ in the last time step, the constant term is evaluated and the value of the boundary corrected and finally the method solve of Eigen PartialPivLU is called to solve the linear system. The last step is to update the linear combination of base function that will be used to plot the solution.

4.3 Mesh class
The Mesh class is used to build the Gauss mesh.

namespace SEM_NI{
class Mesh
{
    public:
    Mesh(UInt n, UInt N, real xmin, real xmax);
    Mesh(real xmin, vector_Type const &dx, intVector_Type const &n, UInt N);
    Mesh(Mesh const &)=default;

    real operator()(UInt i) const {return M_nodes(i);};
    real operator()(UInt i, UInt j) const {return M_nodes(j + M_cumulativen(i));};

    UInt getN() const {return M_N;};
    const intVector_Type& getn() const {return M_n;};
    real getDx(UInt i) const {return M_dx(i);};
    UInt getSize() const {return M_nodes.size();};
    UInt getMaxn() const {return M_n.maxCoeff();};
    real getGaussNodes(UInt i, UInt j) const {return M_gaussNodes[i][j];};
    real getGaussWeights(UInt i, UInt j) const {return M_dx(i)/2.*M_gaussWeights[i][j];};

    private:
    UInt M_N;
    intVector_Type M_n;
    intVector_Type M_cumulativen;
    vector_Type M_dx;
    vector_Type M_nodes;
    std::vector<vector_Type> M_gaussNodes;
    std::vector<vector_Type> M_gaussWeights;
};
}  // namespace SEM_NI

The class has two constructor, one simple to build uniform mesh, the second instead is used to build a personalized mesh taking as arguments the elements length and a vector that contains the number of Gauss nodes in each element.

The function call operator returns a particular node in the mesh using two different numeration: the first takes simply the position in the grid: the second takes the elements in which is contained the node and the position of the node in that element. Note that with the second operator there are some nodes that can be returned using two different indices. This type of numeration is used widely in the code because is really natural, however there is some slowdown caused by the need of more nested for loops.

There are also a lot of getters to maintain a clean code.

4.4 BaseFunction class

The BaseFunctions class is used to handle the base functions associated with each element in the mesh. Taking the mesh in the constructor, it calculates the coefficients of the Lagrange polynomials on the reference elements in \([-1, 1]\) with the right degree.

namespace SEM_NI{

    class BaseFunctions
    {
        public:
        typedef Eigen::FullPivLU<Eigen::MatrixXd> linearSolver_Type;
        BaseFunctions(Mesh const &mymesh);
    }
There are 3 methods that permits to evaluate the base functions in a particular point using Horner algorithm. The idea is the same as the function call operator in the mesh class. If we take for example the member function evald, we have the point x in which we want to evaluate the function, the position i in the grid of the element and j that is the number of the base function in that element. For efficiency reason, if we evaluate a base function outside its element the result is not 0 but a wrong number. There is also a private member function, affineTransformation, that is used to map the elements in $[-1,1]$ and its used by the eval functions. Last we decided to use friendship with the class BFLinearCombination to make easier the acces to private elements of BaseFunctions class when used by BFLinearCombination class.

### 4.5 BFLinearCombination

The class BFLinearCombination is used to evaluate a particular linear combination of the base functions. When we solve the PIDE with SEM-NI we get a vector of coefficients, which are also the value of the solution in the mesh nodes, that we used to construct the solution using this class.
The constructor takes as argument the \texttt{BaseFunctions} object and the vector of the linear combination. We also have a \texttt{setter} used to reset the vector of the coefficients. The evaluation functions need the points in which we want to evaluate the linear combination of base functions, but also the element in which the point \(x\) is located. This is done only for efficiency reason, because we want to avoid the function to find the right element.

### 4.6 Integrator and GaussLegendre classes

The \texttt{Integrator} class is used to calculate the composite quadrature using a generic quadrature rule that is passed as policy as template parameter.

```cpp
namespace SEM_NI{
    template<class Method>
    class Integrator
    {
    public:
    Integrator(UInt n=0);
    template<class FUN>
    real solve(real a, real b,FUN const &integrand,UInt N)const;
    
    private:
    vector_Type M_nodes;
    vector_Type M_weights;
    
    };//namespace SEM_NI
```
private:
vector_Type M_nodes;
vector_Type M_weights;
UInt M_n;

public:
GaussLegendre(UInt n):M_nodes(n),M_weights(n),M_n(n){
    gl_generator(M_weights,M_nodes,n);
};
void generateNodes(vector_Type &nodes) const;
void generateWeights(vector_Type &weights) const;
}

}//namespace SEM_NI

4.7 Problem class

This class is used to define all the functions and value that appears in (12).

namespace SEM_NI{

class Problem
{
public:
    std::function<real(real)> initialCondition;
    std::function<real(real,real)> leftBoundary;
    std::function<real(real,real)> rightBoundary;
    std::function<real(real)> transport;
    std::function<real(real)> diffusion;
    std::function<real()> reaction;
    std::function<real(real)> levyMeasure;

    Problem()=default;
};

}//namespace SEM_NI

To achieve maximum flexibility we decide to use C++11 function wrappers to define all the functions that defines the problem. In this way we avoid to recompile the library when we want to solve another problem. For example we can define the function using lambda functions in the main. We think this is a clean way to address the problem, to avoid the need of passing function object for every function in (12) directly to the class SemPideSolver.

4.8 How to use SemPide Library

To install and use SemPide library you need gcc 4.7 compiler (it’s important that the compiler has C++11 features) and the Eigen library version 3.1. SemPide library is provided with a makefile: modify the line EIGEN in the makefile and put the path to the folder in which you put Eigen library. Then you have two options: build a dynamic library or build a static library. The command make dynamic builds the dynamic library, the command make static builds the static one. You can deactivate debugging and activate optimization with the command macro DEBUG=no.

After compiling the library, to solve a PIDE problem of the form presented above, you have to write a main source file. You have to specify the problem using the class Problem simply defining the functions using for example C++11 lambdas. Then you have two choixes: you can use the solver with uniform mesh or use the advanced one (as described above under SemPideSolver class).
Also in the case of user friendly solver you can chose the number of elements, the number of nodes in each element and the number of time steps. Call the constructor for \texttt{SemPideSolver} and then call the function \texttt{solve}. To save the value of the solution along the mesh you can use the function member \texttt{print}. A note: this library is designed to solve Financial problem where is important only the value of the function (and its derivatives) in $t = 0$, so the class \texttt{SemPideSolver} doesn’t store the value of the solution in $s \in (0, T)$. If you want, you can get the coefficients of the linear combination of base functions using the function member \texttt{getSolution}.
5 Application to standard options

In this section we show some example of application of the SemPide library.

5.1 Convergence Test for Merton Model

We start with a convergence test for the Merton model for which we have a closed form solution for the price of European option (file test1.cpp).

We chose the following parameters:

\[ r = 0.05 \quad \sigma = 0.2 \]
\[ \lambda = 0.4 \quad \mu = -0.3 \quad \delta = 0.2 \]
\[ K = 100 \quad T = 1 \]

For these parameters the price of European option is:

<table>
<thead>
<tr>
<th></th>
<th>call</th>
<th>put</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13.461376589873916</td>
<td>8.584319039945328</td>
</tr>
</tbody>
</table>

We test the convergence of the price for \( K = 100 \), varying the number of elements and the degree of each elements. We used for each test \( N_t = 250 \) temporal steps with Crank Nicolson temporal discretization. For the convergence test in the degree we used \( N = 4 \) elements, for the convergence test in the number of elements we used a degree of \( n = 3 \). From the plot we can see that we have exponential convergence in the degree of the polynomials and more than quadratic convergence in the number of elements. We can notice from the plot that the convergence saturates near degree 7. Probably, over this degree, the error associated with the discretization is dominated by other errors like domain truncation or finite precision error.

5.2 Application to Up&Out and Binary Option

We test the algorithm also with some non standard option like barrier and binary option (file test3.cpp). We note that this options have a discontinuity in the initial condition. We used the same parameters as in the convergence tests. For the Up&Out option we have a barrier in \( B = 130 \). In Figure (3) we can see a plot of the price, the delta and the gamma of the option for this option. Thanks to the high order of the polynomials we have an high resolution for the gamma.
of the option which is very important when we want to build a portfolio to hedge the position in the option. We have also tested the algorithm with a binary option that has an initial condition like an heaviside function. The result is not so exciting like for the barrier option. In fact we can notice that we have some numerical instability near the discontinuity of the initial condition that lead to an error in the gamma of the option (the second derivatives of the solution). We can see it in detail in figure (7). This kind of instability is probably given by the high order of the base functions and the use of Crank-Nicolson temporal discretization that can give some problems with discontinuities in the initial condition (see [Seyd]).

5.3 Variance Gamma model

In file `test2.cpp` we tried the SEM algorithm also with an infinite activity model: the Variance Gamma model. We used the Asmussen Rosinski approximation for the Levy measure using a truncation parameter $\epsilon = 0.5$. Under this threshold we have high instability. There is no standard for the choise of the $\epsilon$, in [ContT] is suggested that a low level is always cause of high numerical instability. We chose the following parameters

\[ r = 0.05 \quad \theta = 0.25 \quad \sigma = 0.3 \quad k = 0.2 \]
Figure 5: Particular of the instability for the second derivatives of the Put solution

Figure 6: Call option using Variance Gamma
We can see from the plot two different behaviors: for the call option that has an unlimited initial condition, the result is really poor: we can notice a bad plot for the first and second derivatives of the solution. For the put option instead, we have a good behaviour, even if we can notice a little bump in the second derivatives in the neighbor of the price $S = 100$. We have to note that the truncation of the Lévy measure can affect really badly the value of the integral matrix. In fact the Asmussen Rosinski approximation introduces a discontinuity in the measure so Gauss integration may perform badly.
Conclusions

We have developed a C++ code that uses a Spectral Element Method with numerical integration (SEM-NI) to solve a particular kind of partial integro differential equation (PIDE) arising in option pricing. We have tested the code using two different model and many option type. In the case of the Merton model the performance is very good: we have exponential convergence in the degree of the elements. Also the use of the model for pricing barrier and binary option is good. To apply the algorithm to the Variance Gamma model we needed the Asmussen Rosinski approximation of the Lévy measure that introduces a discontinuity in the measure that is the cause of numerical instability when pricing option with unlimited initial condition.

We concluded that this method is well suited for the Merton model but perform badly when used for different models with infinite activity Lévy measure, or in general with measure with low regularity.

We suggest a future development in the direction of stochastic volatility and jumps: the Bates model. The C++ code could give some guide lines for the development of a SEM-NI code in 2-D for the PIDE of the Bates model, even if, probably, a big part of the code must be rewrited.
References


