Basket Option Pricing on Sparse Grids Using Fast Fourier Transforms

Silvan Villiger

Computational Science Engineering

Master Thesis

Seminar for Applied Mathematics

ETH Zürich

Professor : Christoph Schwab

Supervisor : Christoph Winter

Draft: November 3, 2006 - May 1, 2007

Contents

1 Introduction

"The Fast Fourier Transform - the most valuable numerical algorithm of our lifetime" (G. Strang, 1993). When valuing and risk-managing derivatives, practitioners demand fast and accurate methods in order to calculate prices and sensitivities. Common quadrature methods compute one option value in $O(N)$. Another approach, initiated by Carr and Madan [2], proposes for numerically inverting the Fourier transform of the damped option price. Thus one gets the option values for a whole range of N strike prices with one single IFFT in $O(N \ln N)$ which can all be used for example in the calibration of volatility smiles. Since such a calibration process involves fitting the implied volatilities of market prices, considering just options on single stocks will do it. In this thesis, we will try to extend the idea of Carr and Madan to the multidimensional case in order to investigate the prices of options on portfolios. Together with the market prices of such options, the resulting method could also be used to calibrate the implied correlations between the stocks of the portfolio.

Carr and Madan have transformed the option value as a function of the strike price. In the multidimensional case this strike price remains one-dimensional while the density at maturity depends on all the underlyings. Therefore, an analogue transformation would not yield a closed form expression with the characteristic function of the stock values. Moreover, the strike price is not defined for all options. It is for these reasons that we transform the option value as a function of the initial stock prices as in Lewis [15] and thereby obtain the option prices for a whole grid of such initial values.

We discuss the extension of Lewis' method on multidimensional full grids and the necessary modifications in chapter 3. Then we try to investigate a generic treatment of the method on sparse grids in chapter 4. After presenting a documentation of our program in chapter 5 we demonstrate the results of our method and argue its drawbacks and advantages. We finish the thesis by making some suggestions for future investigations in chapter 7.

2 Modeling and Pricing Background

The stock prices are modeled by a stochastic process $\mathbf{S}_t$ whose evolution is governed by the underlying probability measure $\mathcal{P}$. The Fundamental Theorem of Asset Pricing exhibits that a price model is arbitrage free if and only if there exists an equivalent risk-neutral measure, that is a probability measure $\mathcal{Q} \sim \mathcal{P}$ such that $\exp\{-\int_0^t r(s)\,ds\}\, S_t$ is a $\mathcal{Q}$-martingale. Assuming that the riskless rate r is constant and that the measure $\mathcal{Q}$ is already chosen, then the option price at time 0 can be calculated by

$$V(\mathbf{S}_0) = e^{-rT}\, \mathbb{E}^{\mathcal{Q}}\, [f(\mathbf{S}_T)] \tag{2.1}$$

where $f(S_T) = V(S_T, T)$ denotes the payoff of the option. Furthermore, we assume the underlying prices at maturity T to be modeled by

$$\mathbf{S}_T = \exp\{rT + \mathbf{s} + \mathbf{X}_T\}, \tag{2.2}$$

where $\mathbf{s}$ is given by $\mathbf{s} = \ln \mathbf{S}_0$ and $\mathbf{S}_0$ are the initial stock prices. The random vector $\mathbf{X}_T$ describes the log-returns on forward contracts to buy stocks at time T. Notice that in many important stock price models we know the characteristic function

$$\Phi_T(\mathbf{z}) = \mathbb{E}\, \left[e^{i\,\mathbf{z}^T \mathbf{X}_T}\right] \tag{2.3}$$

but not the density $\rho_T(\mathbf{x})$ of $\mathbf{X}_T$ in closed form.

2.1 Stock Price Models

2.1.1 Black Scholes Model

Black and Scholes [1] modeled the underlying price processes $(\mathbf{S}_t)$ by a geometric brownian motion

$$\frac{d\mathbf{S}_t}{\mathbf{S}_t} = r\,dt + A\,d\mathbf{W}_t \tag{2.4}$$

with constant interest rate r, constant volatility matrix $\Sigma = AA^T$ and independent browniant motions $\mathbf{W}_t$.

The random vector $\mathbf{X}_T$ is then normally distributed

$$\mathbf{X}_T^{\mathrm{BS}} = A\,\mathbf{W}_T - \frac{1}{2}\,\mathrm{diag}(\Sigma)\,T \quad \sim \quad \mathcal{N}_d(-\frac{1}{2}\,\mathrm{diag}(\Sigma)\,T,\ \Sigma\,T)$$

and the characteristic function reads

$$\Phi_T(\mathbf{z}) = \exp\left\{-\frac{1}{2}\,T\ \left(i\,\mathrm{diag}(\Sigma)^T\mathbf{z} + \mathbf{z}^T\Sigma\ \mathbf{z}\right)\right\}. \tag{2.5}$$

2.1.2 Lévy Models

Empirically observed log returns of risky assets are not normally distributed but exhibit significant skewness and kurtosis and they occasionally appear to change discontinuously. Unfortunately, these empirical facts are not covered by the Black Scholes model. However, we can include jumps by modelling the underlyings with a Lévy process. A càdlàg stochastic process $\{X_t :\ t>0\}$ on $\mathbb{R}^d$ such that $X_0 = 0$ a.s. is called a Lévy process if it has independent and stationary increments and is stochastically continuous.

The characteristic function $\Phi(\mathbf{z}) = e^{-t\,\Psi(\mathbf{z})}$ of a Lévy process can be obtained with the Lévy-Khinchin representation [20]

$$\Psi(\mathbf{z}) = i\,\gamma^T\mathbf{z} - \frac{1}{2}\mathbf{z}^T\Sigma\,\mathbf{z} + \int_{\mathbb{R}^d}\left(e^{i\,\mathbf{z}^T\mathbf{x}} - 1 - i\,\mathbf{z}^T\mathbf{x}\,\mathbb{I}_{\{|\mathbf{x}|\le 1\}}\right)\,\nu(d\mathbf{x})$$

where $\Sigma \in \mathbb{R}^{d\times d}$ is a covariance matrix, $\gamma \in \mathbb{R}^d$ a drift vector and ν the Lévy measure which satisfies

$$\int_{|\mathbf{x}|<1}|\mathbf{x}|^2\,\nu(d\mathbf{x}) < \infty, \qquad \int_{|\mathbf{x}|>1}\nu(d\mathbf{x}) < \infty$$

The triplet (Σ, ν, γ) is called characteristic triplet of the stochastic process $\mathbf{X}_t$ and $-\Psi(\mathbf{z})$ is known as Lévy symbol.

Multidimensional Lévy densities can be obtained from one-dimensional Lévy densities using Lévy copulas [21, 22]. However, they can be calculated explicitly for special cases. Let, for example, $k_1, \ldots, k_d$ be one-dimensional Lévy densities. Then the jumps of a Lévy process with Lévy density

$$k(x_1, \ldots, x_d) = k_1(x_1)\, \delta_{\{x_2=\cdots=x_d=0\}} + \ldots + k_d(x_d)\, \delta_{\{x_1=\cdots=x_{d-1}=0\}}$$

are independent and the marginal Lévy densities are given by $k_1, \ldots, k_d$. In case of $\Sigma = 0$ and $\gamma = 0$ we can calculate the corresponding multidimensional Lévy symbol simply by

$$\Psi(\mathbf{z}) = \sum_{i=1}^{d} \Psi_i(z_i).$$

Assuming $k_1 = \cdots = k_d$, the jumps of a Lévy process with Lévy density

$$k(x_1, \ldots, x_d) = k_1(x_1)\, \delta_{\{x_1=\cdots=x_d\}}$$

are complete dependent and the marginal Lévy densities are given by k_1. The Lévy symbol for $\Sigma = 0$ and $\gamma = 0$ can then be computed by

$$\Psi(\mathbf{z}) = \Psi_1 \left(\sum_{i=1}^{d} z_i \right).$$

An common one-dimensional example of Lévy processes in option pricing is the Variance Gamma process [23]

$$X_t = \theta\, \gamma_t(\nu) + \sigma\, W_{\gamma_t(\nu)}.$$

The Variance Gamma model is obtained by evaluating the Brownian motion with drift at a random time $\gamma_t(\nu)$ which is modelled by a gamma process. The characteristic exponent of the process is given by

$$\Psi(z) = \frac{1}{\nu}\, \log(1 - i\, z\, \nu\, \theta + \frac{1}{2}\, \sigma^2 \nu\, z^2) - i\, z\, \gamma$$

where the drift parameter γ can be determined by $\Phi(-i) = 1$ because of the martingale condition $\mathbb{E}\,[\, \exp\{\, X_t \,\} \mid X_0 = 0\,] = 1$.

2.2 Multidimensional Options

2.2.1 Basket Call Option

A call option on a portfolio with component weights $\mathbf{c}$ offers its holder the right but not the obligation to buy the portfolio at strike price $K = e^k$. Hence, it has the payoff

$$f(\mathbf{S}_T) = (\mathbf{c}^T\mathbf{S}_T - K)^+ = \max\left\{\mathbf{c}^T\mathbf{S}_T - K,\, 0\right\} \tag{2.6}$$

at maturity T. Basket options are usually applied in trading two or more foreign currencies. According to the portfolio theory, the volatility of the basket is generally smaller than that of each individual currency. Therefore, the basket option premium is less than the sum of the premiums of all individual options on each underlying currency.

2.2.2 Maximum Call Option

A call option on the maximum of d stock prices offers its holder the right to buy the best of those assets at strike price K and its payoff is

$$f(\mathbf{S}_T) = (\max\{S_1, \ldots, S_d\} - K)^+. \tag{2.7}$$

Contracts of this type are often used in insurance when we expect at most one extreme event, because insuring just the most expensive one is cheaper than insuring all events.

3 Multidimensional Pricing Using FFT

The d-dimensional Fourier transform is defined by

$$\mathcal{F}\left\{g(\mathbf{x})\right\}(\mathbf{z}) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{i\,\mathbf{z}^T\mathbf{x}}\, g(\mathbf{x})\, d\mathbf{x}, \tag{3.1}$$

where $\mathbf{z} \in \mathbb{R}^d$. The function $g(\mathbf{x})$ is called Fourier integrable if $\mathcal{F}\left\{g(\mathbf{x})\right\}(\mathbf{z})$ exists and is analytic for all $\mathbf{z} \in \mathbb{R}^d$. The inverse Fourier transform can consequently be calculated by

$$g(\mathbf{x}) = \frac{1}{(2\pi)^d} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-i\,\mathbf{x}^T\mathbf{z}}\, \mathcal{F}\left\{g(\mathbf{x})\right\}(\mathbf{z})\, d\mathbf{z}. \tag{3.2}$$

3.1 Extension of Lewis' Method

Contrary to the method of Carr and Madan [2] the approach of Lewis [15] can be used to price contracts with arbitrary payoff structures. For the reasons mentioned in the introduction the option value

$$V(\mathbf{s}) = e^{-rT} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(e^{\mathbf{s}+\mathbf{x}+rT})\, \rho_T(\mathbf{x})\, d\mathbf{x}$$

is considered to be a function of the initial log stock prices. Its Fourier transform can be calculated by

$$\begin{aligned}
\mathcal{F}\left\{V(\mathbf{s})\right\}(\mathbf{z}) &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{i\,\mathbf{z}^T\mathbf{s}}\, V(\mathbf{s})\, d\mathbf{s} \\
&= e^{-rT} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \rho_T(\mathbf{x}) \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{i\,\mathbf{z}^T\mathbf{s}}\, f(e^{\mathbf{s}+\mathbf{x}+rT})\, d\mathbf{s}\, d\mathbf{x} \\
&= e^{-rT} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-i\,\mathbf{z}^T\mathbf{x}} \rho_T(\mathbf{x})\, d\mathbf{x} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{i\,\mathbf{z}^T\mathbf{y}}\, f(e^{\mathbf{y}+rT})\, d\mathbf{y}
\end{aligned}$$

which yields the following simple expression

$$\mathcal{F}\left\{V\right\}(\mathbf{z}) = e^{-rT}\, \Phi_T(-\mathbf{z})\, \mathcal{F}\left\{f^*\right\}(\mathbf{z}) \tag{3.3}$$

where $f^*(\mathbf{y}) = f(e^{\mathbf{y}+rT})$.

Unfortunately, the payoff usually grows to infinity and therefore its Fourier transform $\mathcal{F}\{f^*\}(\mathbf{z})$ does not exist. We will address this problem in chapter 3.2 and assume for now that the payoff actually is Fourier integrable.

Whenever we know easy evaluable closed form expressions for $\Phi_T(-\mathbf{z})$ and $\mathcal{F}\{f^*\}(\mathbf{z})$, we can quickly get an approximation for the option price by discretizing the inverse transform

$$V(\mathbf{s}) = \frac{e^{-rT}}{(2\pi)^d} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-i\,\mathbf{s}^T\mathbf{z}}\, \Phi_T(-\mathbf{z})\, \mathcal{F}\{f^*\}(\mathbf{z})\, d\mathbf{z} \tag{3.4}$$

of (3.3) using a quadrature rule. In order to be able to evaluate the occurring sum with a FFT algorithm we need an equidistant grid with a number of points which is factorable into small factors. This requirements lead us to Newton-Côtes formulas. For the use of sparse grids in chapter 4.1 we assume furthermore that the grids are nested. We therefore restrict ourselves to semi-open Newton-Côtes formulas [26] because nested closed and open rules have $2^n \pm 1$ points which is not factorable into small primes.

Since for most models discussed in [6] the characteristic function decays faster than every power of its argument at infinity, we may approximate $V(\mathbf{s})$ by cutting off the integrand at some finite values $\mathbf{z}_{\min}$ and $\mathbf{z}_{\max}$

$$V(\mathbf{s}) \approx \frac{e^{-rT}}{(2\pi)^d} \int_{z^{(1)}_{n_1,\min}}^{z^{(1)}_{n_1,\max}} \cdots \int_{z^{(d)}_{n_d,\min}}^{z^{(d)}_{n_d,\max}} e^{-i\,\mathbf{s}^T\mathbf{z}}\, \Phi_T(-\mathbf{z})\, \mathcal{F}\{f^*\}(\mathbf{z})\, d\mathbf{z}.$$

The quadrature rule for (3.4) then reads

$$V(\mathbf{s}) \approx \frac{e^{-rT}}{(2\pi)^d} \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} \omega_{\mathbf{n},\mathbf{k}}\, \Phi_T(-\mathbf{z}_{\mathbf{n},\mathbf{k}})\, \mathcal{F}\{f^*\}(\mathbf{z}_{\mathbf{n},\mathbf{k}})\, e^{-i\,\mathbf{s}^T\mathbf{z}_{\mathbf{n},\mathbf{k}}} \tag{3.5}$$

$$\omega_{\mathbf{n},\mathbf{k}} = \omega^{(1)}_{n_1,k_1} \cdots \omega^{(d)}_{n_d,k_d}, \qquad \mathbf{z}_{\mathbf{n},\mathbf{k}} = (z^{(1)}_{n_1,k_1}, \ldots, z^{(d)}_{n_d,k_d})^T$$

where $\omega^{(l)}_{n_l,k_l}$ is the one-dimensional weight of the quadrature rule, $N_l = 2^{n_l}$ is the number of nodes for quadrature level n_l and the nodes are given by

$$\begin{aligned} z^{(l)}_{n_l,\min} &= z^{(l)}_{\text{cent}} - \tfrac{N_l}{2}\,\Delta z^{(l)}, & z^{(l)}_{n_l,k_l} &= z^{(l)}_{n_l,\min} + k_l\,\Delta z^{(l)}, \\ z^{(l)}_{n_l,\max} &= z^{(l)}_{n_l,\min} + N_l\,\Delta z^{(l)}, & z^{(l)}_{0,0} &= z^{(l)}_{\text{cent}}. \end{aligned} \tag{3.6}$$

The step size in dimension $l = 1, \ldots, d$ is denoted by $\Delta z^{(l)}$ and the grid center $\mathbf{z}_{\text{cent}}$ is a parameter vector which is usually set to 0. An illustration of the grid can be found in figure 1.

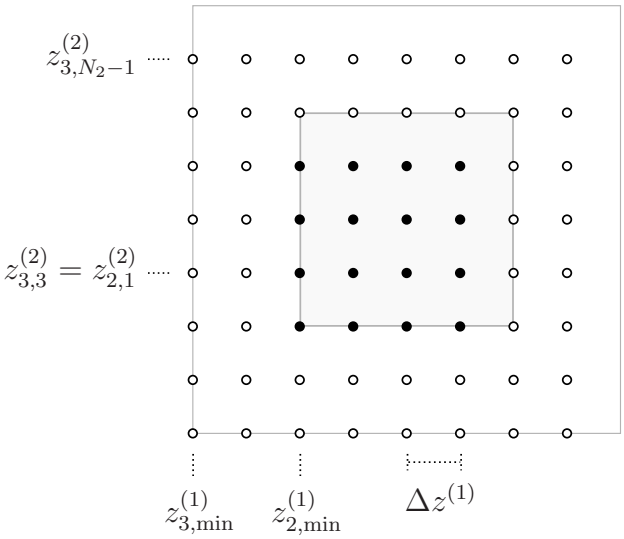

Figure 1: Full grid expansion in z space illustrated with $n = 2, 3$.

Notice that if the integrand is analytic in a stripe around the real axis even a simple rectangular rule will yield exponential convergence and using more sophisticated rules may not be worth the effort (see [13]). We will therefore stick with the rectangular rule with the weights

$$\omega^{(l)}_{n_l,k_l} = \Delta z^{(l)} = \frac{z^{(l)}_{n_l,max} - z^{(l)}_{n_l,min}}{N_l}. \tag{3.7}$$

for now. Instead of only evaluating the sum (3.5) in $O(N_1 \cdots N_d)$ we could get price approximations for a whole equidistant grid

$$\begin{aligned} s^{(l)}_{\min} &= s^{(l)}_{\text{cent}} - s^{(l)}_{\text{range}}, \qquad & s^{(l)}_{n_l,j_l} &= s^{(l)}_{\min} + j_l\, \Delta s^{(l)}_{n_l}, \qquad & s^{(l)}_{0,0} &= s^{(l)}_{\text{cent}} \\ s^{(l)}_{\max} &= s^{(l)}_{\text{cent}} + s^{(l)}_{\text{range}}, \qquad & \Delta s^{(l)}_{n_l} &= (s^{(l)}_{\max} - s^{(l)}_{\min})/N_l. \end{aligned} \tag{3.8}$$

of initial stock prices $\mathbf{s_{n,j}} = (s^{(1)}_{n_1,j_1}, \ldots, s^{(d)}_{n_d,j_d})^T$ in $O(N_1 \cdots N_d \ \ln[N_1 \cdots N_d])$ by applying a d-dimensional FFT algorithm [27].

The parameter vectors $\mathbf{s}_{\text{cent}}$ and $\mathbf{s}_{\text{range}}$ can be used to define the location and the size of the grid which is pictured in figure 2.

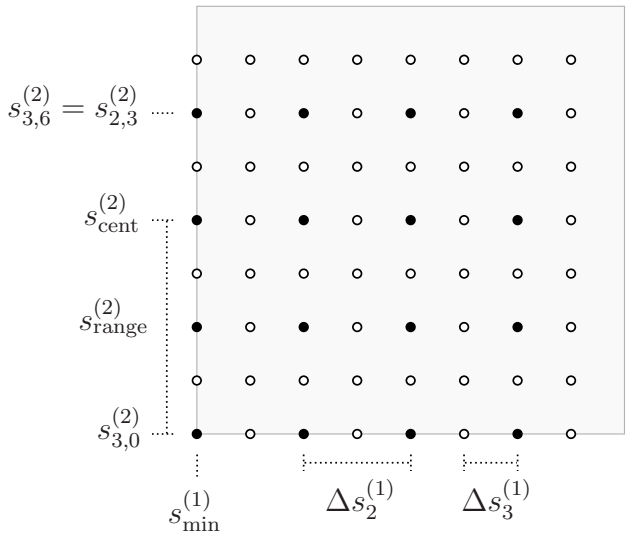

Figure 2: Full grid expansion in s space illustrated with $n = 2, 3$.

A d-dimensional FFT algorithm calculates

$$\text{FFTD}\left\{g_{\mathbf{n},\mathbf{k}}\right\} = \sum_{k_1=0}^{N_1-1} \cdots \sum_{k_d=0}^{N_d-1} g_{\mathbf{n},\mathbf{k}}\, e^{-i\,2\pi\,\sum_{l=1}^{d} \frac{j_l k_l}{N_l}} \tag{3.9a}$$

$$\text{or} \qquad \text{IFFTD}\left\{h_{\mathbf{n},\mathbf{j}}\right\} = \frac{1}{N^d} \sum_{j_1=0}^{N_1-1} \cdots \sum_{j_d=0}^{N_d-1} h_{\mathbf{n},\mathbf{j}}\, e^{i\,2\pi\,\sum_{l=1}^{d} \frac{j_l k_l}{N_l}}. \tag{3.9b}$$

In order to get a sum of this form, we are forced to impose the following condition on the grid spacings

$$\Delta s^{(l)}_{n_l}\, \Delta z^{(l)} = \frac{2\pi}{N_l}. \tag{3.10}$$

Now the quadrature rule (3.5) can be rewritten in terms of a FFT

$$V(\mathbf{s}_{\mathbf{n},\mathbf{j}}) \approx \frac{e^{-rT}}{(2\pi)^d}\, e^{-i\,\mathbf{s}^T_{\mathbf{n},\mathbf{j}}\,\mathbf{z}_{\mathbf{n},\min}}\ \text{FFTD}\left\{g_{\mathbf{n},\mathbf{k}}\right\} \tag{3.11a}$$

$$g_{\mathbf{n},\mathbf{k}} = \omega_{\mathbf{n},\mathbf{k}}\, \Phi_T(-\mathbf{z}_{\mathbf{n},\mathbf{k}})\, \mathcal{F}\left\{f^*\right\}(\mathbf{z}_{\mathbf{n},\mathbf{k}})\, e^{-i\,\sum_{l=1}^{d} s^{(l)}_{\min}\, k_l\, \Delta z^{(l)}}. \tag{3.11b}$$

3.2 Fourier Integrability of the Payoff

As mentioned in chapter 3.1, the payoff usually grows to infinity and for this reason its Fourier transform

$$\mathcal{F}\{f^*\}(\mathbf{z}) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{i\,\mathbf{z}^T\mathbf{y}}\, f(e^{\mathbf{y}+rT})\, d\mathbf{y} \tag{3.12}$$

does not exist. Lewis used a generalized Fourier transform in [15] in order to deal with this problem. He assumed the one-dimensional version of (3.12) to only exist for $\tilde{z} \in \mathbb{C}$ with $\mathrm{Im}(\tilde{z}) \in (a, b)$ and inverted the transform by integrating along a straight line parallel to the real axis $\tilde{z} = i\omega + z$ for $\omega \in (a, b)$. By this, he implicitly allowed to introduce a factor $e^{-\omega\, y}$ in the integrand of (3.12). This factor could be used to dampen the growth because typical one-dimensional payoff structures only grow in one direction. However, this is no longer possible in the multidimensional case. Consider the two-dimensional basket call option (2.6) with $c_1, c_2 > 0$ as an illustration: if $c_1 e^{y_1+rT} = e^k + \delta$ with $\delta > 0$, then the integrand in (3.12) is greater than δ for all y_2 and, in consequence, damping in one direction by $e^{-\omega_2 y_2}$ is not enough for the integral over y_2 to exist.

We will now discuss two alternative ideas addressing the problem that the Fourier transform of the payoff does not exist.

3.2.1 Gaussian Damping of the Payoff

In order to dampen the payoff in both directions we could introduce $e^{-\alpha_l y_l^2}$ factors by transforming the damped option price $\exp\{-\mathbf{s}^T \,\mathrm{diag}(\alpha)\, \mathbf{s}\}\, V(\mathbf{s})$ instead of just $V(\mathbf{s})$. Unfortunately, this is no longer a convolution and therefore the substitution $\mathbf{s} = \mathbf{y} - \mathbf{x}$ in the derivation of (3.3) will not lead to a separation of the integrals over $\mathbf{x}$ and those over $\mathbf{y}$ anymore. Even if the transformation exists, its analytic calculation gets complicated if not impossible and has to be done again for every combination of contract and model type.

We could obtain a separation into contract and model by abandon the use of FFTs in the inverse transform and applying Parseval's theorem

$$V(\mathbf{s}) = e^{-rT} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathcal{F}\left\{e^{-\mathbf{x}^T\,\mathrm{diag}(\alpha)\,\mathbf{x}}\, f(e^{\mathbf{s}+\mathbf{x}+rT})\right\}(\mathbf{z}) \cdot \mathcal{F}\left\{e^{+\mathbf{x}^T\,\mathrm{diag}(\alpha)\,\mathbf{x}}\, \rho_T(\mathbf{x})\right\}(\mathbf{z})\; d\mathbf{z}$$

in order to "borrow" the factors $e^{-\alpha_l x_l^2}$ from the density similar to the idea of Papapantoleon [5]. However, we will see in chapter 3.3.1 that the analytic transformation of multidimensional payoffs is very difficult even if the integrals exist. This approach is therefore only attractive if the Fourier transform of the damped payoff exists, is cheap to evaluate and the advantages of the FFT are not desired.

3.2.2 Modify the Payoff

Another idea to get existing Fourier transforms would be to simply cut the payoff at some finite values $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$

$$\tilde{f}(\mathbf{y}) = f^*(\mathbf{y})\, \mathbb{I}_{\{\mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max}\}} = f(e^{\mathbf{y}+rT})\, \mathbb{I}_{\{\mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max}\}}. \tag{3.13}$$

Probabilistic calculus shows that the error for option values belonging to underlying prices around $\mathbf{s}_{\text{cent}}$ decays exponentially in $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$ if the density of the associated model decays fast enough.

Bear in mind that the oscillation frequency of $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ goes up for large $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$. However, this should not cause any problems for meaningful parameter choices since the relation (3.10) directly implies a finer grid in the Fourier space $\mathbf{z}$ for larger $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$ as we will see in chapter 3.3.2.

3.3 Fourier Transform of the Cut Payoff

Due to the problems of the damping approaches mentioned above, we will focus on cutting off the payoff. In this chapter we will discuss two methods dealing with the Fourier transform $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ of the cut payoff.

3.3.1 Analytic Integration

Calculating $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ analytically would yield a closed form expression for the Fourier transform of the option price (3.3), which can be approximatively inverted by (3.11). Due to the smoothness and the fast decay of the transformed option price, even a simple rectangular rule will yield up to exponential convergence as investigated in [13].

Unfortunately, the analytic calculation of $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ is usually very complicated or even impossible. In order to remove the max in the transformation of the basket call option (2.6) for example, we would have to split the integral into many d-dimensional integrals whose inner integration bounds depend on the outer integration variables.

3.3.2 Double Discretization

Instead of calculating $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ by hand, we could also approximate it with a quadrature rule. As we need the transformation of the cut payoff on a whole grid of $\mathbf{z}_{\mathbf{n},\mathbf{k}}$ in the inverse transform (3.11b), we can save huge effort by using the FFT again which is the reason for us to call this method double discretization method.

Because of the substitution $\mathbf{s} + \mathbf{x} = \mathbf{y}$ in the derivation of (3.3) and the fact that $\mathbf{X}_T$ is just used to introduce variance into the stock prices, we choose the grid $\mathbf{y}_{\mathbf{n},\mathbf{j}}$ to be identical to the grid of initial stock prices $\mathbf{s}_{\mathbf{n},\mathbf{j}}$ in (3.8). If we choose $\mathbf{y}_{\min} \geq \mathbf{s}_{\min}$ and $\mathbf{y}_{\max} \leq \mathbf{s}_{\max}$, then the integrals can be chopped off at $\mathbf{s}_{\min}$ and $\mathbf{s}_{\max}$ without loss of accuracy and the Fourier transform of the cut payoff can be written as

$$\mathcal{F}\{\tilde{f}\}(\mathbf{z}) = \int_{s_{\min}^{(1)}}^{s_{\max}^{(1)}} \cdots \int_{s_{\min}^{(d)}}^{s_{\max}^{(d)}} e^{i\,\mathbf{z}^T\mathbf{y}}\, f(e^{\mathbf{y}+rT})\, \mathbb{I}_{\{\mathbf{y}_{\min} \leq \mathbf{y} \leq \mathbf{y}_{\max}\}}\, d\mathbf{y}.$$

We can approximate these integrations for the whole grid $\mathbf{z_{n,k}}$ by applying a rectangular rule with the weights

$$\omega_{n_l,j_l}^{(l)} = \Delta s_{n_l}^{(l)} = \frac{s_{max}^{(l)} - s_{min}^{(l)}}{N_l} \tag{3.14}$$

and evaluating the resulting sum with an IFFT algorithm (3.9b)

$$\mathcal{F}\left\{\tilde{f}\right\}(\mathbf{z_{n,k}}) \approx e^{i\,\mathbf{z}_{\mathbf{n},\mathbf{k}}^T\,\mathbf{s}_{\min}}\; N^d\;\mathrm{IFFTD}\left\{h_{\mathbf{n},\mathbf{j}}\right\} \tag{3.15a}$$

$$h_{\mathbf{n},\mathbf{j}} = \omega_{\mathbf{n},\mathbf{j}}\,\tilde{f}(\mathbf{s_{n,j}})\,e^{i\,\sum_{l=1}^{d} z_{n_l,\min}^{(l)}\,j_l\,\Delta s_{n_l}^{(l)}}. \tag{3.15b}$$

The price approximations on a grid of underlying prices can therefore be optained with just two d-dimensional FFTs by putting (3.15) into (3.11).

Bear in mind that the condition (3.10) restricts our freedom in the choice of the quadrature step sizes $\Delta\mathbf{s_n}$ and $\Delta\mathbf{z}$. Since the payoff is usually not continuously differentiable, the approximation (3.15) converges with first order in $\Delta\mathbf{s_n}$ where the discretization (3.11) shows up to exponential convergence in $\Delta\mathbf{z}$. Therefore, we generally choose rather small step sizes in $\mathbf{s}$ space which coincides with the desire of option price approximations on a fine grid. Please note that the differences between the asymptotic speeds of convergence flatten out in higher dimensions because of arising pre-asymptotic effects.

We can now discuss the question why we decided to refine the grid only in $\mathbf{s}$ space and to expand it in $\mathbf{z}$ space for increasing quadrature levels $\mathbf{n}$. As we will see in chapter 4.1, converging one-dimensional quadrature rules in both spaces are needed for the sparse grid quadrature to converge, but the condition (3.10) inhibits halving the step sizes in both spaces simultaneously for an increase of the level by one. Therefore, the grid has to be expanded in one or both spaces. Expanding the grid in $\mathbf{s}$ space, however, is unnatural since the corresponding rules will not converge as long as $\mathbf{y}_{\min} \leq \mathbf{s}_{\min}$ or $\mathbf{y}_{\max} \geq \mathbf{s}_{\max}$ and the expansion will not yield anything afterwards. Hence we get a converging quadrature rule in $\mathbf{s}$ space by refining its grid and we choose a constant spacing in $\mathbf{z}$ space

$$\Delta z^{(l)} = 2\pi \;/\; (s_{\max}^{(l)} - s_{\min}^{(l)}) \tag{3.16}$$

which is fine enough to resolve the oscillations of (3.3). The thereby implied grid expansion leads to convergence in $\mathbf{z}$ space because of the fast decay of the characteristic function.

4 FFT Pricing on Sparse Grids

Traditional tensor product quadrature rules as the multidimensional rectangular rule presented in chapter 3 are affected by the curse of dimensionality, that is, their computational cost $O(N^d)$ grows exponentially with the dimension d. In this chapter we will introduce the Smolyak construction [9] of quadrature rules on sparse grids in order to vanquish this curse.

4.1 Sparse Grid Quadrature

The basic idea is to write the one-dimensional quadrature rule $\mathcal{Q}_n^1$ as a telesopic sum of converging rules with lower level m

$$\mathcal{Q}_n^1 = \sum_{m=1}^{n} \Delta_m, \qquad \Delta_m = \mathcal{Q}_m^1 - \mathcal{Q}_{m-1}^1, \qquad \mathcal{Q}_{-1}^1 f = 0. \tag{4.1}$$

In order to get practical methods, we need the calculation via telescopic sums to require the same number of function evaluations as the direct computation. We therefore assume from now on the one-dimensional quadrature rules to be nested, that is, all nodes of $\mathcal{Q}_{m-1}^1$ can also be found in $\mathcal{Q}_m^1$. Consequently, the approximation of a d-dimensional integral

$$\mathcal{I}^d = \int_\Omega f(\mathbf{x})\, d\mathbf{x}, \qquad \Omega \subset \mathbb{R}^d$$

with the *full tensor product quadrature rule* reads

$$\mathcal{Q}_{\mathbf{n}}^d = \sum_{\mathbf{j} \in \mathcal{I}_{\mathbf{n}}} \omega_{\mathbf{n},\mathbf{j}}\, f(\mathbf{x}_{\mathbf{n},\mathbf{j}}) = \sum_{\mathbf{m} \le \mathbf{n}} \Delta_{\mathbf{m}} \tag{4.2}$$

where the *excess formulas* are defined by

$$\Delta_{\mathbf{m}} = \Delta_{m_1}^{(1)} \otimes \ldots \otimes \Delta_{m_d}^{(d)} = \sum_{\mathbf{j} \in \mathcal{I}_{\mathbf{m}}} \hat{\omega}_{\mathbf{m},\mathbf{j}}\, f(\mathbf{x}_{\mathbf{m},\mathbf{j}}) \tag{4.3}$$

The *excess weights* can be calculated with

$$\hat{\omega}_{\mathbf{m},\mathbf{j}} = \hat{\omega}_{m_1,j_1}^{(1)} \cdots \hat{\omega}_{m_d,j_d}^{(d)}. \tag{4.4}$$

$$\hat{\omega}_{m_l,j_l}^{(l)} = \begin{cases} \omega_{m_l,j_l}^{(l)} - \omega_{(m_l-1),s}^{(l)} & \text{if } \exists\ s: \quad x_{m_l,j_l}^{(l)} = x_{(m_l-1),s}^{(l)} \\ \omega_{m_l,j_l}^{(l)} & \text{else} \end{cases}$$

An excess formula $\Delta_{\mathbf{m}}$ consists of $\prod N(m_l)$ nodes and therefore gets very large for large $|\mathbf{m}|$, respectively, for large levels in many dimensions.

The idea of *sparse grid quadrature rules* is to simply drop expensive difference formulas with large $|\mathbf{m}|$ by summing just over $\mathbf{m} \in \mathcal{G}(\mathbf{n})$

$$\mathcal{Q}^d_{\mathbf{n},\alpha} = \sum_{\mathbf{m} \in \mathcal{G}(\mathbf{n})} \Delta_{\mathbf{m}} \tag{4.5}$$

where $\mathcal{G}(\mathbf{n})$ is an admissible index set

$$\forall\ \mathbf{m},\ l = 1, .., d: \quad \mathbf{m} \in \mathcal{G}(\mathbf{n}), \quad m_l \geq 1 \quad \rightarrow \quad \mathbf{m} - \mathbf{e}_l \in \mathcal{G}(\mathbf{n}). \tag{4.6}$$

A common choice for $\mathcal{G}(\mathbf{n})$ is

$$\mathcal{G}(\mathbf{n}) = \{\mathbf{m} :\ |\tilde{\mathbf{m}}|_\alpha \leq 1\}, \qquad |\tilde{\mathbf{m}}|_\alpha = \left(\sum_{l=1}^{d} \tilde{m}_l^\alpha \right)^{\frac{1}{\alpha}}, \qquad \tilde{m}_l = \frac{m_l}{n_l} \tag{4.7}$$

which leads to the so called *generalized sparse grids*. Normal sparse grids are obtained with $\alpha = 1$ and $\alpha = \infty$ leads to full tensor product grids. We could even get super sparse grids by choosing $\alpha \in (0, 1)$ which are of interest in integrating very smooth functions. For further details we refer to the work of Gerstner and Griebel [7].

4.1.1 Notation and Sparse Grid Construction

The restriction to nested quadrature rules and the assumptions from the FFT, namely, that the grid is equidistant and that the number of grid points N in level $\mathbf{m}$

$$N(m_l) = 2^{m_l}, \qquad l = 1, \ldots, d. \tag{4.8}$$

is factorable into small factors, were the reason to choose semi-open Newton-Côtes formulas in chapter 3. We have used a rectangular rule to approximate the Fourier transform of the cut payoff (3.15). Its grid consisted of nodes $\mathbf{s}_{\mathbf{m},\mathbf{j}}$ with indices $\mathbf{j}$ and got refined for increasing levels $\mathbf{m}$. Another rectangular rule was applied in the approximative inversion of the transformed option prices (3.11). The condition (3.10) implied an expansion of this second grid with indices $\mathbf{k}$ instead of a refinement. We are therefore confronted with two different types of grids distinguished by the notation $\mathcal{J}$ and $\mathcal{K}$ whose construction has to be discussed.

First, we denote the set of all indices belonging to a full tensor product grid of level $\mathbf{m}$ by *index set*

$$\mathcal{I}_{\mathbf{m}} = \{\ \mathbf{u} = (u_1, \ldots, u_d)\ \mid\ \forall\, l:\ u_l = 0, \ldots, N(m_l)-1\ \} \tag{4.9}$$

and the part of them which are new to this level by *excess indices*

$$\mathcal{I}'_{\mathbf{m}} = \{\ \mathbf{u}\ \mid\ \forall\, l:\ u_l \in \mathcal{I}'_{m_l}\ \}, \quad \mathcal{I}'_m = \mathcal{I}_m \backslash \mathcal{I}_{m-1}, \quad \mathcal{I}'_0 = \{0\}. \tag{4.10}$$

Thereafter, we consider just the *excess grid* consisting of the *excess nodes*, that is, the nodes which occur the first time in the full grid of level $\mathbf{m}$. Their so-called *hierarchical indices* $\mathbf{q}$ are collected in the *excess sets*

$$\mathcal{D}_{\mathbf{m}} = \{\ \mathbf{q} \in \mathcal{I}_{\mathbf{m}}\ \mid\ \forall\, l:\ \ q_l = \rho_{m_l}(u_l),\quad u_l \in \mathcal{I}'_{m_l}\ \} \tag{4.11}$$

where the index mapping ρ depends on the grid type

$$\rho_m^{\mathcal{J}}(u) = \begin{cases} 0 & \text{if } m \le 1 \\ 2u - N(m) + 1 & \text{else} \end{cases} \tag{4.12a}$$

$$\rho_m^{\mathcal{K}}(u) = \begin{cases} 0 & \text{if } m \le 1 \\ u - \frac{N(m)}{2} & \text{if } m \ge 2,\ u < \frac{3\,N(m)}{4} \\ u & \text{else} \end{cases} \tag{4.12b}$$

The sparse grid constructions from excess grids with levels $\mathbf{m} \in \mathcal{G}(\mathbf{n})$ is illustrated in figures 3 and 4 where filled dots represent excess nodes.

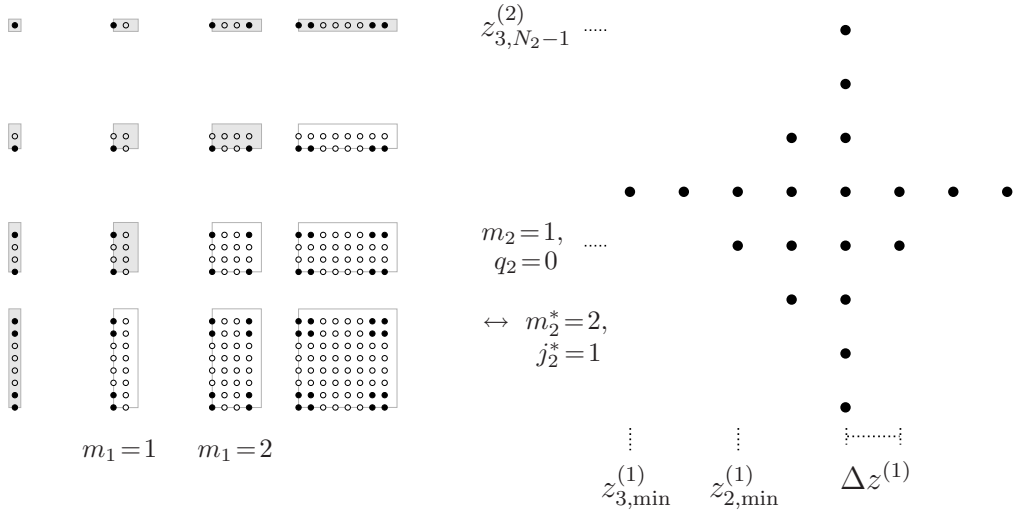

Figure 3: Sparse grid construction in z space illustrated with $n = 3$.

Due to the nestedness of the grids, the newly introduced nodes $\mathbf{x}_{\mathbf{m},\mathbf{q}}$ will show up again in every excess formula $\Delta_{\mathbf{m}^*}$ with level $\mathbf{m}^* \geq \mathbf{m}$. Their indices $\mathbf{j}^*$ resp. $\mathbf{k}^*$ in those finer grids can be calculated componentwise with

$$j^* = \sigma^{\mathcal{J}}_{m,m*}(q) = \begin{cases} 0 & \text{if } m = 0,\ m^* = 0 \\ N(m^*-1) & \text{if } m = 0,\ m^* \geq 1 \\ q\,\frac{N(m^*)}{N(m)} & \text{else} \end{cases} \tag{4.13a}$$

$$k^* = \sigma^{\mathcal{K}}_{m,m*}(q) = \begin{cases} 0 & \text{if } m = 0,\ m^* = 0 \\ \frac{N(m^*)}{2} & \text{if } m = 0,\ m^* \geq 1 \\ q + \frac{N(m^*)-N(m)}{2} & \text{else} \end{cases} \tag{4.13b}$$

Moreover, an inversion of this mapping could be used in order to get the hierarchic representation $(\mathbf{m}, \mathbf{q})$ of any node $(\mathbf{m}^*, \mathbf{j}^*)$ resp. $(\mathbf{m}^*, \mathbf{k}^*)$, that is, the grid $\mathbf{m}$ of its first occurrence and its index $\mathbf{q}$ there.

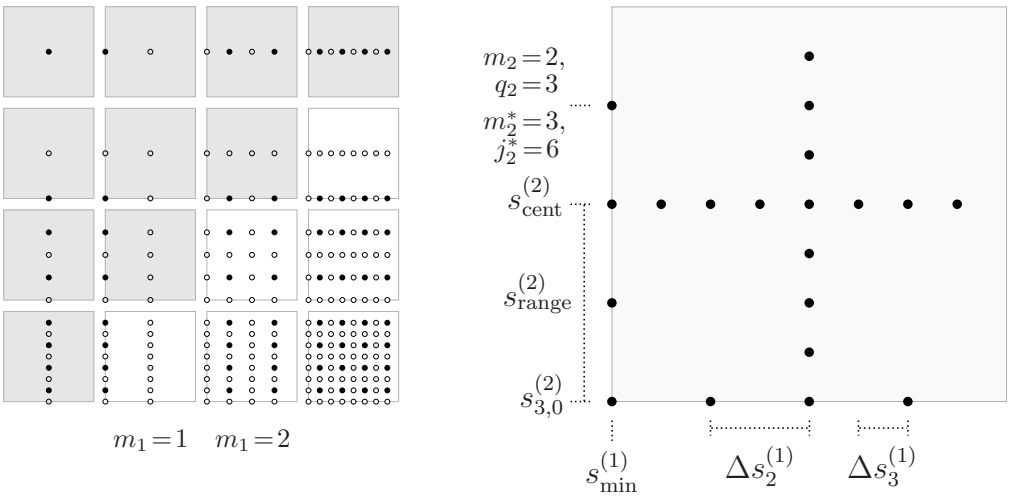

Figure 4: Sparse grid construction in s space illustrated with $n = 3$.

An *iterator*, which visits each node exactly once, can now be constructed by picking every $\mathbf{m} \in \mathcal{G}(\mathbf{n})$, one after the other, and going through each index $\mathbf{q} \in \mathcal{D}_{\mathbf{m}}$ whose node is new to this level.

Figure 4 also illustrates why we have made the unnatural choice $s_{0,0} = s_{\text{cent}}$ in (3.8) instead of $s_{0,0} = s_{\text{min}}$. As we get the option price approximations only on the sparse grid, a fine resolution along a cross through the area of interest $\mathbf{s}_{\text{cent}}$ is by far more rewarding than solutions around $\mathbf{s}_{\text{min}}$. In addition, we expect to get periodicity effects at the borders $\mathbf{s}_{\text{min}}$ and $\mathbf{s}_{\text{max}}$ due to cutting off the payoff as in chapter 3.2.2.

4.1.2 Efficient Computation

The formula (4.5) is not convenient for implementation as it would lead to multiple function evaluations per node. However, we can precalculate the weights for every node of the grid. Therefore, we collect all index pairs $(\mathbf{m}^*, \mathbf{j}^*)$ resp. $(\mathbf{m}^*, \mathbf{k}^*)$ representing the same node $(\mathbf{m}, \mathbf{q})$ in the set

$$\mathcal{U}^{\mathcal{J}}_{\mathbf{m},\mathbf{q}} = \{\ (\mathbf{m}^*, \mathbf{j}^*)\ \mid\ \mathbf{m}^* \in \mathcal{G}(\mathbf{n}),\quad \mathbf{m}^* \geq \mathbf{m},\quad j^*_l = \sigma^{\mathcal{J}}_{m_l, m_l *}(q_l)\ \} \tag{4.14a}$$

$$\mathcal{U}^{\mathcal{K}}_{\mathbf{m},\mathbf{q}} = \{\ (\mathbf{m}^*, \mathbf{k}^*)\ \mid\ \mathbf{m}^* \in \mathcal{G}(\mathbf{n}),\quad \mathbf{m}^* \geq \mathbf{m},\quad k^*_l = \sigma^{\mathcal{K}}_{m_l, m_l *}(q_l)\ \} \tag{4.14b}$$

We then get the *sparse grid weights* $\nu_{\mathbf{m},\mathbf{q}}$ by adding up the excess weights $\hat{\omega}_{\mathbf{m}^*,\mathbf{j}^*}$ from every excess formula $\Delta_{\mathbf{m}^*}$ containing the node $\mathbf{x}_{\mathbf{m},\mathbf{q}}$

$$\nu_{\mathbf{m},\mathbf{q}} = \sum_{\mathcal{U}_{\mathbf{m},\mathbf{q}}} \hat{\omega}_{\mathbf{m}^*,\mathbf{j}^*}. \tag{4.15}$$

The sparse grid approximation (4.5) of the integral can then be expressed as a sum over those weights times the function values

$$\mathcal{Q}^d_{\mathbf{n},\alpha} = \sum_{\mathbf{m}\,\in\,\mathcal{G}(\mathbf{n})}\ \sum_{\mathbf{q}\,\in\,\mathcal{D}_{\mathbf{m}}} \nu_{\mathbf{m},\mathbf{q}}\, f(\mathbf{x}_{\mathbf{m},\mathbf{q}}). \tag{4.16}$$

4.1.3 Error Representation

Novak and Ritter [11] converted the sparse grid quadrature rule (4.5) with $\alpha = 1$ and $n_l = n$ for $l = 1, \ldots, d$ into

$$\mathcal{Q}^d_{n,1} = \sum_{s=0}^{n} \mathcal{Q}^{d-1}_{s,1} \otimes \Delta_{n-s}$$

which leads to the error representation

$$\begin{aligned}\mathcal{R}^d_{n,1} &= \mathcal{I}^d - \mathcal{Q}^d_{n,1} \\ &= \mathcal{I}^{d-1} \otimes \mathcal{R}^1_{n,1} + \sum_{s=0}^{n} \mathcal{R}^{d-1}_{s,1} \otimes \Delta_{n-s}.\end{aligned}$$

As we have chosen a converging sequence of one-dimensional quadrature rules, the first term of this representation gets small for large n. The same holds true for the second term because Δ_{n-s} vanishes for small s and $\mathcal{R}^{d-1}_{s,1}$ for large s by induction. Therefore, we get a quickly converging quadrature rule if the corresponding one-dimensional rules converge fast.

4.1.4 Complexity

We will now turn towards the problem of node counting. The number of points in a full grid with resolution $N = 2^n$ in every direction amounts

$$|\mathcal{Q}_{\mathbf{n}}^d| = O(N^d) = O(2^{dn}) \tag{4.17}$$

As this number grows exponentially with the dimension, data management gets unbearable for $d > 3$ due to enormous memory consumption.

For the complexity of sparse grids with $\alpha = 1$ we take the recursion

$$|\mathcal{Q}_{n,1}^d| = |\mathcal{Q}_{n,1}^{d-1}| + \sum_{m=1}^{n} 2^{m-1} \, |\mathcal{Q}_{n-m,1}^{d-1}|, \qquad |\mathcal{Q}_{n,1}^1| = 2^n \tag{4.18}$$

from [17]. Out of this, we can deduce the formula

$$|\mathcal{Q}_{n,1}^d| = \sum_{j=0}^{\min(n,d-1)} 2^{n-j} \begin{pmatrix} n \\ j \end{pmatrix} \begin{pmatrix} d-1 \\ j \end{pmatrix} \tag{4.19}$$

and get the following asymptotic growth

$$|\mathcal{Q}_{n,1}^d| = O(n^{d-1} \, 2^n) = O(N \, \log^{d-1}(N) \,) \tag{4.20}$$

Notice that the exponential dependence from the dimension moves into the logarithm which is the reason for sparse grids to get interesting in high dimensional algorithms.

An expression for the complexity of generalized sparse grids can be obtained analogously. We count recursively the points of all $d-1$ dimensional subgrids according to the nodes of the one-dimensional grid along dimension d. Thereby, the level bound of the subgrid belonging to the node (m_d, q_d) gets reduced to $h_{d-1} = (h_d^\alpha - \tilde{m}_d^\alpha)^{\frac{1}{\alpha}}$ in order to warrant the condition $|\tilde{\mathbf{m}}|_\alpha < 1$

$$|\mathcal{Q}_{h_d,\alpha}^d| = |\mathcal{Q}_{h_d,\alpha}^{d-1}| + \sum_{m_d=1}^{\lfloor h_d \, n_d \rfloor} 2^{m_d-1} \, |\mathcal{Q}_{h_{d-1},\alpha}^{d-1}|, \qquad |\mathcal{Q}_{h_1,\alpha}^1| = 2^{\lfloor h_1 \, n_1 \rfloor}. \tag{4.21}$$

Due to the real indexing, an analytic investigation of this recursion equation with combinatoric tools is complicated. Therefore, such grids are examined numerically.

4.2 Sparse Grid FFT

As in the full grid case, we can use the FFT in order to speed up the evaluation of a sparse grid quadrature (4.5) for a whole grid of $\mathbf{z}$ values if we consider integrals of type (3.1). The idea consists of evaluating all excess formulas of (4.5) for all nodes of the sparse grid in $\mathbf{z}$ space and add them to those nodes. As the methodology depends on the actual grids in both spaces, we will discuss the procedure directly with the choices made in chapter 3.

4.2.1 Sparse Grid Transform of the Cut Payoff

First of all, we fix the level $\mathbf{m} \in \mathcal{G}(\mathbf{n})$ of the excess formula $\Delta_{\mathbf{m}}$ and consider its calculation for a full grid of $\mathbf{z}$ values which also has level $\mathbf{m}$ and is defined as in (3.6). The FFT can then be directly applied analogue to (3.15)

$$\Delta_{\mathbf{m}}(\mathbf{z}_{\mathbf{m},\mathbf{k}}) \;=\; \prod_{l=1}^{d} e^{i\, z_{m_l,k_l}\, s^{(l)}_{\min}}\, N(m_l)\;\; \text{IFFTD}\;\{h_{\mathbf{m},\mathbf{j}}\} \tag{4.22a}$$

$$h_{\mathbf{m},\mathbf{j}} \;=\; \hat{\omega}_{\mathbf{m},\mathbf{j}}\, \tilde{f}(\mathbf{s}_{\mathbf{m},\mathbf{j}}) \prod_{l=1}^{d} e^{i\, z^{(l)}_{m_l,\min}\, j_l\, \Delta s^{(l)}_{m_l}}. \tag{4.22b}$$

After that, we discuss the evaluation of the excess formula $\Delta_{\mathbf{m}}$ for an excess grid with arbitrary level $\hat{\mathbf{m}} \in \mathcal{G}(\mathbf{n})$ consisting of nodes $\mathbf{z}$ with indices $\hat{\mathbf{q}} \in \mathcal{D}^{\mathcal{K}}_{\hat{\mathbf{m}}}$. As the grids are constructed by tensor products, we can process the index vectors componentwise for each dimension $l = 1, \ldots, d$.

If $\hat{m}_l \le m_l$, the node with index $\hat{q}_l$ corresponds to the node with index k_l of the grid (3.6) with level m_l due to the nestedness of the grids. We can therefore use the mapping

$$k_l = \sigma^{\mathcal{K}}_{\hat{m}_l,m_l}(\hat{q}_l), \qquad p_l = 0. \tag{4.23}$$

in order to read the result of the excess formula at the location k_l from the precalculated grid (4.22).

If $\hat{m}_l > m_l$, the node $\hat{q}_l$ is not part of the grid (3.6) with level m_l. However, the grid spacing in $\mathbf{z}$ space is constant as discussed in chapter 3.3.2 and we can shift this grid by full periods $z^{(l)}_{m_l,\min} \mathrel{+}= p_l \, N(m_l) \, \Delta z^{(l)}$ with $p_l \in \mathbb{Z}$ such that the node is actually part of it. Notice that such a shift does not change $h_{\mathbf{m},\mathbf{j}}$ because of the spacing condition (3.10) and the fact $e^{2\,\pi\,i} = 1$

$$\begin{aligned} \Delta_{\mathbf{m}}(\mathbf{z}_{\mathbf{m},\mathbf{k}}) \;=\; \prod_{l=1}^{d} \; & e^{i\, p_l \, N(m_l)\, \Delta z^{(l)}\, s^{(l)}_{\min}} \\ & e^{i\, z_{m_l,k_l}\, s^{(l)}_{\min}} \, N(m_l) \;\cdot\; \text{IFFTD}\,\{h_{\mathbf{m},\mathbf{j}}\}\,. \end{aligned} \tag{4.24}$$

For this reason, we can reuse the precalculated results from (4.22) by simply multiplying the result with some factors depending on the shift of the grid. The index k_l in the grid after the shift and the number of shifts p_l can be calculated with

$$\begin{aligned} &\hat{k}_l = \hat{q}_l; \quad p_l = 0; \\ &\texttt{while}\;\; \hat{k}_l < \sigma^{\mathcal{K}}_{m_l,\hat{m}_l}(0) && \{\; \hat{k}_l \mathrel{+}= N(m_l); \quad p_l --; \,\} \\ &\texttt{while}\;\; \hat{k}_l > \sigma^{\mathcal{K}}_{m_l,\hat{m}_l}(N(m_l)-1) && \{\; \hat{k}_l \mathrel{-}= N(m_l); \quad p_l ++; \,\} \\ &k_l = \sigma^{\mathcal{K}}_{\hat{m}_l,m_l}(\hat{k}_l) \end{aligned} \tag{4.25}$$

Hence, an excess formula $\Delta_{\mathbf{m}}$ of the sparse grid quadrature (4.5) in $\mathbf{s}$ space can be calculated for all nodes $\mathbf{z}_{\hat{\mathbf{m}},\hat{\mathbf{q}}}$ of the sparse grid in $\mathbf{z}$ space with just one FFT of level $\mathbf{m}$. The algorithm can be summarized as follows

```
for m ∈ G(n) {
   precalculate (4.22b);
   for m̂ ∈ G(n) {
      for q̂ ∈ D^K_m̂ {
         for l = 1,...,d {
            if m̂_l ≤ m_l then (4.23);
            else              (4.25);
         }
         calculate Δ_m for z_{m̂,q̂} with (4.24);
      }
   }
}
```

4.2.2 Sparse Grid Inversion of the Transformed Option Price

In the inversion of the transformed option price we have to evaluate all excess formulas for all nodes of the sparse grid in $\mathbf{s}$ space and add them to those nodes. Unfortunately, the step size in $\mathbf{s}$ space is not constant anymore. We therefore can not calculate an excess formula for just one full grid of $\mathbf{s}$ values and shift it by full periods as in chapter 4.2.1. However, the fact that excess nodes in $\mathbf{s}$ space still form full grids allows us to use FFTs in the evaluation of an excess formula for a particular excess grid of nodes $\mathbf{s}$.

As a start, we fix the level $\mathbf{m} \in \mathcal{G}(\mathbf{n})$ of the excess formula $\Delta_{\mathbf{m}}$ once more. Thereafter, we consider directly its computation for all nodes $\mathbf{s}$ with indices $\hat{\mathbf{q}} \in \mathcal{D}^{\mathcal{J}}_{\hat{\mathbf{m}}}$ of an excess grid with arbitrary level $\hat{\mathbf{m}} \in \mathcal{G}(\mathbf{n})$. The level of the full grid which corresponds to this excess grid can be calculated with

$$\hat{\mathbf{m}}' = \max\left\{\hat{\mathbf{m}} - 1,\ 0\right\}. \tag{4.26}$$

The basic idea of the method consists of doing FFTs on some sort of smallest common subgrids with level $\underline{\mathbf{m}}$ analogously to (3.11)

$$\Delta_{\underline{\mathbf{m}}}(\mathbf{s}_{\underline{\mathbf{m}},\underline{\mathbf{j}}}) \approx \frac{e^{-rT}}{(2\pi)^d}\, e^{-i\, \mathbf{s}^T_{\underline{\mathbf{m}},\underline{\mathbf{j}}}\, \mathbf{z}_{\underline{\mathbf{m}},\min}}\ \mathrm{FFTD}\left\{g_{\underline{\mathbf{m}},\underline{\mathbf{k}}}\right\} \tag{4.27a}$$

$$g_{\underline{\mathbf{m}},\underline{\mathbf{k}}} = \hat{\omega}_{\underline{\mathbf{m}},\underline{\mathbf{k}}}\, \Phi_T(-\mathbf{z}_{\underline{\mathbf{m}},\underline{\mathbf{k}}})\, \mathcal{F}\left\{f^*\right\}(\mathbf{z}_{\underline{\mathbf{m}},\underline{\mathbf{k}}})\, e^{-i\, \sum_{l=1}^{d} s^{(l)}_{\min}\, \underline{k}_l\, \Delta z^{(l)}}. \tag{4.27b}$$

We assign an index $\mathbf{p}$ to each of those subgrids

$$\mathbf{p} \in \mathcal{I}_{\overline{\mathbf{m}}-\underline{\mathbf{m}}} \tag{4.28}$$

where the level vectors are defined by

$$\underline{\mathbf{m}} = \min\left\{\mathbf{m}, \hat{\mathbf{m}}'\right\}, \qquad \overline{\mathbf{m}} = \max\left\{\mathbf{m}, \hat{\mathbf{m}}'\right\}. \tag{4.29}$$

An iteration over the nodes of a subgrid can be performed with

$$\underline{\mathbf{j}} \in \mathcal{I}_{\underline{\mathbf{m}}}, \qquad \underline{\mathbf{k}} \in \mathcal{I}_{\underline{\mathbf{m}}}. \tag{4.30}$$

Because of the tensor product construction of the involved grids, we discuss the components of the index vectors for each dimension $l = 1, \ldots, d$ separately once again.

If $\hat{m}'_l \leq m_l$, the grid of the excess formula is larger or equal to the excess grid in $\mathbf{s}$ space and we have to split it into subformulas of level $\underline{m}_l$. In order to read the data from the nodes of the subformula into the placeholders of the subgrid, we need a mapping between the associated indices

$$k_l = \underline{k}_l + p_l \, 2^{\hat{m}'_l}; \qquad j_l = \underline{j}_l; \tag{4.31}$$

where the excess grid can be mapped onto its full grid as usual

$$\hat{q}_l = \rho^{\mathcal{J}}_{\hat{m}_l}(u_l); \qquad u_l = j_l + 2^{\hat{m}'_l};$$

Then, each of the subformulas can separately be evaluated for all nodes $\hat{\mathbf{q}}$ of the excess grid in $\mathbf{s}$ space with one FFT (4.27).

If $\hat{m}'_l > m_l$, the excess grid is the larger one and has to be splitted into subgrids of level $\underline{m}_l$. In order to add the results from the placeholders of the subgrid to the nodes of the excess grid, we need a mapping between the corresponding indices once again

$$j_l = \underline{j}_l \, 2^{\overline{m}_l - \underline{m}_l} + p_l; \qquad k_l = \underline{k}_l; \tag{4.32}$$

After that, we can calculate the excess formula for each part of the excess grid with an FFT (4.27). A summary of the algorithm is given below

```
for m ∈ G(n) {
   for m̂ ∈ G(n) {
      for p ∈ I_{m̄−m} {
         for k ∈ I_m
            read data from (m, k) to (m, k) using (4.31), (4.32);
         calculate Δ_m for s_{m,j} with (4.27);
         for j ∈ I_m
            write data from (m, j) to (m̂, q̂) using (4.31), (4.32);
      }
   }
}
```

5 Documentation

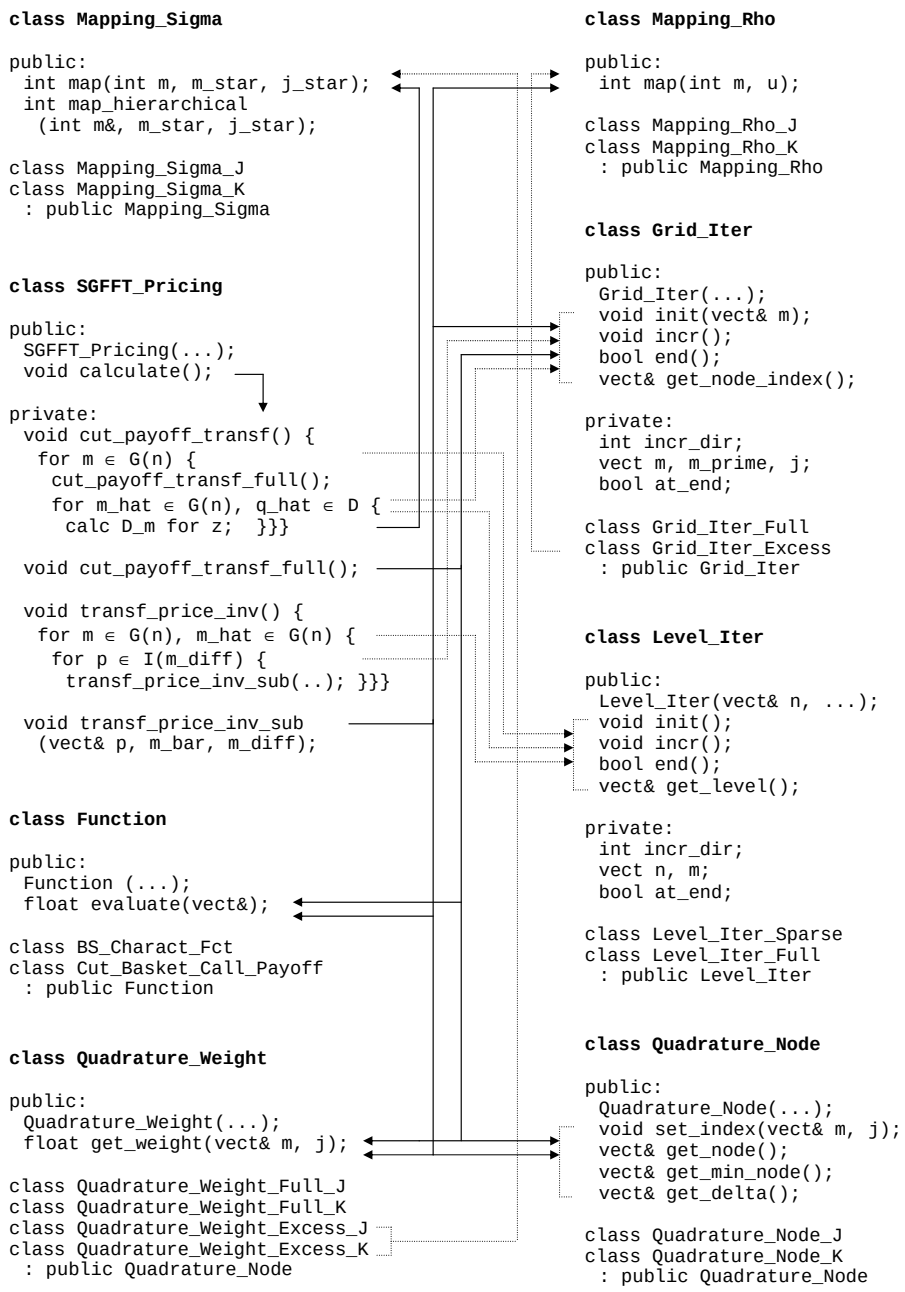

Figure 5: Project overview

The code was written with the goal to create an environment for the investigation of pricing with FFT on different data structures and grid types. For this purpose, we often preferred generic structures over fast inlined implementations. As the d-dimensional FFT dominates the runtime of the algorithm, the optimized library FFTW [27] was used in order to perform this task.

This chapter only presents an overview using pseudocode. A syntactically precise and comprehensive documentation got written in Doxygen [28] and is available as a `html-` or `pdf-` file.

5.1 Parameter Handler

First of all, the `main()-` method selects the desired strategies and instantiates them. The therefore necessary parameters are obtained via their string representations from the `get()-` methods of the `Parameter_Handler`. After that, the strategies are passed to the `SGFFT_Pricing` object which actually calculates the prices.

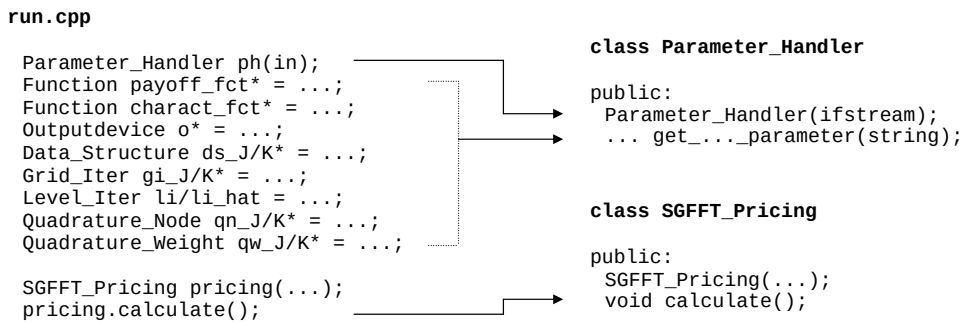

Figure 6: Parameter Handler

Whenever we want to calculate the prices for a whole sequence of parameters, the perl script `parameter_run.pl` could be of interest. By passing an input file with syntax $a : h : b$ for a parameter value to this script, it creates a sequence of input files and initiates the calculation for the parameter values $a, a+h, \ldots, b$. Moreover, some performance results of interest get collected in a text file during the process.

5.2 SGFFT Pricing

The option prices are obtained as described in chapter 4.2. The public function `calculate()` calls `cut_payoff_transf()` in order to compute the values of the transformed payoff $\mathcal{F}\{f^*\}(\mathbf{z}_{\mathbf{m},\mathbf{k}})$ on the sparse grid nodes in $\mathbf{z}$ space analogue to the algorithm of chapter 4.2.1. The precalculation (4.22b) of the excess formulas for a full grid of the same level is done by the function `cut_payoff_transf_full()`. Thereafter, the algorithm of chapter 4.2.2 is used so as to perform the inversion of the transformed prices by calling `transf_price_inv()`. Thereby, the task of evaluating the subformula (4.27) for a full grid, respectively, calculating the full excess formula for a subgrid of nodes in $\mathbf{s}$ space is passed to the function `transf_price_inv_sub()`.

5.3 Data Structure

A generic data structure offers the possibility to implement different mappings from node indices to memory locations. In the application on sparse grid Fourier transforms, mainly a direct mapping onto an array and a mapping via hash functions onto hash maps are worth considering.

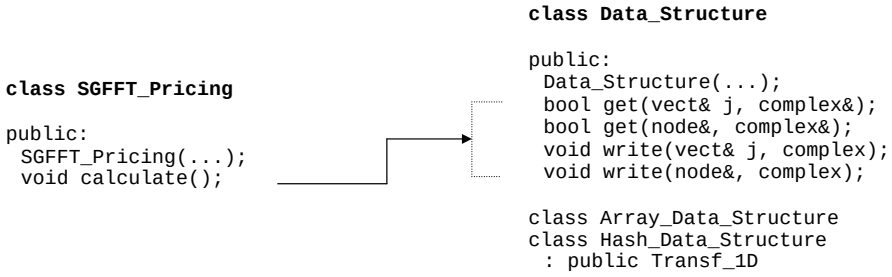

Figure 7: Data Structure

5.3.1 Abstract Strategy

`bool get(vect&, complex&);` returns the value of the specified node index. Thanks to function overloading, it accepts linear indices $\mathbf{j}$ or hierarchical ones $(\mathbf{m}, \mathbf{q})$ and can therefore be used for both representations.

`void write(vect&, complex);` writes a parameter value at the node specified by an index onto the data structure whereas both representations may be used as well.

5.3.2 Concrete Strategies

`class Data_Structure_Array` saves the values of grid nodes directly at their linear indices in a multidimensional array. This simple and efficient structure is a good choice for full grids. In the case of sparse grids, however, many array elements remain unallocated and a hash structure may perform better.

`class Data_Structure_Hash` maps the nodes $(\mathbf{m}, \mathbf{q})$ onto a position `pos` in the hashmap using a hash function `pos = hash(`$(\mathbf{m}, \mathbf{q})$`)` [18]. All nodes $(\mathbf{m}, \mathbf{q})$ are saved at `pos` together with a unique number `nr`. The actual values of the nodes can then be written into a container at position `nr`. The fact that we can save arbitrary node collections favours this data structure for sparse grids.

5.4 Level Iterator

Distinct sparse grid types only differ in the construction of their admissible set $\mathcal{G}(\mathbf{n})$ (4.5). It is therefore obvious to implement an iterator $\mathbf{m} \in \mathcal{G}(\mathbf{n})$ for each grid type. An important aspect of the iteration process is to avoid copying data in order to preserve the complexity of sparse grids.

5.4.1 Abstract Strategy

`void init();` sets the level $\mathbf{m}$ back to the beginning of the iteration sequence and initializes the private members of the iterator.

`void incr();` increases the iteration sequence by one step. Whenever the admissible set $\mathcal{G}(\mathbf{n})$ is given by an inequality, we can get the next level $\mathbf{m}$ by setting $m[i] = 0$ for $i \in [0, \ldots,$ `incr_dir`$]$ and increasing $m[$`incr_dir`$]$ by one where `incr_dir` is the first dimension into which an index increase would not hurt the said inequality.

`bool end();` returns true if the iteration sequence has reached its end. The actual state of the iterator is meaningless in this case.

`vect& get_level();` returns the actual level $\mathbf{m}$.

5.4.2 Concrete Strategies

`class Level_Iter_Full` iterates just over the single level $\mathbf{m} = \mathbf{n}$ and can therefore be used in a direct full grid construction.

`class Level_Iter_Sparse` implements an iterator over the levels $\mathbf{m} \in \mathcal{G}(\mathbf{n})$ of a generalized sparse grid where $\mathcal{G}(\mathbf{n})$ is defined in (4.7).

5.5 Grid Iterator

The task of a grid iterator is to visit all nodes belonging to a particular level. As we encountered different types of grids in our method derivation, for example, full and excess grids, we apply the strategy pattern [24] once more in order to allow a generic treatment of such iterators.

5.5.1 Abstract Strategy

`void init(vect& m);` sets the level $\mathbf{m}$, initializes the private members and start with the node index $\mathbf{j}$ at the beginning of the iteration sequence.

`void incr();` increases the iteration sequence by one step.

`bool end();` returns true, if the iteration sequence has reached its end. The actual state of the iterator is meaningless in this case.

`vect& get_node_index();` returns the actual node index $\mathbf{j}$.

5.5.2 Concrete Strategies

`class Grid_Iter_Full` iterates over a full tensor product grid of level $\mathbf{m}$, that is, over every $\mathbf{j} \in \mathcal{I}_{\mathbf{m}}$. The function `incr()` is realized similar to the one of the level iterator in chapter 5.4.1. It sets $j[i] = 0$ for $i \in [0, \ldots,$ `incr_dir`$]$ and increases $j[$`incr_dir`$]$ by one where `incr_dir` is the first dimension l into which an index increase would not hurt the inequality $j_l < N(m_l)$.

`class Grid_Iter_Excess` visits just the nodes which are new to level $\mathbf{m}$ by iterating over a full grid $\mathbf{u} \in \mathcal{I}_{\mathbf{m}'}$ and mapping those indices to the excess grid using $j_l = \rho_{m_l}(u_l + 2^{m'_l})$ where $\mathbf{m}' = \max\{\mathbf{m} - 1, 0\}$ and $l = 1, \ldots, d$. The mapping ρ may be passed to the constructor which allows us to create iterators over excess grids of typ $\mathcal{J}$ and $\mathcal{K}$.

5.6 Function

The payoff of an option such as the characteristic function of a model are both functions from $\mathbb{R}^d$ to $\mathbb{R}$ and can therefore be realized as concrete strategies derivated from a generic function object. Such an object offers an `evaluate(vect&)` function which takes a node $\mathbf{x}$ and returns the value of the function at this node.

Bear in mind, that the calculation arithmetic of C++ knows infinity, for example, $\log(0) = -\infty$, $\exp(-\infty) = 0$. This fact gets relevant if we want to extend the function object by transformations $\varphi(\mathbf{x})$. That is to say, we can then improve the use of floating point precision by using logarithms since the function value can get very large and the jacobi determinant very small at the same time.

5.6.1 Concrete Strategies

`class BS_Charact_Fct` evaluates the characteristic function $\Phi_T(\mathbf{z})$ of the Black Scholes model by calculating (2.5).

`class Cut_Basket_Call_Payoff` returns the payoff of a call option on a portfolio of underlying assets (2.6)

5.7 Quadrature Nodes

In order to get the coordinates of a node, we need an object which calculates them out of the level and node index. As we came across two different types of full grid expansion in chapter 3, one in $\mathbf{s}$ and one in $\mathbf{z}$ space, we will make use of the strattegy pattern once more.

5.7.1 Abstract Strategy

`void set_index(vect& m, j);` selects the index $\mathbf{j}$ in the full grid of level $\mathbf{m}$ and calculates its coordinates $\mathbf{x}_{\mathbf{m},\mathbf{j}}$. The spacing $\Delta\mathbf{x}_\mathbf{m}$ of the full grid and its smallest node $\mathbf{x}_{\mathbf{m},\min}$ are noticed as a side product.

`vect& get_node();` returns the coordinates $\mathbf{x}_{\mathbf{m},\mathbf{j}}$ of the selected node.

`vect& get_min_node();` returns the smallest node $\mathbf{x}_{\mathbf{m},\min}$ of the grid.

`vect& get_delta();` returns the spacing $\Delta\mathbf{x}_\mathbf{m}$ of the full grid of level $\mathbf{m}$.

5.7.2 Concrete Strategies

`class Quadrature_Node_J` implements the grid of initial stock prices $\mathbf{s_{m,j}}$ defined by (3.8). This grid gets refined for an increase in the level.

`class Quadrature_Node_K` creates the grid in $\mathbf{z}$ space proposed in (3.6). In contrast to the grid of type $\mathcal{J}$, an increasing level leads to grid expansion due to the spacing condition (3.10).

5.8 Quadrature Weights

Since we want to allow ordinary as well as sparse grid quadrature and to leave open the possibility of extending the method by other quadrature rules or an efficient computation described in chapter 4.1.2, we need a generic treatment of the weights. An appropriate strategy offers a `get_weight(vect& m,j)` function which returns the weight of the node $(\mathbf{m}, \mathbf{j})$.

5.8.1 Concrete Strategies

`class Quadrature_Weight_Full_J` calculates the weights of an ordinary full grid quadrature rule in $\mathbf{s}$ space.

`class Quadrature_Weight_Full_K` deals with the corresponding full grid weights in Fourier space $\mathbf{z}$.

`class Quadrature_Weight_Excess_J` returns the excess weights of a sparse grid quadrature rule in the space of initial underlying prices $\mathbf{s}$.

`class Quadrature_Weight_Excess_K` computes excess weights in $\mathbf{z}$ space.

5.9 Mappings

Many algorithms, for example, an excess grid iterator, do not need to know the actual space they work in even if they depend on it via a mapping. They work with generic function objects and can be used for both spaces by passing the corresponding concrete mappings. Such a generic object offers a function `map` which takes some indices and returns the mapped ones. Two important concrete objects in the sparse grid construction are the mapping $\rho_m(u)$ (4.12) from excess indices u onto hierarchical ones q and the mapping $\sigma_{m,m^*}(q)$ (4.13) from a node (m, q) onto its corresponding index (m^*, j^*).

5.10 Output Devices

Naturally, the required information on the algorithm's activities during the computation depends strongly on the particular field of application. If we want to, for example, debug and investigate the algorithm's behaviour, we would like to observe every relevant arithmetic operation in order to reveal a possible misbehaviour early. In a benchmark, however, the memory consumption, the number of particular operations per time unit as well as the asymptotic behaviour of some quantities play an important role and in applications, only the result matters. In order to intervene as little as possible in the algorithm itself if the information requirements change, we make use of different output devices. The idea is to provide an output interface which always gets invoked by the algorithm and concrete devices which treat those calls accordingly. A device printing only the result, for example, can simply ignore every call except for the one of `write_summary()`.

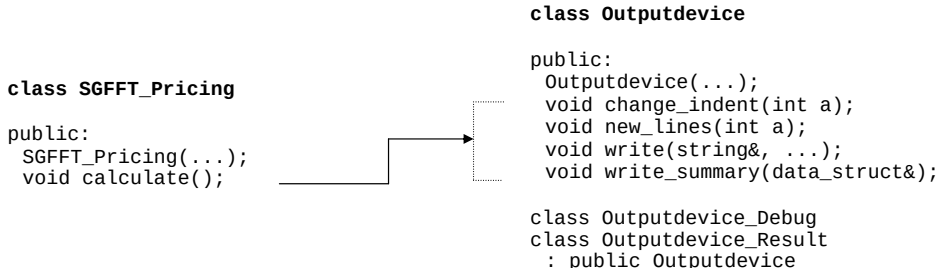

Figure 8: Outputdevice

5.10.1 Abstract Strategy

`void change_indent(int a);` simply changes the indent of the text.

`void write(string&, ..)` writes a complex number, a vector or even a whole grid of values together with a description.

5.10.2 Concrete Strategies

`class Outputdevice_Debug` prints almost every relevant state of the data structure and should therefore be used for debugging.

`class Outputdevice_Result` writes just the input parameters, some quantities of the algorithm as well as the resulting option values.

6 Results

As a first step, we will discuss the different methods of chapter 3 in the one-dimensional case so as to get a feeling for their behavior and the drawbacks of an extension into the multidimensional setup. In a second part, we will analyze the multidimensional double discretization method of chapter 3.3.2 on full and sparse grids.

6.1 Analysis of One-Dimensional Methods

The impact of the dampening factor onto the method of Lewis [15] can be seen in figure 9. A small factor results in a diverging denominator of the transformed option price which leads to large discretization errors. If we choose it too large, however, the transformed price decays more slowly and we are confronted with bigger truncation errors.

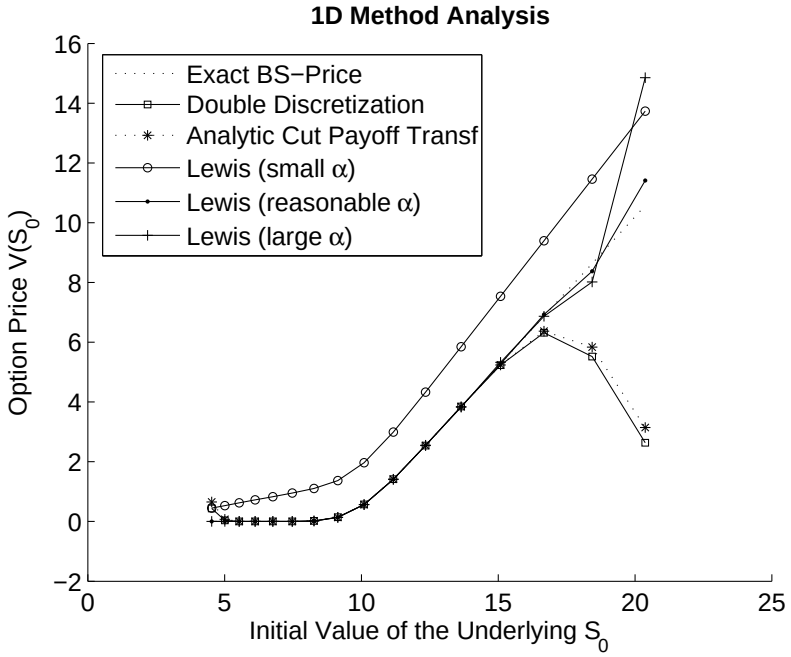

Figure 9: Calculated prices of a one-dimensional call option

We can avoid choosing the dampening factor by replacing the payoff with a modified one as described in chapter 3.2.2. However, the error explodes for initial stock prices $\mathbf{s}$ close to the cut off locations $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$ because of a huge payoff discrepancy after them.

The convergence of the option price at $\mathbf{s}_{\text{cent}}$ can be seen in figure 10. As expected, we can recognize that the method of Lewis converges exponentially due to the fast decay of the characteristic function. However, the constant step size in $\mathbf{z}$ space leads to a persistent discretization error which could be scaled down by increasing the dampening factor or refining the grid in $\mathbf{z}$ and therefore expanding it in $\mathbf{s}$ space.

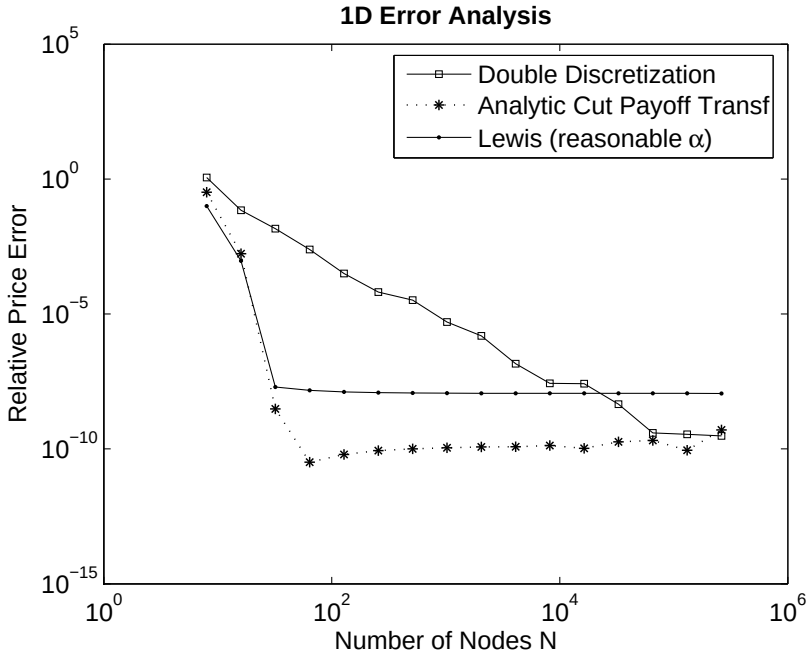

Figure 10: Relative price errors at $\mathbf{s}_{\text{cent}}$

As discussed in chapter 3.1, we also get exponential convergence if the Fourier transform of the payoff is known analytically because of the fast decay of the characteristic function. However, the double discretization method applies a discretized payoff transform which is of first order as the payoff is not analytic in a stripe around the real axis. We therefore lose exponential convergence if the Fourier transform of the payoff is not known analytically.

Another important feature of figure 10 is the stagnant convergence of the methods using a modified payoff. This residual error comes from the fact, that a wrong contract is approximated and could be scaled down by increasing the actual cut off locations, that is, by expanding the grid, respectively, by increasing the cut off parameters $\mathbf{y}_{\min}$ and $\mathbf{y}_{\max}$.

6.2 Analysis of the Double Discretization Method

We start with an illustration of the results produced by the double discretization method. Figure 11 shows the calculated option prices for a sparse grid of underlying stock values $\mathbf{s}$ in comparison with the exact two-dimensional Black-Scholes prices.

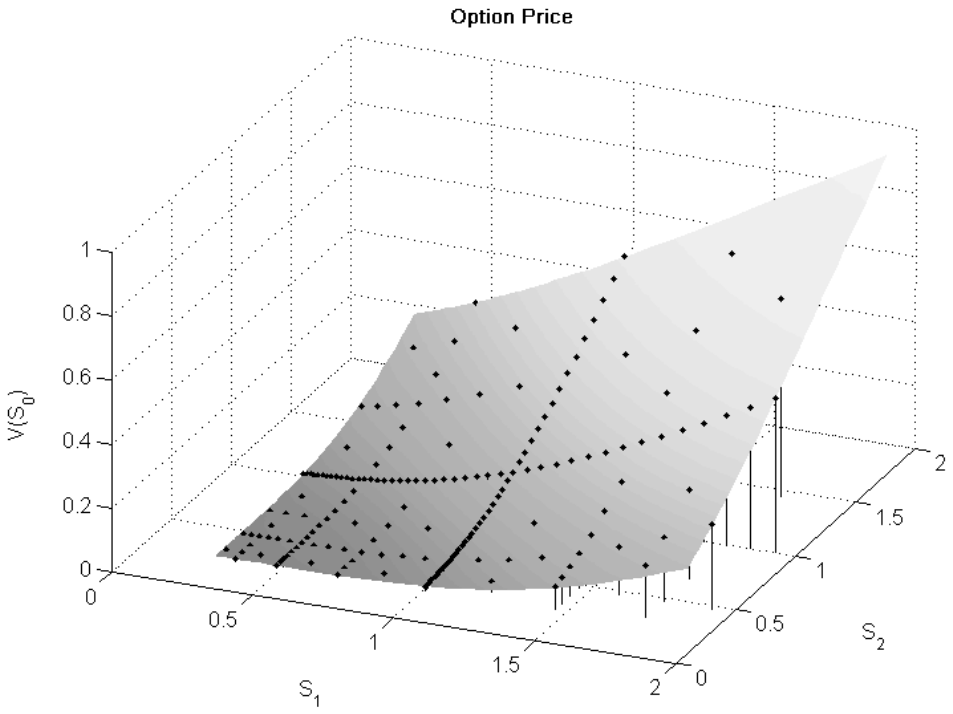

Figure 11: Sparse grid option prices in 2D

We can recognize the fine resolution along the axes through $\mathbf{s}_{\text{cent}}$ and the rougher spacing elsewhere. Notice that we have cut off the most outer nodes because of the periodic effects of the modified payoff which could already be seen in figure 9.

The convergence of the double discretization method at $\mathbf{s}_{\text{cent}}$ for two up to four dimensions can be seen in the figures 12 to 14.

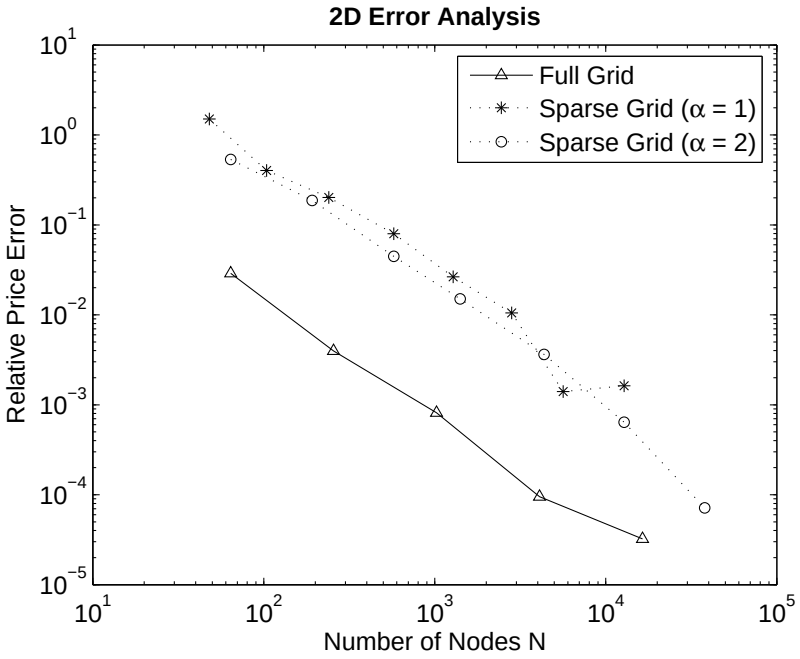

Figure 12: Relative price errors at $\mathbf{s}_{\text{cent}}$

As expected, we get first order convergence and a slight constant superiority of the full grid method in the two or three dimensional case. The reason for this is that in low dimensions there is still a large part of important informations not lying near any axis which are simply not resolved fine enough in the sparse grid case.

Notice, by the way, that we can somehow interpolate between full and sparse grid behaviour by choosing a sparse grid parameter α lying in $(1, \infty)$. This may come in handy when proceeding into higher dimensions because a full grid level increase will then lead to a huge jump in the number of nodes. However, a number somewhere in between might be preferred due to low accuracy needs or given memory restrictions. A smaller sparse grid parameter α would reduce said jump and therefore allow us to choose such an intermediate number of nodes.

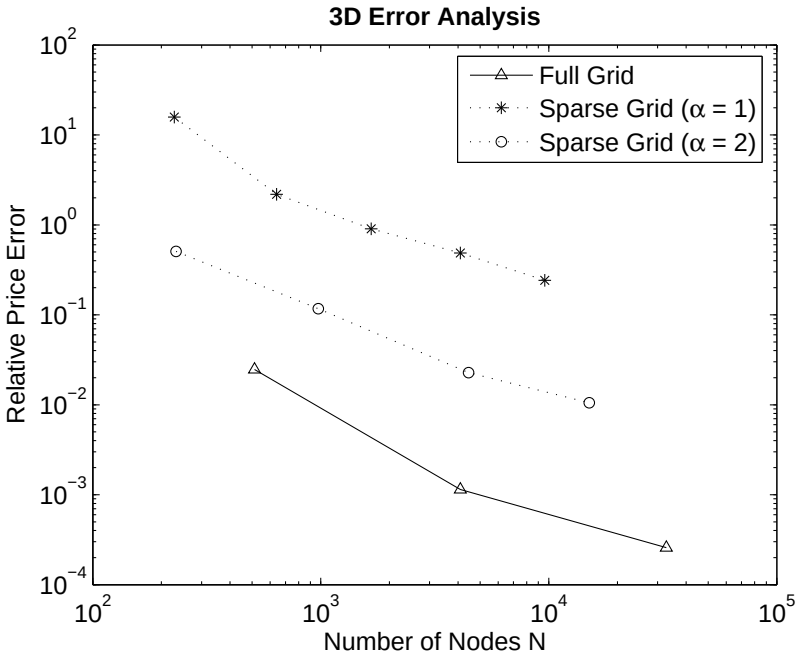

Figure 13: Relative price errors at $\mathbf{s}_{\text{cent}}$

By proceeding into higher dimensions, however, the sparse grid method starts to catch up as the part of important informations lying off the axes gets smaller and smaller. We have seen in [14] that this transition starts to happen at about five or six dimensions. In our case, however, such dimensions are hardly obtainable on an ordinary computer as calculating prices for a whole grid of underlying values is more expensive. Therefore, we have shrunk the size of the grid $\mathbf{s}_{\text{range}}$ in order to reduce the part of important informations lying off the axes and provoked the transition to already happen in the four-dimensional case as seen in figure 14. As a side effect, the convergence has begun to stagnate after some large number of nodes due to a larger spacing in the $\mathbf{z}$ grid which starts to dominate the discretization error.

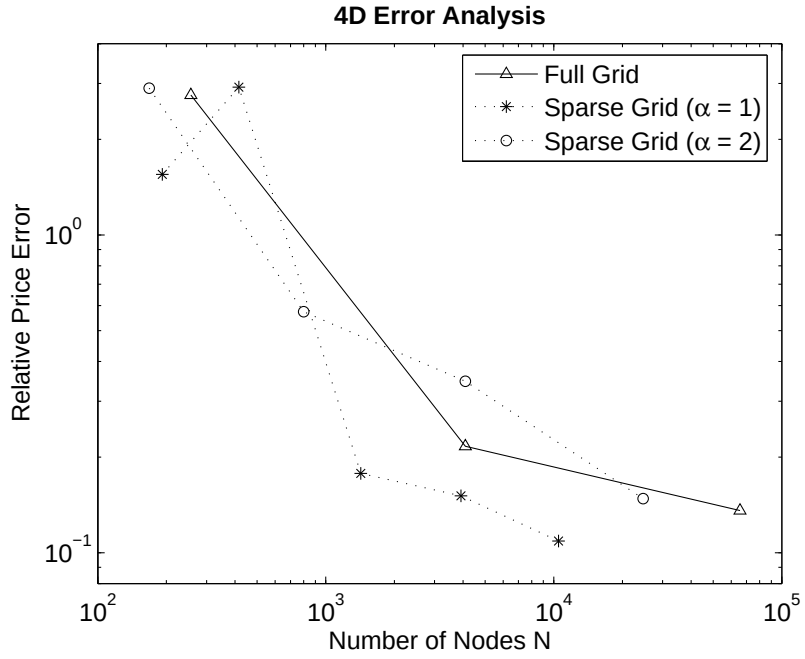

Figure 14: Relative price errors at $\mathbf{s}_{\text{cent}}$

Figure 15 reveals the fact, that the relative price error remains constant for an increasing number of dimensions but unchanging level.

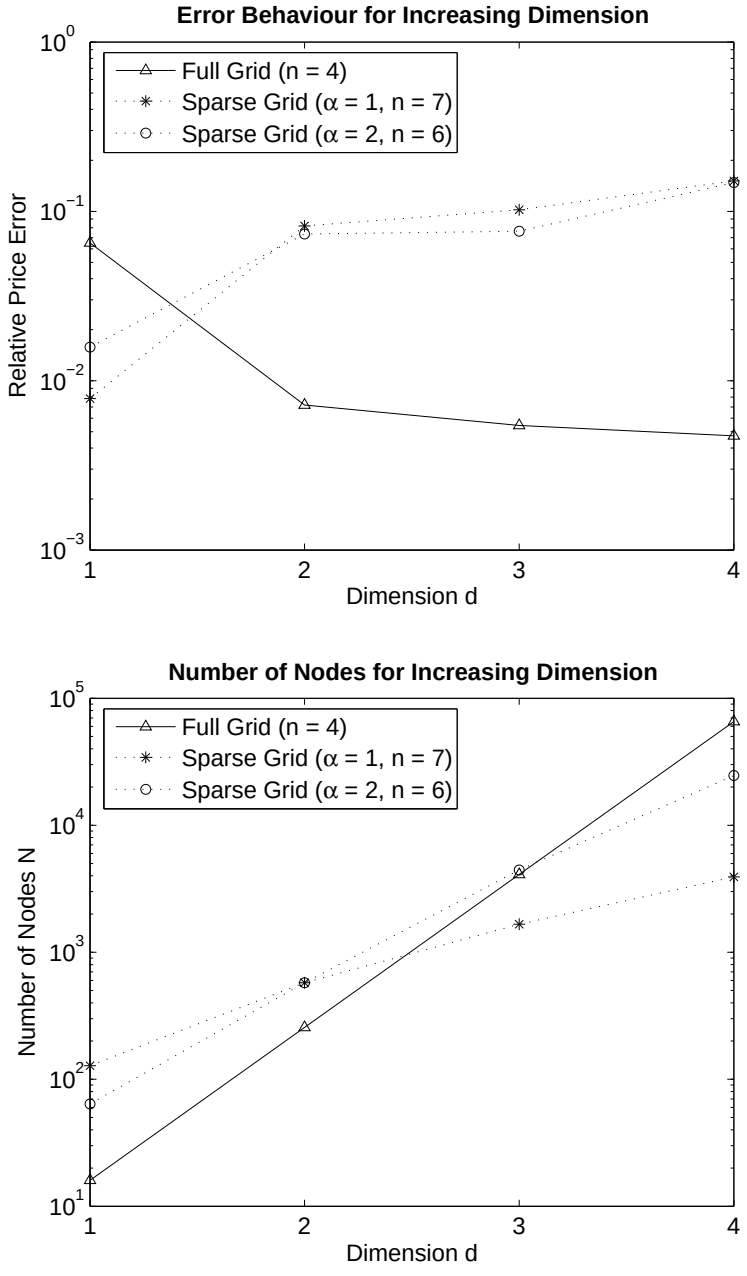

Figure 15: Errors and number of nodes for increasing dimension

Together with the realization that the number of nodes grows slower than exponentially for increasing dimension but unchanging level, we can conclude

that the effort for increasing the number of dimensions at constant accuracy grows slower than exponentially. Therefore, we are able to break the curse of dimensionality using sparse grids.

Finally, we conclude from figure 16 that the calculation time for full grid problems grows almost linear in the number of nodes as there are many other efforts aside from the two-dimensional FFT. The asymptotic effort of the sparse grid problems grows a bit steeper but all in all they are just slower by some constant factor. This fact supports our assumption that the number of nodes is a good asymptotic measure for the complexity of methods.

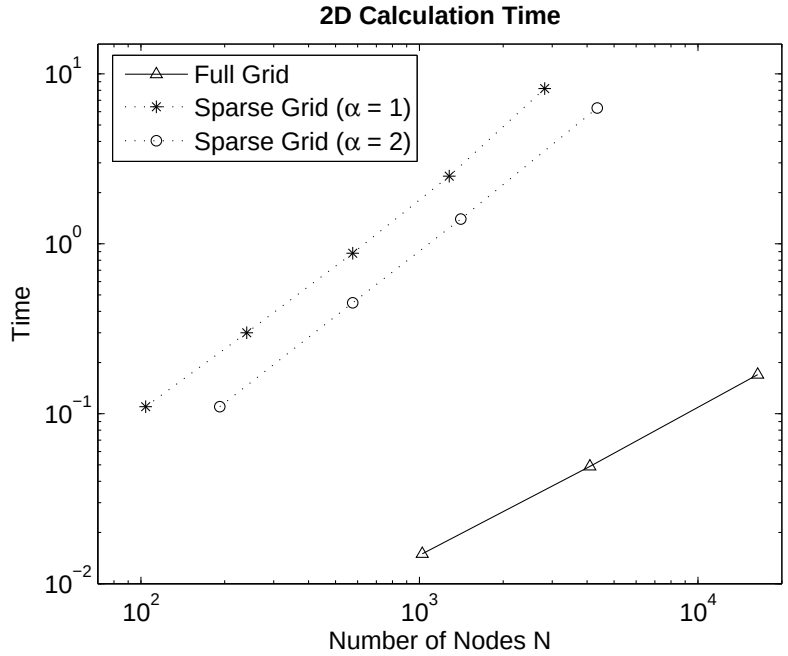

Figure 16: Calculation time in the two-dimensional case

7 Suggestions for Future Work

The extension of FFT methods to the multidimensional case has brought new ideas but also many new problems. We will use this chapter in order to present some suggestions for possible future investigations.

7.1 Dismiss the FFT Algorithm

The main drawback of using FFT Algorithms in order to evaluate the occuring sums for a whole grid of parameters is that we are forced to choose equidistant spacings in quadrature rule and parameter grid which are, moreover, linked by condition (3.10). However, there are many promising transformations and non-equidistant quadrature rules which would hurt those restrictions. Therefore, it could be interesting to evaluate the sums for each parameter individually without using the FFT.

At the moment the limiting factor is the numerical transformation of the payoff which is of first order due to the function not beeing analytic in a stripe around the real axis. However, we could think of eliminating the kink with transformations and achieving exponential convergence using more sophisticated quadrature rules such as Kronrod Patterson and Clenshaw Curtis. In the case of basket call options, for example, we could remove the kink with a simple one-dimensional linear transformation. In order to do so, we ought to write the transformed cut payoff as

$$\mathcal{F}\left\{\tilde{f}\right\}(\mathbf{z}) = \int_{y_{\min}^{(d)}}^{y_{\max}^{(d)}} \cdots \int_{y_{\min}^{(2)}}^{y_{\max}^{(2)}} \int_{y^*}^{y_{\max}^{(1)}} e^{i\,\mathbf{z}^T\mathbf{y}} \left(\mathbf{c}^T e^{\mathbf{y}+rT} - K\right) d\mathbf{y} \tag{7.1}$$

where the kink position y^* in dimension one can be calculated with

$$\mathbf{c}^T e^{(\tilde{y},y_2,\ldots,y_d)^T+rT} - K = 0 \quad \rightarrow \quad \tilde{y}, \qquad y^* = \max\left\{\tilde{y}, y_{\min}^{(1)}\right\}$$

Substituting $\int_{y^*}^{y_{\max}(1)} .. dy_1 \rightarrow \int_{y_{\min}(1)}^{y_{\max}(1)} .. dt$ would then remove the kink.

Another downside of our method is the expansion of the $\mathbf{z}$ grid. As the characteristic function decays faster than every power of its argument at infinity, we have to discretize some sort of bump. However, we need quite a high level or sparse grid parameter α in order to get nodes lying far away from the origin but not near to any axis. This problem will probably get limiting as soon as we will be able to speed up the convergence of the payoff transformation because we could then reduce the number of levels.

Last but not least, dismissing the FFT algorithm would result in a more flexible memory management beeing particularly interesting for parallelizations. If we were only interested in the price at $\mathbf{s}_{\text{cent}}$, for example, we could think of algorithms whose memory requirement does neither depend on the resolution nor on the dimension. This could be achieved by calculating the transformed payoff for a particular $\mathbf{z}$ value not before it is actually used in the quadrature rule of the inverse transform.

7.2 Analytic Transformation of the Payoff

It does not hurt to repeat once more that we could simply achieve exponential convergence by calculating the transformed payoff $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ analytically. The resulting closed form expression for the Fourier transform of the option price (3.3) could be approximatively inverted by (3.11). Due to the smoothness and the fast decay of the transformed price, even a rectangular rule would yield up to exponential convergence as investigated in [13].

Unfortunately, the analytic calculation of $\mathcal{F}\{\tilde{f}\}(\mathbf{z})$ is not at all straight forward. Nevertheless, it is promising to invest some more work into this problem. For example, it could be possible to use some sort of recursive evaluation algorithm or simpler modified payoffs.

7.3 Parallelization

The parallelization of the summation over every node in $\mathbf{z}$ grid for a whole grid of $\mathbf{s}$ values (3.11) is a typical broadcast problem. Even if we were able to find an algorithm which evenly distributes data storage and workload, each computer would have to access data from every other computer during the inverse transform. The therefore necessary communication effort will rather soon get limiting. At least, the deterministic structure of the algorithm would allow us to avoid master-slave pradigms.

Notice that working without data structures and FFTs would allow us to parallelize the algorithm using almost no communication. A simple approach would be to specify a block size of function evaluations h and the number of processors p in advance. The CPU with number $i = 1, \ldots, p$ could process only the nodes $(i-1+kp)\,h+1, \ldots, (i+kp)\,h$ for $k = 0, 1, \ldots$ where the order of nodes is defined by the sequence which a single computer would use. The communication effort would then reduce to the collection of the summation blocks. Notice that one disadvantage remains compared to Monte Carlo

methods. That is to say, a wrong result occurs when one CPU malfunctions which is quite likely for a large number of computers. However, Monte Carlo methods are hardly affected by such defects because of the mean still beeing a good approximation. We could fix this problem by using redundant calculations.

7.4 American Contracts

Extending the method in order to deal with american contracts is straight forward by using the CONV method described in [4]. We just have to apply a time-stepping scheme to our method by starting with

$$V(\mathbf{s}, t_p) = f(e^{\mathbf{s}}), \qquad T = t_p$$

and calculating

$$\tilde{V}(\mathbf{s}, t_{i-1}) = e^{-r\,\Delta t} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} V(\mathbf{s}+\mathbf{x}+r\,\Delta t, t_i)\, \rho_{\Delta t}(\mathbf{x})\, d\mathbf{x}$$

for each time step $i = 1, \ldots, p$ where $T = t_p$ is the maturity of the option. The option value at time t_{i-1} can then be obtained by applying an early exercise condition

$$V(\mathbf{s}, t_{i-1}) = \max\{\, \tilde{V}(\mathbf{s}, t_{i-1}), f(e^{\mathbf{s}}) \,\}\,.$$

8 Conclusions

We have discussed an extension of Lewis' method [15] on multidimensional contracts. As damping in one direction is not enough for the Fourier transform of the payoff to exist in two or more dimensions, we have modified the payoff by cutting it. Unfortunately, the analytic calculation of this transform is usually very complicated or even impossible. Therefore, we have proposed to use another quadrature rule in order to compute the values of the transformed payoff for all nodes $\mathbf{z}$ required by the approximative inversion of the option price. Choosing equidistant grids and satisfying a spacing condition allowed us to evaluate both quadrature rules using multidimensional FFTs. However, we have lost exponential convergence as the payoff transform converges with first order. Exponential convergence could only be regained by finding a method to calculate the payoff transform analytically or by dropping the FFT in order to be able to use more sophisticated quadrature rules and grid transformations.

In a second step, we have worked out a generic treatment of the method on sparse grids where the calculation speed has just dropped by some constant factor. Sparse grids get particularly interesting in high dimensions as they start to converge faster than full grid methods and allow us to be more flexible in choosing the number of nodes which is directly related to the computational effort and memory requirements.

All in all, we see our method as a viable alternative for low dimensions as the calculation is extremely fast on full grids. The main advantages over other methods are that we get all option prices on a whole grid of underlying strike prices simultaneously, that we just need to know the characteristic function and not the density of $\mathbf{X}_T$ in closed form and that calculating american contracts on full and sparse grids is straight forward. Moreover, we think that the method has a lot of potential for improvements, especially in higher dimensions, by investigating our suggestions for future work.

References

[1] F. Black, M. Scholes, *The Pricing of Options and Corporate Liabilities*, J. Polit. Econ. 81, 637-654, 1973

[2] P. Carr, D.P. Madan, *Option Valuation Using the Fast Fourier Transform*, J. Comput. Finance 2 (4), 61-73, March 1999

[3] S. Raible, *Lvy Processes in Finance: Theory, Numerics, and Empirical Facts*, Ph.D. thesis, University of Freiburg, 2000

[4] F. Fang, R. Lord, C.W. Oosterlee, *Fast and Accurate Methods in Pricing Early Exercise Options Under Lvy Processes*, Ecole Polytechnique, Workshop on Fin. Modelin with Jump proc., September 2006

[5] E. Eberlein, A. Papapantoleon, *Valuation of Exotic and Credit Derivatives in Lvy Models*, University of Freiburg, Workshop on Credit Risk under Lvy Models, September 2006

[6] R. Cont, P. Tankov, *Financial Modelling with Jump Processes*, Chapman & Hall / CRC Press, 2003

[7] T. Gerstner, M. Griebel, *Numerical Integration using Sparse Grids*, Numer. Algorithms, 18, 209-232, (1998)

[8] T. Gerstner, M. Griebel, *Dimension-adaptive tensor-product quadrature*, Computing, 71(1):65-87, 2003.

[9] S.A. Smolyak, *Quadrature and interpolation formulas for tensor products of certain classes of fcts*, Soviet Math. Dokl. 4 (1963), 240-243

[10] K. Petras, *On the Smolyak cubature error for analytic functions*, Advances in Computational Mathematics 12 (2000), 71-93

[11] E. Novak and K. Ritter, *High-dimensional integration of smooth functions over cubes*, Numer. Math. 75 (1996) 79-97.

[12] P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Academic Press, New York, 1975.

[13] J. A. C. Weideman, *Numerical integration of periodic functions: a few examples*, Amer. Math. Monthly 109 (2002), 21-36.

[14] S. Villiger, *Optionsbewertung mittels hochdimensionaler numerischer Integration*, Bachelor Thesis, SAM, ETH Zürich, 2005

[15] A. L. Lewis, *A Simple Option Formula for General Jump-Diffusion and Other Exponential Lvy Processes*, Envision Fin. Sys. and OptionCity.net, August 2001

[16] J. Bronni, *Effiziente Fouriertransformation auf Dünnen Gittern*, Diplomarbeit, Universität Tübingen, 2005

[17] K. Hallatschek, *Fouriertransformation auf Dünnen Gittern mit Hierarchischen Basen*, Numerische Mathematik, 63, 1992

[18] C. Gasquet, P. Witomski, R. Ryan, *Fourier Analysis and Applications*, Springer, 1999

[19] Vasile Gradinaru, *Whitney Elements on Sparse Grids*, Dissertation, Universität Tübingen, April 2002

[20] K.-I. Sato. *Lévy processes and infinitely divisible distributions*, Cambridge Studies in Advanced Mathematics Volume 68, 1999

[21] J. Kallsen and P. Tankov, *Characterization of dependence of multidimensional Lévy processes using Lévy copulas*, Journal of Multivariate Analysis, (97):1551-1572, 2006

[22] P. Tankov, *Dependence structure of Lévy processes with applications to risk management*, Rapport Interne No. 502, CMAPX École Polytechnique, Mars 2003.

[23] D. Madan and P. Carr and E. Chang, *The Variance Gamma Process and Option Pricing*, European Finance Review, 2, 79-105, 1998

[24] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series

[25] T. Ottmann und P. Widmayer, *Algorithmen und Datenstrukturen*, Spektrum 4. Auflage 2002

[26] `http://www.wikipedia.org/`, *Wikipedia Encyclopedia*, The biggest multilingual free-content encyclopedia on the internet

[27] `http://www.fftw.org/`, *Fastest Fourier Transform in the West*, C subroutine library for computing FFTs

[28] `http://www.stack.nl/~dimitri/doxygen/`, *Documentation System for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D*