DEGREE PROJECT, IN MATHEMATICAL STATISTICS , SECOND LEVEL
STOCKHOLM, *SWEDEN 2015*

Efficient Sensitivity Analysis using Algorithmic Differentiation in Financial Applications

ERIC SUNNEGÅRDH, LUDVIG LAMM

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCI SCHOOL OF ENGINEERING SCIENCES

Efficient Sensitivity Analysis using Algorithmic Differentiation in Financial Applications

ERIC SUNNEGÅRDH
LUDVIG LAMM

Master's Thesis in Mathematical Statistics (30 ECTS credits)
Master Programme in Applied and Computational Mathematics (120 credits)
Royal Institute of Technology year 2015
Supervisors at Cinnober: Magnus Sandström
Supervisor at KTH: Thomas Önskog
Examiner: Thomas Önskog

TRITA-MAT-E 2015:70
ISRN-KTH/MAT/E--15/70-SE

Royal Institute of Technology
SCI School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

Efficient Sensitivity Analysis using Algorithmic Differentiation in Financial Applications

by

Eric Sunnegårdh and Ludvig Lamm

One of the most essential tasks of a financial institution is to keep the financial risk the institution is facing down to an acceptable level. This risk can for example be incurred due to bought or sold financial contracts, however, it can usually be dealt with using some kind of hedging technique. Certain quantities refereed to as "the Greeks" are often used to manage risk. The Greeks are usually determined using Monte Carlo simulation in combination with a finite difference approach, this can in some cases be very demanding considering the computational cost. Because of this, alternative methods for determining the Greeks are of interest.

In this report a method called Algorithmic differentiation is evaluated. As will be described, there are two different settings of Algorithmic differentiation, namely, forward and adjoint mode. The evaluation will be done by firstly introducing the theory of the method and applying it to a simple, non financial, example. Then the method is applied to three different situations often arising in financial applications. The first example covers the case where a grid of local volatilities is given and sensitivities of an option price with respect to all grid points are sought. The second example deals with the case of a basket option. Here sensitivities of the option with respect to all of the underlying assets are desired. The last example covers the case where sensitivities of a caplet with respect to all initial LIBOR rates, under the assumption of a LIBOR Market Model, are sought.

It is shown that both forward and adjoint mode produces results aligning with the ones determined using a finite difference approach. Also, it is shown that using the adjoint method, in all these three cases, large savings in computational cost can be made compared to using forward mode or finite difference.

Sammanfattning

Effektiv Känslighetsanalys med Algoritmisk Differentiering i Finansiella Tillämpningsområden

av

Eric Sunnegårdh och Ludvig Lamm

En av de mest centrala uppgifter för en finansiell institution är att hålla sina finansiella risker på en acceptabel nivå. De risker som avses kan till exempel uppkomma på grund av köpta eller sålda finansiella kontrakt. Oftast kan dock dessa risker hanteras med hjälp av någon typ av garderingsteknik. Ett antal känslighetsmått som används för att gardera mot risk är ofta refererade till som *the Greeks*. Vanligtvis kan dessa beräknas genom att använda Monte Carlo-simulering i kombination med finita differensmetoden, detta kan dock bli mycket krävande med avseende på den datorkraft som behövs för beräkningarna. Därför är andra metoder för att beräkna *the Greeks* av intresse.

I denna rapport utvärderas en metod som kallas Algoritmisk differentiering. Som det senare beskrivs, finns det två typer av Algoritmisk differentiering, vilka kallas *forward mode* och *adjoint mode*. Utvärderingen görs genom att först introducera teorin bakom metoden och sedan appliceras den på ett lättare, icke finansiellt, exempel. Därefter appliceras metoden på tre tillämpningsområden inom finansindustrin. Det första exemplet beskriver ett fall där ett rutnät av volatiliteter är givet och känsligheter av ett optionspris med avseende på alla punkter i rutnätet eftersöks. Det andra exemplet beskriver fallet av en korgoption där känsligheter med avseende på alla underliggande aktier för optionen eftersöks. I det sista exemplet beskrivs ett fall där känsligheter av en ränteoption med avseende på alla initiala LIBOR-räntor eftersöks, här görs antagandet om en LIBOR Marknadsmodell.

Det visas att både *forward mode* och *adjoint mode* producerar resultat som är i linje med de värden som bestäms med hjälp av finita differensmetoden. Det visas även att användning av *adjoint mode*, i alla tre finansiella exempel, kan reducera den datorkraft som behövs i jämförelse med *forward mode* och finita differensmetoden.

Acknowledgements

We would like to thank Cinnober for giving us the opportunity to write our thesis for them. Also, an especial gratitude goes to our Supervisor at Cinnober, Magnus Sandström, who introduced us to this subject and also triggered a lot of interesting discussions connected to it. Finally we would like to sincerely thank our Supervisor at the Royal Institute of Technology, Thomas Önskog, for outstanding feedback and help throughout the project.

Stockholm, September 2015

Eric Sunnegårdh
Ludvig Lamm

Contents

Chapter 1

Background

The financial markets are an ever growing world spanning industry. A lot of effort is put into optimizing different aspects of it. In the following sections an introduction to financial contracts and some fundamentals of risk management will be given to clarify when computational power, in some cases, becomes a limiting factor.

1.1 Derivatives

In finance, a derivative is a contract which is dependent on one or more underlying assets. An underlying asset can for example be a stock, bond or commodity. A commonly traded type of derivative is an option. There are different kinds of options, one example is the European call option. This contract gives the buyer of the option the right, but not the obligation, to buy some underlying asset at a predetermined date and price specified in the contract. The predetermined date is generally referred to as maturity date or time of exercise and the predetermined price as strike price [9]. The payoff, $f(S_T)$, of such an option can mathematically be represented as,

$$f(S_T) = \max(S_T - K, 0) \tag{1.1}$$

where S_T is the value of the asset at maturity date T and K the strike price.

In order to enter an option contract, the buyer of the option has to pay a "premium" or price to the seller. There are various methods and models to determine the price of the option, some of which will be investigated more thoroughly later in this report. Though, common for all the methods is that

all strive to estimate the discounted expected payoff of the option, since this obviously would be a fair price [9]. Whenever referring to an expected value, if nothing else is said, the risk neutral measure is in mind and will be denoted by $E[(\dots)]$. The risk neutral measure is a probability measure having the property that every share price will be exactly the same as the discounted expectation of the share price when using this specific measure. This is also referred to as a martingale measure [10]. Hence, the price V of a European call option can be expressed as,

$$V = \exp(-rT) \cdot E[\max(S_T - K, 0)], \tag{1.2}$$

where r is the (constant) interest rate.

As already mentioned, apart from the European call option there are several other contracts that are exchange traded. A European put option differs from the call option in the way that the buyer of the option has the right, but not the obligation, to sell an underlying asset. There are also American options, which gives the buyer the possibility to exercise the option at any time between now and the maturity date. All these contracts are exchange traded and are often referred to as vanilla options [9].

In addition to the exchange traded contracts there are also over-the-counter contracts. These kind of contracts are usually traded by financial institutions and their clients. This opens up the possibility of entering very complex contracts, which sometimes are hard to evaluate by analytically means. In these cases simulation methods are widely used to estimate the values of the contracts. However, this can be very costly when speaking in means of computational power [2]. Commonly, option contracts traded over-the-counter are usually referred to as exotic options [9].

1.2 Stock price dynamics and the Monte Carlo method

When evaluating expectations, such as Equation (1.2), it is common to model the stock price dynamics as a stochastic differential equation (SDE) which has the following form,

$$dS(t) = a(S(t), t)dt + b(S(t), t)dW(t). \tag{1.3}$$

Here $a(S(t), t)$ is called the drift coefficient, $b(S(t), t)$ is called the diffusion coefficient and W is a Wiener process, also referred to as a Brownian motion. To clarify, Equation (1.3) is a short-hand expression for the following equation,

$$S(t) = S(0) + \int_0^t a(S(u), u)du + \int_0^t b(S(u), u)dW(u),$$

where $S(t) \in \mathbb{R}^{d_1}$ and $W(t) \in \mathbb{R}^{d_2}$, commonly d_1 is equal to d_2. This equation has the characteristics of the sum of an ordinary Lebesgue integral and an Itô integral [6]. In some cases the SDE can be explicitly integrated, as in the Black-Scholes model which uses the following scalar SDE,

$$dS(t) = rS(t)dt + \sigma S(t)dW(t),$$

where r is the risk free interest rate and σ is the volatility. This is a geometric Brownian motion which has the following solution,

$$S(T) = S(0) \exp\left((r - \frac{1}{2}\sigma^2)T + \sigma W(T)\right). \tag{1.4}$$

Using the expression above, an analytical expectation of $f(S(T))$ can be derived using the probability distribution of $W(T)$. Alternatively, a simulation method, called Monte Carlo simulation, can be used to determine the expectation. The Monto Carlo estimate can be denoted by,

$$\hat{V} = M^{-1} \sum_m f(S^m),$$

where S^m is determined using Equation (1.4) with independently sampled values from the probability distribution for $W(T)$ and M is the total number of simulations ran.

Generally one can not explicitly integrate the SDE, so the time interval $[0, T]$ is split into N steps where each step has the size $h = T/N$. Then the numerical approximation for each step is

$$\hat{S}_{n+1} = \hat{S}_n + a(\hat{S}_n, t_n)h + b(\hat{S}_n, t_n)\Delta W_n \tag{1.5}$$

where ΔW is a Wiener increment. These increments are independent and have a normal distribution with zero mean and a variance equal to the time

step h. This means that for each of the M simulated paths, N independent samples are generated to produce one $\hat{S}_N$. The variance for the Monte Carlo simulation is $M^{-1}Var[f(S)]$ [6] and since the samples are independent, the error term is proportional to $\mathcal{O}(M^{-1/2})$.

1.3 Risk management and hedging

To hedge is to invest in a position, i.e. buy or sell contracts or assets, which intent to eliminate risk, usually by reducing possible future profits. There are numerous hedging techniques used in different situations in the financial market. A farmer can for example hedge his future profit by entering a forward contract giving him the right, and obligation, to sell a certain amount of crop to a certain price at a certain time in the future. This way he eliminates the risk associated with the crop dropping in price until the crop is ready to be delivered [1].

Hedging is also widely used by financial institutions to manage their risk incurred due to different investments. Sometimes the institutions want to hedge themselves against interest rate risk incurred from portfolios consisting of some assets and liabilities. The liability can for example be an issued bond, i.e. a short position in a bond (the opposite of a short position is a long position, which means buying a contract or asset). A commonly used method in this case is the portfolio immunization strategy. The strategy basically consists of ensuring that the average duration of the assets is equal to one of the liabilities (duration is a measure of how long on average the holder of a bond has to wait before receiving cash payments) [9]. Using this strategy the profit (or loss) in the assets will offset the loss (or profit) in the liabilities, as long as the interest rate is only affected by a parallel shift.

Another common situation when financial institutions are faced with risk is when they issue over-the-counter contracts. Whenever these kind of contracts are issued some kind of hedging is essential to keep the risk acceptable. This is often done with dynamic hedging, i.e. adjusting the hedge on a regular basis. In this case "a regular basis" is vague but commonly means somewhere between once a day to once every month. The sensitivities of the contract price, or the Greeks as they are often referred to, are used to determine how each adjustment should be carried out. For example, the risk associated with holding a short position in an option can be hedged using a

specific dynamic hedging technique called "delta-hedging", meaning buying "delta" units of the underlying asset. Here, delta is the price sensitivity with respect to the current price of the underlying asset, i.e.

$$\Delta = \frac{\partial V}{\partial S_0},$$

where S_0 is the current stock price and V is the option price [7].

In some cases closed form expressions can be found for the sensitivities, although when this is not the case simulation techniques are commonly used. This however, can be very demanding in terms of computational power [5].

1.4 Sensitivities

In addition to the sensitivity "delta" presented in the previous section, there are other first order price sensitivities. The following list presents a few of them,

- Vega (ν) - the derivative of the option price with respect to the volatility of the underlying asset, i.e. $\nu = \frac{\partial V}{\partial \sigma}$.
- Theta (Θ) - the derivative of the option price with respect to the passage of time, i.e. $\Theta = \frac{\partial V}{\partial \tau}$. Here the passage of time is equivalent to the time elapsed since the option was issued.
- Rho (ρ) - the derivative of the option price with respect to the risk free interest rate, i.e. $\rho = \frac{\partial V}{\partial r}$.

In some situation it can be beneficial to calculate second order price sensitivities, one example is Gamma, which is the second derivative with respect to the current stock price, i.e.

$$\Gamma = \frac{\partial^2 V}{\partial S_0^2}.$$

Whenever these sensitivities are necessary to calculate and no closed form expression can be formulated, the Monte Carlo simulation method is commonly used. The following section will present three different approaches combined with the Monte Carlo simulation.

Probably the most intuitive approach is to apply a finite difference approximation,

$$\frac{\partial V}{\partial \theta} \approx \frac{V(\theta + \Delta\theta) - V(\theta - \Delta\theta)}{2\Delta\theta},$$

for the first order derivative and,

$$\frac{\partial V^2}{\partial \theta^2} \approx \frac{V(\theta + \Delta\theta) - 2V(\theta) + V(\theta - \Delta\theta)}{(\Delta\theta)^2},$$

for the second order derivative. Here V represents the option price and θ an input parameter (e.g. current stock price, risk free interest rate or volatility). The main shortcoming with this approach is that it requires two additional sets of Monte Carlo simulations for each input parameter θ, i.e. the computational cost increases proportional to the number of input parameters and can therefore become large. Also great care must be taken when choosing the size of $\Delta\theta$ since a too large value will make the finite difference approximation error significant, and a too small value can make the variance become large if the payoff function of the option is discontinuous [7].

Another commonly used method is the Likelihood ratio method. Generally, this approach can be used whenever a probability distribution can be determined for the underlying. Assume that the value of a derivative can be expressed as,

$$E[f(S)] = \int_{\mathbb{R}} f(S) \cdot p_\theta(S) dS$$

where S is the underlying, f the payout function and p_θ the probability density of the underlying which depend on the parameters θ. If differentiating with respect to θ and then assuming f to be Lipschitz continuous and the process S to be smooth, the order of integration and differentiation can be interchanged [11]. Thus, the following can then be obtained,

$$\begin{aligned}\frac{\partial E[f(S)]}{\partial \theta} &= \int_{\mathbb{R}} f(S) \cdot \frac{\partial p_\theta(S)}{\partial \theta} dS = \int_{\mathbb{R}} f(S) \frac{\partial log p_\theta(S)}{\partial \theta} p_\theta(S) dS \\ &= E\left[f(S) \frac{\partial log p_\theta(S)}{\partial \theta} \right]\end{aligned}$$

where the final expectation can be evaluated analytically, or if that is not possible, numerical methods can be used. This method is beneficial when dealing with discontinuous payoff functions since these will not have to be differentiated. Although as stated earlier, this requires that a probability

density and the derivative of the probability density can be obtained. This however, can be complicated. Also, this method does not generalise well when it comes to path calculations involving multiple small timesteps since this often leads to an estimator with a high variance [7]. This situation arises when evaluating options with path dependent payoffs, for example American options.

For the third approach called the Pathwise derivative method, assume that the value of a derivative can be expressed as,

$$E[f(S)] = \int_{\mathbb{R}} f(S(T)) \cdot p_W(W) dW$$

where $p_W(W)$ is the probability density function for $W(T)$ given by,

$$p_W(W) = \frac{1}{\sqrt{2\pi T}} \exp(-\frac{W^2}{2T})$$

Notice that this also can be obtained from Equation (1.4) by performing a change of variables [6]. Also notice that in Equation (1.4) the parameters in θ enter the integral through the probability density of $S(T)$ which is given by,

$$p_\theta(S) = \left(\frac{\partial S}{\partial W}\right)^{-1} p_W = \frac{1}{S\sigma\sqrt{2\pi T}} \exp\left(-\frac{1}{2}\left(\frac{\log\left(\frac{S}{S_0}\right) - \left(r - \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}}\right)^2\right)$$

but in the second case the parameters enter through the expression of $S(T)$ given in Equation (1.4). Now, as in the case of the Likelihood ratio method, this third approach uses the procedure of interchanging the expectation and derivative, this gives,

$$\frac{\partial E[f(S)]}{\partial \theta} = \int \frac{\partial f(S)}{\partial S(T)} \frac{\partial S(T)}{\partial \theta} p_W dW = E\left[\frac{\partial f(S)}{\partial S(T)} \frac{\partial S(T)}{\partial \theta}\right].$$

From here on the Monte Carlo method described in section 1.2 can be used to estimate the expectation, i.e.

$$\frac{\partial \hat{V}}{\partial \theta} = M^{-1} \sum_m \frac{\partial f}{\partial S}(\hat{S}_N^{(m)}) \frac{\partial \hat{S}_N^{(m)}}{\partial \theta}$$

where $\hat{S}_N^{(m)}$ is determined using the Euler scheme discretisation in Equation (1.5) for every path m. If differentiating the Euler scheme, the following is obtained,

$$\frac{\partial \hat{S}_{n+1}}{\partial \theta} = (1 + \frac{\partial a}{\partial S}h + \frac{\partial b}{\partial S}\Delta W_n)\frac{\partial \hat{S}_n}{\partial \theta} + \frac{\partial a}{\partial \theta}h + \frac{\partial b}{\partial \theta}\Delta W_n, \tag{1.6}$$

which then can be used to step by step determine the sensitivity of the path to changes in the input parameters θ and finally determining a value of $\frac{\partial \hat{S}_N^{(m)}}{\partial \theta}$ for every path.

As mentioned, this last method described, is called the pathwise derivative method. A benefit, compared to the likelihood ratio method, is that it generalises well to path calculations [6]. But the major drawback with this approach is that the payout function has to be differentiable. However, when this is the case and multiple sensitivities are required there exists an efficient implementation for the pathwise derivative method [5]. This implementation is generally referred to as algorithmic differentiation and comes in two settings, the forward mode and the adjoint mode, these settings will be described in the following chapter.

Chapter 2

Algorithmic differentiation

In this chapter an introduction to Algorithmic differentiation (AD), also called Automatic differentiation, is given. This is then followed by a simple example with numerical results to clarify the theory. The final section gives some introduction to available tools performing these calculations automatically.

2.1 Background and notations

Given a function in the form of a computer program, Automatic differentiation is a set of techniques to rapidly and with high precision compute partial derivatives to this function. The technique relies on the fact that any computer program basically is a sequence of elementary functions such as exponential functions or logarithms, combined by elementary operations such as addition or subtraction. Hence, by using the chain rule of differentiation, derivatives of all orders can be computed to just a small additional cost of evaluating the original function itself. Using these facts, computer programs can be generated automatically to accurately and efficiently evaluate derivatives of an arbitrary function [8]. Algorithmic differentiation basically comes in two settings, namely, forward and reverse mode. Given a function $f(x) = g(h(x))$ the chain rule yields

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial h}\frac{\partial h}{\partial x}.$$

Using the first setting, forward mode, the chain of derivatives will be evaluated from the inside out, i.e. first $\frac{\partial h}{\partial x}$ should be determined and then $\frac{\partial g}{\partial h}$.

In the second setting, reverse mode (also referred to as adjoint mode), the derivatives should be evaluated from the outside to the inside, i.e. $\frac{\partial g}{\partial h}$ first and then $\frac{\partial h}{\partial x}$.

Consider the scalar function $P = f(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha}$ is an input vector. This function calculates an intermediate vector $\mathbf{S}$, which values are used to compute the final P value. This function will be used as an example to explain the notations used for both methods. The notations used in this report for automatic differentiation will be the same as in the researching community. Forward mode uses the dot notation, for example $\dot{\mathbf{S}}$ means the derivative of $\mathbf{S}$ with respect to one specific input parameter in $\boldsymbol{\alpha}$. Hence, using the chain rule, $\dot{\mathbf{S}}$ can be expressed as

$$\dot{\mathbf{S}} = \frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}} \dot{\boldsymbol{\alpha}},$$

here, $\frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}}$ is the Jacobian of $\mathbf{S}$ as a function of $\boldsymbol{\alpha}$. Further on, if following the dot notation, $\dot{\boldsymbol{\alpha}}$ is the derivative of the input vector with respect to one of its elements α_i, i.e.

$$\dot{\boldsymbol{\alpha}} = \frac{\partial \boldsymbol{\alpha}}{\partial \alpha_i} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^T.$$

where the 1 has position i in the resulting vector. For example, if the derivative of P with respect to a specific input parameter is sought, where $\mathbf{S}$ is an intermediate step, $\dot{P}$ can be expressed as follows,

$$\dot{P} = \frac{\partial P}{\partial \mathbf{S}} \dot{\mathbf{S}},$$

here, $\frac{\partial P}{\partial \mathbf{S}}$ is a row vector. Using the chain rule this can be expanded to,

$$\dot{P} = \frac{\partial P}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}} \dot{\boldsymbol{\alpha}}.$$

Notice that here the calculations are executed in the following order,

$$\dot{\boldsymbol{\alpha}} \rightarrow \dot{\mathbf{S}} \rightarrow \dot{P}. \tag{2.1}$$

Reverse mode uses bar notation, for example $\bar{\mathbf{S}}$ means the derivatives of P with respect to $\mathbf{S}$ and if using the chain rule this can be expanded to

$$\bar{\mathbf{S}} = \left(\frac{\partial P}{\partial \mathbf{S}} \right)^T \bar{P}$$

where ^{T} denotes either a matrix or vector transpose. Also, following the bar notation, $\bar{P} = \frac{\partial P}{\partial P} = 1$ is given by definition. Now, the derivative of P with respect to the input vector $\boldsymbol{\alpha}$ is denoted by $\bar{\boldsymbol{\alpha}} = \left(\frac{\partial P}{\partial \boldsymbol{\alpha}}\right)^T$. Expanding this, the following expression for $\bar{\boldsymbol{\alpha}}$ is obtained

$$\bar{\boldsymbol{\alpha}} = \left(\frac{\partial P}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}}\right)^T = \left(\frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}}\right)^T \bar{\mathbf{S}},$$

and using the bar definition for $\bar{S}$,

$$\bar{\boldsymbol{\alpha}} = \left(\frac{\partial \mathbf{S}}{\partial \boldsymbol{\alpha}}\right)^T \left(\frac{\partial P}{\partial \mathbf{S}}\right)^T \bar{P}. \tag{2.2}$$

The returned $\bar{\boldsymbol{\alpha}}$ now contains derivatives with respect to all elements of the input vector. Notice, that these calculations are carried out in the reverse order, compared to forward mode, starting with $\bar{P}$,

$$\bar{P} \rightarrow \bar{\mathbf{S}} \rightarrow \bar{\alpha}.$$

For the current function $P = f(\boldsymbol{\alpha})$, which only has one output and multiple inputs, the reverse mode is the most efficient method when sensitivities with respect to all input parameters are desired. This is due to the fact that performing the calculations in Equation (2.2) results in a vector containing all desired sensitivities while performing the calculations in Equation (2.1) only one sensitivity is determined. Hence, if N sensitivities are desired, Equation (2.1) has to be evaluated N times making the forward mode N times more costly than the adjoint mode.

2.2 Forward and reverse mode

Consider the scalar function $F(\mathbf{X})$, then the computation of the sensitivities, using the forward notation, looks like the following

$$\dot{F} = \frac{\mathfrak{d}F}{\mathfrak{d}\mathbf{X}} \dot{\boldsymbol{X}} = \frac{\mathfrak{d}F}{\mathfrak{d}X_1}\dot{X}_1 + ... + \frac{\mathfrak{d}F}{\mathfrak{d}X_N}\dot{X}_N,$$

the symbol $\mathfrak{d}$ denoted a derivative where no implicit dependencies are considered. Following the researching community's practice, it should be represented as a matrix equation. This can be done by including the trivial

equations $\dot{X}_1 = \dot{X}_1, ..., \dot{X}_N = \dot{X}_N$, yielding

$$\begin{bmatrix} \dot{X}_1 \\ \vdots \\ \dot{X}_N \\ \dot{F} \end{bmatrix} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ \frac{\partial F}{\partial X_1} & \cdots & \frac{\partial F}{\partial X_N} \end{bmatrix} \begin{bmatrix} \dot{X}_1 \\ \vdots \\ \dot{X}_N \end{bmatrix}.$$

Notice that the empty entries in the matrix represents zeros. This matrix is usually denoted by D and has the dimensions $(N+1) \times N$, where N is the length of the input vector. To compute all the derivatives for the input vector $\mathbf{X}$, the matrix equation has to be evaluated N times. Each time with an updated input vector $\dot{\mathbf{X}}$, for example, $\dot{\mathbf{X}} = (1, 0, ..., 0)^T$ will result in $\frac{\partial F}{\partial X_1}$ and $\dot{\mathbf{X}} = (0, ..., 0, 1)^T$ will result in $\frac{\partial F}{\partial X_N}$.

Now consider the case of reverse mode, here the quantities $\bar{\boldsymbol{X}}_{\boldsymbol{i}}$ are sought. Using the notation for adjoint variables these can be expanded to the following,

$$\bar{\boldsymbol{X}}_{\boldsymbol{i}} = \frac{\partial F}{\partial X_i} \bar{F}.$$

Now these equations can be written as a matrix equation having the following appearance,

$$\begin{bmatrix} \bar{X}_1^{(0)} \\ \vdots \\ \bar{X}_N^{(0)} \end{bmatrix} = \begin{bmatrix} 1 & & \frac{\partial F}{\partial X_1} \\ & \ddots & \vdots \\ & 1 & \frac{\partial F}{\partial X_N} \end{bmatrix} \begin{bmatrix} \bar{X}_1^{(1)} \\ \vdots \\ \bar{X}_N^{(1)} \\ \bar{F} \end{bmatrix}$$

assuming all bar values in the input vector, $\begin{bmatrix} \bar{X}_1^{(1)} & \cdots & \bar{X}_N^{(1)} & \bar{F} \end{bmatrix}^T$ are equal to zero except for $\bar{F} = \frac{\partial F}{\partial F} = 1$. The additional index introduced here denotes that the values in the input vector are being updated. This way, the sensitivities of F with respect to all input parameters are calculated in one sweep. Notice that the matrix in the adjoint calculation is just the transpose of the one in forward mode.

The previous example is a trivial case, in a more general setting intermediate variables and their sensitivities have to be calculated. Thus, assume that $\mathbf{X_n} = F^n(\mathbf{X_{n-1}})$ gives an intermediate value in the process of calculating the final vector $\mathbf{X_N} = F^N(\mathbf{X_{N-1}})$ and that,

$$F^n(\mathbf{X_{n-1}}) \equiv \begin{pmatrix} \mathbf{X_{n-1}} \\ f_n(\mathbf{X_{n-1}}) \end{pmatrix} \tag{2.3}$$

where f_n is a scalar function and $\mathbf{X_{n-1}}$ is a column vector. Notice that each time the function F^n is applied to the column vector X_n, it increases the vectors length by one additional row.

Now differentiating Equation (2.3), and defining an intermediate vector of derivative as $\dot{\mathbf{X}}_\mathbf{n}$, gives the following relation,

$$\dot{\mathbf{X}}_\mathbf{n} = D^n \dot{\mathbf{X}}_\mathbf{n-1}$$

and

$$D^n = \begin{pmatrix} I^{n-1} \\ \frac{\partial f_n}{\partial \mathbf{X_{n-1}}} \end{pmatrix}$$

where I^{n-1} represents the identity matrix with dimensions $(k \times k)$ where k is equal to the length of $\mathbf{X_{n-1}}$ and $\frac{\partial f_n}{\partial \mathbf{X_{n-1}}}$ corresponds to a row vector containing all partial derivatives of the currents step scalar function f_n. So for each step, evaluating intermediate sensitivities $\dot{\mathbf{X}}_\mathbf{n}$, each new D matrix will expand by one additional row and column compared to the D matrix of the previous step. The final vector of sensitivities $\dot{\mathbf{X}}_\mathbf{N}$ can then be computed as

$$\dot{\mathbf{X}}_\mathbf{N} = D^N \cdot D^{N-1} \cdot ... \cdot D^1 \cdot \dot{\mathbf{X}}_\mathbf{0}.$$

Here, $\dot{\mathbf{X}}_\mathbf{N}$ now contains sensitivities of the entire output vector $\mathbf{X_N}$ with respect to one element in the input vector X_0. Usually, in financial applications, only the last element of $\dot{\mathbf{X}}_\mathbf{N}$ will be of interest (this will be explained more thoroughly later).

In the case of reverse mode, in this more general setting, consider the column vector $\bar{\mathbf{X}}_\mathbf{n}$ denoting the derivative of a specific element in the output vector, for example $X_N^{(i)}$, with respect to all elements in the intermediate vector $\mathbf{X_n}$. Using the chain rule, the following can be obtained,

$$\bar{\mathbf{X}}_\mathbf{n-1} = \left(\frac{\partial X_N^{(i)}}{\partial \mathbf{X_{n-1}}} \right)^T = \left(\frac{\partial X_N^{(i)}}{\partial \mathbf{X_n}} \frac{\partial \mathbf{X_n}}{\partial \mathbf{X_{n-1}}} \right)^T = \left(\frac{\partial \mathbf{X_n}}{\partial \mathbf{X_{n-1}}} \right)^T \left(\frac{\partial X_N^{(i)}}{\partial \mathbf{X_n}} \right)^T$$
$$= (D^n)^T \bar{\mathbf{X}}_\mathbf{n}.$$

hence, this reveals the relation $\bar{\mathbf{X}}_\mathbf{n-1} = (D^n)^T \bar{\mathbf{X}}_\mathbf{n}$. Now, using this relation,

the following expression will give the sensitivity of one specific output element $X_N^{(i)}$ with respect to all elements of in the input vector $\mathbf{X_0}$,

$$\bar{\mathbf{X}}_\mathbf{0} = (D^1)^T \cdot ... \cdot (D^{N-1})^T \cdot (D^N)^T \cdot \bar{\mathbf{X}}_\mathbf{N}.$$

where $\bar{\mathbf{X}}_\mathbf{0}$ now contains sensitivities with respect to all input parameters $\mathbf{X_0}$. Note that to perform the reverse mode calculations an initial forward sweep is necessary to calculate and store all D matrices. Also, as in the previous simpler case the reverse mode requires only one sweep (corresponding to carrying out the N matrix multiplications of the D matrices) to determine sensitivities with respect to all input parameters, while the forward mode requires as many sweeps as sensitivities desired.

2.3 Basic implementation example

Consider the step function

$$a_{n+1} = \exp(0.05 \cdot a_n + b) \tag{2.4}$$

and assume that it is desired to calculate the sensitivity of a_2 with respect to a_0 and b (which are considered to be known input parameters). To clarify the theory explained in previous section these calculations will be carried out using both the forward mode and backward mode. Notice that the same intermediate variable a_1 is used in both methods. According to the step function in (2.4), a_1 can be expressed as,

$$a_1 = \exp(0.05 \cdot a_0 + b). \tag{2.5}$$

2.3.1 Forward implementation

To initiate the forward mode, the sensitivity of the intermediate variable is expressed as,

$$\dot{a}_1 = \frac{\mathfrak{d}a_1}{\mathfrak{d}a_0}\dot{a}_0 + \frac{\mathfrak{d}a_1}{\mathfrak{d}b}\dot{b}, \tag{2.6}$$

where the partial derivatives can be determined using Equation (2.5),

$$\frac{\mathfrak{d}a_1}{\mathfrak{d}a_0} = 0.05 \exp(0.05 \cdot a_0 + b)$$

$$\frac{\partial a_1}{\partial b} = \exp(0.05 \cdot a_0 + b)$$

If inserting the partial derivatives in Equation (2.6) and also introducing the trivial equations $\dot{a}_0 = \dot{a}_0$ and $\dot{b} = \dot{b}$, then this system of equation can be expressed using a matrix notation,

$$\begin{bmatrix} \dot{a}_0 \\ \dot{b} \\ \dot{a}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0.05 \cdot \exp(0.05 \cdot a_0 + b) & \exp(0.05 \cdot a_0 + b) \end{bmatrix} \begin{bmatrix} \dot{a}_0 \\ \dot{b} \end{bmatrix}.$$

In a similar way, the system of equation of the next iteration to obtain $\dot{a}_2$, can be expressed as

$$\begin{bmatrix} \dot{a}_0 \\ \dot{b} \\ \dot{a}_1 \\ \dot{a}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & \exp(0.05 \cdot a_1 + b) & 0.05 \cdot \exp(0.05 \cdot a_1 + b) \end{bmatrix} \begin{bmatrix} \dot{a}_0 \\ \dot{b} \\ \dot{a}_1 \end{bmatrix},$$

where $\dot{a}_2$ is the sensitivity with respect to one element in the initial vector $\begin{bmatrix} \dot{a}_0 \\ \dot{b} \end{bmatrix}$. As mentioned earlier, to determine the sensitivity with respect to the *k-th* element of the input vector, the above expressions has to be evaluated for the initial vector having the *k-th* element set to one and all others set to zero. Hence, in this case the forward mode calculations have to be done twice (since sensitivities with respect to all elements were of interest), the first time with the initial vector as $\begin{bmatrix} \dot{a}_0 \\ \dot{b} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and the second time as $\begin{bmatrix} \dot{a}_0 \\ \dot{b} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

2.3.2 Adjoint implementation

In the adjoint mode, the calculations are made in the reverse order. Notice that the matrices containing the directional derivatives derived when applying the forward mode will still be very useful, they will just be transposed. Starting from the back,

$$\begin{bmatrix} \bar{a}_0^{(1)} \\ \bar{b}^{(1)} \\ \bar{a}_1^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \exp(0.05 \cdot a_1 + b) \\ 0 & 0 & 1 & 0.05 \cdot \exp(0.05 \cdot a_1 + b) \end{bmatrix} \begin{bmatrix} \bar{a}_0^{(2)} \\ \bar{b}^{(2)} \\ \bar{a}_1^{(2)} \\ \bar{a}_2^{(2)} \end{bmatrix},$$

where $\bar{a}_2 = \frac{\partial a_2}{\partial a_2} = 1$ and the adjoint variables $\bar{a}_0^{(2)}$, $\bar{b}^{(2)}$ and $\bar{a}_1^{(2)}$ are initially set to zero. The next backward iteration yields the desired sensitivities,

$$\begin{bmatrix} \bar{a}_0^{(0)} \\ \bar{b}^{(0)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.05 \cdot \exp(0.05 \cdot a_0 + b) \\ 0 & 1 & \exp(0.05 \cdot a_0 + b) \end{bmatrix} \begin{bmatrix} \bar{a}_0^{(1)} \\ \bar{b}^{(1)} \\ \bar{a}_1^{(1)} \end{bmatrix}.$$

Hence, the final vector $\begin{bmatrix} \bar{a}_0^{(0)} \\ \bar{b}^{(0)} \end{bmatrix}$ now contains the the sensitivity of a_2 with respect to a_0 and b. Notice that the adjoint calculations only had to be done once to evaluate both sensitivities, in comparison to the forward mode where two sweeps were necessary.

2.3.3 Verifying the result

To ensure the reader that the forward and adjoint mode gives identical results and also aligns with results determined using a finite difference approach, Table 2.1 presents numerical results from all three methods of the previous example.

Table 2.1: *Numerical results of sensitivities calculated by finite difference, forward mode and adjoint mode.*

	Finite difference	Forward	Adjoint
$\frac{\partial a_2}{\partial a_0}$	0.02240258	0.02240252	0.02240252
$\frac{\partial a_2}{\partial b}$	3.58407199	3.58384478	3.58384478

Note that the number of significant figures in the table is way to large considering the number of significant figures of the model and are not to be trusted, but these are kept to show that the results of the adjoint and forward mode

indeed are identical. The reason why they are identical is because of the fact that the exactly same computations are carried out in both modes, the only difference is the order the operations are done. Also note that the results in Table 2.1 are based on the input parameters chosen as $a_0 = 1$ and $b = 1$ and the finite difference scheme used was of first order, i.e.

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

where $h = 10^{-6}$.

2.4 Tools

Implementing Algorithmic differentiation manually can be a quite mundane and repetitive work, therefore there is a high risk of errors, but this can be avoided by instead using Algorithmic differentiation tools. Currently, there are two categories of AD tools, source code transformation and operator overloading. Source code transformation tools uses the existing source code as input and generates new code which will either perform a forward or adjoint calculation. The other method, operator overloading, only works for languages which support it. C++ is one of these languages and to implement AD in C++ with operator overloading firstly new variable types have to be created. For example, $float$ becomes $ADfloat$. This new AD type will have two properties, one is a list of all the values the variable has ever had, the other is a list of all the derivatives the variable has ever had. This way the program can trace how the variable has changed through out the program. Now operators such as $*$ and $+$ will not work for the new types. This is where operator overloading becomes useful. In languages such as C++, the compiler can be told how to handle operations between custom types, so for each operator this new custom behavior has to be implemented, where the current value and its derivative are saved in a list. There will be no difference in the code except for the new types and that they will require some extra memory, the operators such as $*$ should behave as before. An advantage with the source code transformation tool is that it has a greater control of how the source code is generated, it can perform some optimization which operator overloading cannot [2]. These AD tools are still quite new and not a lot of languages are supported, anything other than C, C++ or FORTRAN

can be hard to find.

No further investigation of these tools will be presented in this report, instead the focus will be on the mathematical side of it and how Algorithmic differentiation can be implemented in some real financial applications. Still, it is important to mention that these tools exist and are probable to become useful in the financial industry.

Chapter 3

Implementing Algorithmic differentiation in finance

In this chapter the methods described in the previous chapter will be implemented on three common financial cases. The first section covers the fundamentals of a volatility surface and how the previously presented theory can be used to determine sensitivities of an option price with respect to all nodes on that volatility surface. The following section investigates the case of a basket option where sensitivities with respect to all underlying stocks are desired. This is followed by the last section covering the case of a caplet where the sensitivities with respect to different forward rates are sought, using the assumptions of a LIBOR Market Model.

3.1 Volatility Surface

The Black-Scholes framework for option pricing can in some senses be considered unrealistic. One reason is the assumption that the volatility is constant. It might make more sense to assume that the volatility can be modeled as a random variable, these models exists and are usually referred to as stochastic volatility models. But the computational complexity of these models and the procedure of fitting model parameters can be very difficult [4]. Instead, Bruno Dupire showed how option prices quoted in the market could be used to calculate a state-dependent volatility $\sigma(S_t, t)$. If assuming that the underlying stock pays zero dividend and following Dupire's approach, the formula for calculating the local volatility can be expressed as,

$$\sigma(S,t;S_0)_{|S=K,t=T} = \sqrt{\frac{2\left(\frac{\partial C(S_0,T,K)}{\partial T} + rK\frac{\partial C(S_0,T,K)}{\partial K}\right)}{K^2\frac{\partial^2 C(S_0,T,K)}{\partial K^2}}} \tag{3.1}$$

where C represents an option price quoted in the market with strike price K and time to maturity T [3]. Note that the derivatives in the Formula (3.1) can be estimated using market data and forming a finite difference. If this is done multiple times for different strike prices and maturities of an option, a local volatility surface can be obtained interpolating the retrieved volatilities. Now, this volatility surface can be used when simulating paths of the option. The dynamics of the process will have the following appearance,

$$dS_t = rS_t dt + \sigma(S_t,t)dW_t. \tag{3.2}$$

where $\sigma(S_t,t)$ is the state dependent volatility and r is the constant rate.

A problem with the Dupire formula (3.1) is that when the second derivative of the option price with respect to the strike price approaches zero the whole denominator approaches zero. Also, a high level of instability can be introduced when applying, for example, a finite difference approach to the option prices to determine their derivatives since there might not be as much available data as would be desired. How unstable the results become can vary a lot depending on which techniques that are used and the amount of data available when evaluating the derivatives. However, whatever method used, it will be interesting evaluating how much an error in each node (obtained from the Dupire formula) will influence the pricing of an option using the dynamics of Formula (3.2). Now, if considering the adjoint method of Algorithmic differentiation presented in the previous chapter, it seems likely that great computational savings can be made for this scenario since sensitivities of one output (the option price) with respect to multiple input parameters (all sensitivity nodes) are desired.

In the following sections, the calculations required to determine sensitivity of an option price with respect to all of the volatility nodes are shown for both forward and adjoint mode. This is then followed by numerical results on the computation times of both methods.

3.1.1 Preliminaries

For this example the price of an European call option will be in focus. This option has the payoff shown in Equation (1.1) and the price of the option is shown in Equation (1.2). Here, the expectation can be evaluated using the dynamics of S_t specified in Equation (3.2). Although, the dynamics of S_t will be discretized by using an Euler Scheme as described in section 1.2, but with the rate constant. Hence, the discretized scheme will have the following appearance,

$$S_{n+1} = S_n + rS_n\Delta t + \sigma(S_n, t_n)S_n\Delta W_n. \tag{3.3}$$

Further on, it is assumed that a grid of local volatilities is given, denoted by $\sigma_{i,j}$ where the indices $i = 1 \dots I$ and $j = 1 \dots J$. Also, i and j corresponds to certain times T_i and stock prices S_j^* respectively, notice that the super-index * indicates the belonging to a certain grid point. So by definition, $\sigma_{i,j} = \sigma(S_j^*, T_i)$. Notice that in Equation (3.3) the volatility depends on both S_n and t_n and that these points does not necessarily correspond to a given grid point. Thus, the given grid points have to be interpolated, both in time and stock price, creating a continuous surface of local volatilities. The equations used for time interpolation is the following,

$$\sigma_j^n = \sigma_{i,j} + (t_n - T_i)\frac{\sigma_{i+1,j} - \sigma_{i,j}}{T_{i+1} - T_i} \tag{3.4}$$

where σ_j^n then denotes a value on the local volatility surface that lies between two grid points stock wise, but lies at a time t_n determined by the Euler scheme. It should be noted that the i-index now is suppressed since the n-index states where in time this volatility should be used. The corresponding interpolation relationship stockwise is,

$$\sigma_n = \sigma_j^n + (S_n - S_j^*)\frac{\sigma_{j+1}^n - \sigma_j^n}{S_{j+1}^* - S_j^*}. \tag{3.5}$$

Here $\sigma_n = \sigma(S_n, t_n)$ and thus denote a local volatility at a certain point (S_n, t_n) obtained through the Euler scheme. Notice that for each σ_j^n to be determined, two grid points are needed and that for each σ_n both σ_j^n and σ_{j+1}^n are needed. Hence, it requires four nodes to calculate one σ_n. Here, it should be noted that also the j-index is suppressed since the volatility does not correspond to a certain grid point, but instead the n-index indicates the

value of the stock price, which is S_n.

The fact that the Euler Scheme allows for S_n to take values outside the boundaries of the volatility grid creates some problems. To handle this, it is assumed that whenever S_n jumps outside the grid the volatility is constant, i.e. if $S_n > S^*_{max}$ then $\sigma(S_n, t_n) = \sigma(S^*_{max}, t_n) = \sigma^n_J$ and if $S_n < S^*_{min}$ then $\sigma(S_n, t_n) = \sigma(S^*_{min}, t_n) = \sigma^n_1$. Here S^*_{min} and S^*_{max} corresponds to the boundary points on the volatility grid, stockwise. Although it is assumed that the given grid of volatilities is large, i.e. it contains grid points corresponding to stock prices about three standard deviations away from the current price. Hence, it is unlikely that the stock price will ever fall outside the grid and therefore only a few of the trajectories will contain such values, making the above restriction more justified.

3.1.2 Forward implementation

Looking back at the theory in chapter 2, the quantities sought here are denoted by $\frac{\partial P}{\partial \sigma_{i,j}} = \dot{P}$ for $i = 1 \dots I$ and $j = 1 \dots J$. To begin, consider the derivative of P with respect to one of the nodes $\sigma_{i,j}$. The input vector, initiating the computations of the derivatives will be,

$$\mathbf{x_0} = \left[\dot{S}_0 \ \dot{\sigma}_{1,1} \ \cdots \ \dot{\sigma}_{1,J} \ \dot{\sigma}_{2,1} \ \cdots \ \dot{\sigma}_{2,J} \ \cdots \ \dot{\sigma}_{i,j} \ \cdots \ \dot{\sigma}_{I,1} \ \cdots \ \dot{\sigma}_{I,J} \right]^T.$$

Following the notations it is obvious that $\dot{\sigma}_{i,j} = \frac{\partial \sigma_{i,j}}{\partial \sigma_{i,j}} = 1$ and that all other derivatives with respect to $\sigma_{i,j}$ equals zero. Since there are multiple steps of evaluating $P = \exp(-rT)\max(S_N - K, 0)$ there will also be multiple steps evaluating the intermediate derivatives. Here P is the payoff function of the call option times the discount factor from the pricing expression of Equation 1.2. Considering the definition of P and Equations (3.3),(3.4) and (3.5), the first step of evaluating P is to use the interpolation relation (3.4) and determine σ^0_j. Hence the first step now, evaluating the intermediate derivatives, is to evaluate the derivative of σ^0_j with respect to all input parameters. The D^1 matrix corresponding to this have the following appearance,

$$\begin{bmatrix} & & & & & & \\ & & & \mathbb{1} & & & \\ & & & & & & \\ \frac{\partial \sigma_j^0}{\partial S_0} & \frac{\partial \sigma_j^0}{\partial \sigma_{1,1}} & \cdots & \frac{\partial \sigma_j^0}{\partial \sigma_{i,j}} & \cdots & \frac{\partial \sigma_j^0}{\partial \sigma_{I,J}} \end{bmatrix}$$

here the $\mathbb{1}$ denotes the identity matrix. Performing this matrix and vector multiplication $D^1\mathbf{x_0}$ the output vector now contains the following values,

$$\begin{bmatrix} \dot{S}_0 & \dot{\sigma}_{1,1} & \cdots & \dot{\sigma}_{i,j} & \cdots & \dot{\sigma}_{I,J} & \dot{\sigma}_j^0 \end{bmatrix}^T.$$

Notice here that only the quantities denoted by $\frac{\partial \sigma_j^0}{\partial \sigma_{i,j}}$ and $\frac{\partial \sigma_j^0}{\partial \sigma_{i+1,j}}$ in the matrix D^1 take values separated from zero. Computing these values gives,

$$\frac{\partial \sigma_j^0}{\partial \sigma_{i,j}} = 1 + (t_0 - T_i)\frac{0-1}{T_{i+1} - T_i} = \frac{T_{i+1} - t_0}{T_{i+1} - T_i} \tag{3.6}$$

and

$$\frac{\partial \sigma_j^0}{\partial \sigma_{i+1,j}} = 0 + (t_0 - T_i)\frac{1-0}{T_{i+1} - T_i} = \frac{t_0 - T_i}{T_{i+1} - T_i}$$

The second step of evaluating P is to find σ_{j+1}^0, which means that the second step now is to evaluate the derivative of σ_{j+1}^0 with respect to all input parameters. This is very similar to the previous step and the matrix D^2 corresponding to these calculations have the following appearance,

$$\begin{bmatrix} & & & & & & \\ & & & \mathbb{1} & & & \\ & & & & & & \\ \frac{\partial \sigma_{j+1}^0}{\partial S_0} & \frac{\partial \sigma_{j+1}^0}{\partial \sigma_{1,1}} & \cdots & \frac{\partial \sigma_{j+1}^0}{\partial \sigma_{i,j}} & \cdots & \frac{\partial \sigma_{j+1}^0}{\partial \sigma_{I,J}} & \frac{\partial \sigma_{j+1}^0}{\partial \sigma_j^0} \end{bmatrix}$$

and multiplying this with the output vector of the previous step gives the new output vector,

$$\begin{bmatrix} \dot{S}_0 & \dot{\sigma}_{1,1} & \cdots & \dot{\sigma}_{i,j} & \cdots & \dot{\sigma}_{I,J} & \dot{\sigma}_j^0 & \dot{\sigma}_{j+1}^0 \end{bmatrix}^T.$$

Again, notice that the only derivatives not equal to zero in D^2 are $\frac{\partial \sigma_{j+1}^0}{\partial \sigma_{i,j+1}}$

and $\frac{\mathfrak{d}\sigma^0_{j+1}}{\mathfrak{d}\sigma_{i+1,j+1}}$. Similarly as before these can be computed to,

$$\frac{\mathfrak{d}\sigma^0_{j+1}}{\mathfrak{d}\sigma_{i,j+1}} = \frac{T_{i+1} - t_0}{T_{i+1} - T_i}$$

and

$$\frac{\mathfrak{d}\sigma^0_{j+1}}{\mathfrak{d}\sigma_{i+1,j+1}} = \frac{t_0 - T_i}{T_{i+1} - T_i}$$

The third step of evaluating P is to determine σ_0, therefore the third step now is to determine the derivative of σ_0 with respect to all input parameters. This matrix D^3 takes the form,

$$\begin{bmatrix} & & & & & & & \\ & & & \mathbb{1} & & & & \\ & & & & & & & \\ & & & & & & & \\ \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}S_0} & \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma_{1,1}} & \cdots & \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma_{i,j}} & \cdots & \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma_{I,J}} & \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_j} & \frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_{j+1}} \end{bmatrix}$$

and multiplying it with the output vector of the previous step gives the new output vector,

$$\begin{bmatrix} \dot{S}_0 & \dot{\sigma}_{1,1} & \cdots & \dot{\sigma}_{i,j} & \cdots & \dot{\sigma}_{I,J} & \dot{\sigma}^0_j & \dot{\sigma}^0_{j+1} & \dot{\sigma}_0 \end{bmatrix}^T.$$

Similarly to the previous step, only a few of the derivatives in D^3 will take a nonzero value. In this case these derivatives are $\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_j}$, $\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_{j+1}}$ and $\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}S_0}$ which take the following values,

$$\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_j} = 1 + (S_0 - S^*_j)\frac{0-1}{S^*_{j+1} - S^*_j} = \frac{S^*_{j+1} - S_0}{S^*_{j+1} - S^*_j}, \tag{3.7}$$

$$\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}\sigma^0_{j+1}} = (S_0 - S^*_j)\frac{1-0}{S^*_{j+1} - S^*_j} = \frac{S_0 - S^*_j}{S^*_{j+1} - S^*_j} \tag{3.8}$$

and

$$\frac{\mathfrak{d}\sigma_0}{\mathfrak{d}S_0} = \frac{1}{S^*_{j+1} - S^*_j}(\sigma^0_{j+1} - \sigma^0_j). \tag{3.9}$$

The fourth step of evaluating P is to determine S_1, which means that the fourth step here is to evaluate the derivative of S_1 with respect to the in-

put parameters. The matrix D^4 corresponding to this have the following appearance,

$$\begin{bmatrix} & & & & & & & & \\ & & & & \mathbb{1} & & & & \\ & & & & & & & & \\ \frac{\partial S_1}{\partial S_0} & \frac{\partial S_1}{\partial \sigma_{1,1}} & \cdots & \frac{\partial S_1}{\partial \sigma_{i,j}} & \cdots & \frac{\partial S_1}{\partial \sigma_{I,J}} & \frac{\partial S_1}{\partial \sigma_j^0} & \frac{\partial S_1}{\partial \sigma_{j+1}^0} & \frac{\partial S_1}{\partial \sigma_0} \end{bmatrix}$$

and multiplying it with the output vector of the previous step gives the new output vector,

$$\begin{bmatrix} \dot{S}_0 & \dot{\sigma}_{1,1} & \cdots & \dot{\sigma}_{i,j} & \cdots & \dot{\sigma}_{I,J} & \dot{\sigma}_j^0 & \dot{\sigma}_{j+1}^0 & \dot{\sigma}_0 & \dot{S}_1 \end{bmatrix}^T.$$

The only nonzero derivatives in this matrix D^4, are $\frac{\partial S_1}{\partial S_0}$ and $\frac{\partial S_1}{\partial \sigma_0}$ and their values are,

$$\frac{\partial S_1}{\partial S_0} = 1 + r\Delta t + \sigma_0 \Delta W_0 \tag{3.10}$$

and

$$\frac{\partial S_1}{\partial \sigma_0} = S_0 \Delta W_0. \tag{3.11}$$

Then, basically the above process starts over again by evaluating first σ_j^1 followed by σ_{j+1}^1, σ_1, and finally $\dot{S}_2$. This process iterates N times until $\dot{S}_N$ is known, i.e. at this point there will be $N \cdot 4$ different D matrices. When $\dot{S}_N$ is known there is only one more derivative to evaluate, namely the one corresponding to the final step of evaluating the payoff function when S_N is known. The matrix $D^{N\cdot 4+1}$ corresponding to last step have the following appearance,

$$\begin{bmatrix} & & & & & & & & \\ & & & & \mathbb{1} & & & & \\ & & & & & & & & \\ \frac{\partial P}{\partial S_0} & \cdots & \frac{\partial P}{\partial \sigma_{i,j}} & \cdots & \frac{\partial P}{\partial \sigma_{I,J}} & \cdots & \frac{\partial P}{\partial \sigma_{j+1}^{N-1}} & \frac{\partial P}{\partial \sigma_{N-1}} & \frac{\partial P}{\partial S_N} \end{bmatrix}$$

multiplying this D matrix with the input vector containing all dot values up

to $\dot{S}_N$ gives the following output which contain the sought value $\dot{P}$,

$$\begin{bmatrix} \dot{S}_0 & \dot{\sigma}_{1,1} & \cdots & \dot{\sigma}_{i,j} & \cdots & \dot{\sigma}_{I,J} & \cdots & \dot{\sigma}_j^N & \dot{\sigma}_{j+1}^N & \dot{\sigma}_N & \dot{S}_N & \dot{P} \end{bmatrix}^T.$$

The only derivative in $D^{N\cdot 4+1}$ taking a nonzero value is $\frac{\mathfrak{d}P}{\mathfrak{d}S_N} = \exp(-rT) \cdot \mathbf{1}_{S_N>K}$.

Considering all these matrices, it is quite obvious that there are a lot of unnecessary matrix operations carried out. These operations are likely to create great performance issues. Although, this can be handled if taking a slightly different approach than the algorithmic procedure previously described and instead focusing on the derivatives actually contributing to the sought value.

To begin, again consider the derivative of P with respect to one of the nodes $\sigma_{i,j}$ and expand it to $\dot{P} = \frac{\partial P}{\partial S_N}\dot{S}_N$. Here, considering the Equation 1.2, then $\frac{\partial P}{\partial S_N} = \exp(rT)\mathbf{1}_{S_N>K}$ and S_N will be known at the end of the forward sweep, i.e. $\frac{\partial P}{\partial S_N}$ will be known. The trickier part is evaluating $\dot{S}_N$, for this a recursion relationship is required. Taking the derivative of Equation (3.3) with respect to $\sigma_{i,j}$ the following recursion can be derived,

$$\dot{S}_{n+1} = \dot{S}_n(1 + r\Delta t + \sigma(S_n, t_n))\Delta W_n + \dot{\sigma}_n S_n \Delta W_n.$$

Notice that for each step in the recursion of $\dot{S}_n$ a new value $\dot{\sigma}_n$ is required. This dot value can be obtained by first taking the derivative of the interpolation relationship in Equation (3.4) which get the following appearance,

$$\dot{\sigma}_j^n = \dot{\sigma}_{i,j} + (t_n - T_i)\frac{\dot{\sigma}_{i+1,j} - \dot{\sigma}_{i,j}}{T_{i+1} - T_i}$$

The interpolation relation in stock price, Equation (3.5), is then used to derive the following relation for $\dot{\sigma}_n$,

$$\dot{\sigma}_n = \dot{\sigma}_j^n + (S_n - S_j^*)\frac{\dot{\sigma}_{j+1}^n - \dot{\sigma}_j^n}{S_{j+1}^* - S_j^*}.$$

This process should recursively be repeated until $\dot{S}_N$ is known. When this is done the sought value is determined as $\dot{P} = \frac{\partial P}{\partial S_N}\dot{S}_N$. Now everything needed to run the Monte Carlo simulation is known and the sensitivity of the option price $V = \exp(-rT) \cdot E[P(S_T)]$ with respect to one node $\sigma_{i,j}$ can be estimated. However, since sensitivities of all local volatility nodes were

desired, this whole procedure has to be run $I \cdot J$ times, which of course will be very demanding in computational power. The following section presents the adjoint approach for the same scenario, which according to the theory should be a lot more efficient.

3.1.3 Adjoint implementation

The strict algorithmic procedure of carrying out the adjoint calculations, presented in chapter 2, can now easily be done using the information from a forward sweep. Assuming all D matrices being saved during the forward sweep, the vector containing sensitivities with respect to all input parameters can be obtained through the following calculations,

$$\mathbf{x}_0 = (D^1)^T \cdot (D^2)^T \cdot \ldots \cdot (D^{N\cdot 4})^T \cdot (D^{N\cdot 4+1})^T \cdot \mathbf{x}_N$$

where $\mathbf{x}_N$ is the initial vector of the backward computations, i.e.

$$\mathbf{x}_N = \begin{bmatrix} \bar{S}_0 & \bar{\sigma}_{1,1} & \cdots & \bar{\sigma}_{i,j} & \cdots & \bar{\sigma}_{I,J} & \cdots & \bar{\sigma}_j^N & \bar{\sigma}_{j+1}^N & \bar{\sigma}_N & \bar{S}_N & \bar{P} \end{bmatrix}^T$$

here, all bar values are initially set to zero except for $\bar{P} = \frac{\partial P}{\partial P} = 1$. But following this blindly will, as in the case of forward mode, result in very costly matrix vector multiplications. Also, saving all these large D matrices can result in the computer running out of memory. Instead the essential computations should be extracted and carried out manually. The following part will describe which these calculations are, and what information needs to be stored from the forward sweep.

To begin, $\bar{P}$ is as before equal to 1 and the next step is to find a recursive relationship to evaluate $\bar{S}_n$. This is done by first expanding $\bar{S}_n$ to,

$$\bar{S}_n = \frac{\partial P}{\partial S_n} = \frac{\partial S_{n+1}}{\partial S_n} \bar{S}_{n+1}.$$

Now, using Equations (3.3) and (3.5),

$$\frac{\partial S_{n+1}}{\partial S_n} = (1 + r\Delta t + \sigma_n \Delta W_n) + S_n \Delta W_n \frac{1}{S^*_{j+1} - S^*_j} (\sigma^n_{j+1} - \sigma^n_j)$$

hence,

$$\bar{S}_n = ((1 + r\Delta t + \sigma_n \Delta W_n) + S_n \Delta W_n \frac{1}{S^*_{j+1} - S^*_j}(\sigma^n_{j+1} - \sigma^n_j))\bar{S}_{n+1}.$$

This reveals the first values that needs to be stored from the forward sweep, i.e.

$$\frac{\mathfrak{d}S_{n+1}}{\mathfrak{d}S_n} = 1 + r\Delta t + \sigma_n \Delta W_n,$$

$$\frac{\mathfrak{d}S_{n+1}}{\mathfrak{d}\sigma_n} = S_n \Delta W_n \tag{3.12}$$

and

$$\frac{\mathfrak{d}\sigma_n}{\mathfrak{d}S_n} = \frac{1}{S^*_{j+1} - S^*_j}(\sigma^n_{j+1} - \sigma^n_j)$$

which corresponds to the derivatives, in the D matrices, from Equations (3.9), (3.11) and (3.10). Since the value $\bar{S}_N = \frac{\partial P}{\partial S_N}$ is known at the end of the forward sweep, this can be used to initialize the recursion of $\bar{S}_n$. Then, using these bar values of S, the $\bar{\sigma}_n$ values can be computed as,

$$\bar{\sigma}_n = \frac{\partial P}{\partial \sigma_n} = (\frac{\partial S_{n+1}}{\partial \sigma_n})\bar{S}_{n+1} = (S_n \Delta W_n)\bar{S}_{n+1}$$

The only values required for these calculations, except from the $\bar{S}_n$ values, are obviously the ones from Equation (3.12). Further on, using these $\bar{\sigma}_n$ values, derivatives of the partially interpolated values $\bar{\sigma}^n_j$ can be evaluated. Here it is important to consider Equation (3.5) and notice that both σ^n_j and σ^n_{j+1} influences σ_n. It is therefore natural to believe that both $\bar{\sigma}^n_j$ and $\bar{\sigma}^n_{j+1}$ will be influenced by $\bar{\sigma}_n$, and if investigating this further the following relations can be derived,

$$\bar{\sigma}^n_j = \frac{\partial P}{\partial \sigma^n_j} = \left(\frac{\partial \sigma_n}{\partial \sigma^n_j}\right)\bar{\sigma}_n = \left(\frac{S^*_{j+1} - S_n}{S^*_{j+1} - S^*_j}\right)\bar{\sigma}_n$$

and

$$\bar{\sigma}^n_{j+1} = \frac{\partial P}{\partial \sigma^n_{j+1}} = \left(\frac{\partial \sigma_n}{\partial \sigma^n_{j+1}}\right)\bar{\sigma}_n = \left(\frac{S_n - S^*_j}{S^*_{j+1} - S^*_j}\right)\bar{\sigma}_n.$$

In these relations additional values to be saved from the forward sweep are revealed, namely the derivatives

$$\frac{\mathfrak{d}\sigma_n}{\mathfrak{d}\sigma_j^n} = \frac{S_{j+1}^* - S_n}{S_{j+1}^* - S_j^*}$$

$$\frac{\mathfrak{d}\sigma_n}{\mathfrak{d}\sigma_{j+1}^n} = \frac{S_n - S_j^*}{S_{j+1}^* - S_j^*}$$

which correspond to those in Equations (3.7) and (3.8). Also note that if the stock price drifts beyond the volatility grid, i.e. $S_n > S_J^*$, then $\sigma_n = \sigma_J^n$ since σ_n is constant outside the grid. Therefore the corresponding bar relationship is $\bar{\sigma}_J^n = \bar{\sigma}_n$.

The final step now is to compute the bar values of the grid points $\bar{\sigma}_{i,j}$, i.e. the sought derivatives. Similarly to the previous step determining $\bar{\sigma}_j^n$, it seems reasonable to believe that $\bar{\sigma}_j^n$ will influence both $\bar{\sigma}_{i,j}$ and $\bar{\sigma}_{i+1,j}$, and likewise that $\bar{\sigma}_{j+1}^n$ will influence both $\bar{\sigma}_{i,j+1}$ and $\bar{\sigma}_{i+1,j+1}$. However, here it is important to notice that the partially interpolated value σ_j^n for every time-step $t_n \in [T_{i-1}, T_{i+1}]$ will depend on the same grid point $\sigma_{i,j}$. Therefore these calculations will be cumulative, in comparison to the previous calculations where each sensitivity were unique for every time step t_n. Investigating this more thoroughly, cumulative relationships of all these bar values can be found, starting with $\bar{\sigma}_{i,j}$ the following is revealed,

$$\bar{\sigma}_{i,j}^{(n-1)} = \bar{\sigma}_{i,j}^{(n)} + \left(\frac{\partial \sigma_j^n}{\partial \sigma_{i,j}}\right) \bar{\sigma}_j^n.$$

Here the additional index $^{(n)}$ introduced denotes that the bar value of the grid point is being updated, this means that the initial index of $\bar{\sigma}_{i,j}$ will be $^{(N)}$ and the final value of the index will be $^{(0)}$, i.e. $\bar{\sigma}_{i,j}^{(0)}$ denotes the sought value. Also, each derivative $\frac{\partial \sigma_j^n}{\partial \sigma_{i,j}}$ gets the following appearance,

$$\frac{\partial \sigma_j^n}{\partial \sigma_{i,j}} = \left(\frac{T_{i+1} - t_n}{T_{i+1} - T_i}\right)$$

which corresponds to the derivative in Equation (3.6), i.e. these are also values that needs to be stored when performing the forward sweep. Similarly the following relations can be used to obtain the remaining bar values,

$$\bar{\sigma}_{i+1,j}^{(n-1)} = \bar{\sigma}_{i+1,j}^{(n)} + \left(\frac{t_n - T_i}{T_{i+1} - T_i}\right) \bar{\sigma}_j^n,$$

$$\bar{\sigma}_{i,j+1}^{(n-1)} = \bar{\sigma}_{i,j+1}^{(n)} + \left(\frac{T_{i+1} - t_n}{T_{i+1} - T_i}\right) \bar{\sigma}_{j+1}^{n}$$

and

$$\bar{\sigma}_{i+1,j+1}^{(n-1)} = \bar{\sigma}_{i+1,j+1}^{(n)} + \left(\frac{t_n - T_i}{T_{i+1} - T_i}\right) \bar{\sigma}_{j+1}^{n}.$$

Notice that, for each step in n there will only be four grid points accumulating an additional value when being updated, however the recently introduced index $^{(n)}$ will decrease for all grid points in each step. This concludes the adjoint calculations, since now the sensitivities with respect to all local volatility nodes can be determined. One key observation to make is that in forward mode, just a few of the nonzero derivatives contributes to the retrieved value from one sweep, but in the case of adjoint mode all nonzero derivatives will contribute to the result in one sweep. Also, notice that the adjoint mode now only required one set of Monte Carlo simulated paths, which is a factor of $I \cdot J$ less than in the case of forward mode. The following section will present some numerical results on the performance tests of both methods.

3.1.4 Numerical results

In the following tables, Table 3.2, Table 3.3 and Table 3.4, numerical results, computed using forward mode, adjoint mode and finite difference, are presented. Table 3.1 clarifies the correspondence of the indices I and J to stock price and time respectively. The calculations were made for a hypothetical grid of local volatilities. The grid was of size 5×5 and a linearly interpolated illustration of the grid is shown in Figure 3.1.

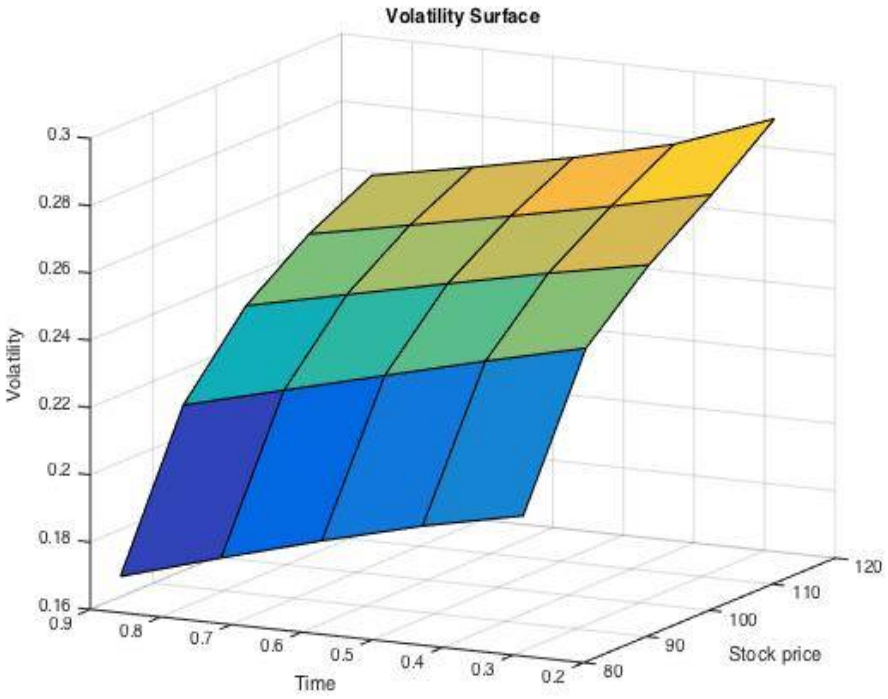

Figure 3.1: *A 3D-Plot showing the hypothetical local volatility surface used as input for this example.*

Also, the other input parameters were given the following values, $K = 100$, $S_0 = 100$, $r = 0.05$, $T = 1$. The number of steps in the discretisation in time was set $n = 100$ and the number of Monte Carlo paths was set to $M = 10^6$.

Table 3.1: **Indices values**. *The table shows the correspondence of the indices I and J to stock price and time respectively.*

Indices value	1	2	3	4	5
Stock price (i)	80	90	100	110	120
Time (j)	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{5}{6}$

Table 3.2: **Finite Difference**. *Numerical values of sensitivities calculated by using a finite difference approach, each cell corresponds to the sensitivity of P with respect to one grid point $\sigma_{i,j}$.*

$\frac{\partial P}{\partial \sigma_{i,j}}$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.1018	0.9753	2.6703	0.8924	0.1594
$i=2$	0.7382	2.3759	3.2527	1.9540	0.8973
$i=3$	1.0368	2.3660	2.8260	1.8346	1.1489
$i=4$	0.6104	2.3370	3.3493	1.8871	0.8021
$i=5$	0.0649	0.8915	2.3402	0.8389	0.1065

Table 3.3: **Forward mode**. *Numerical values of sensitivities calculated by using forward mode, each cell corresponds to the sensitivity of P with respect to one grid point $\sigma_{i,j}$.*

$\frac{\partial P}{\partial \sigma_{i,j}}$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.1016	0.9669	2.7434	0.8793	0.1554
$i=2$	0.7335	2.4214	3.3151	1.9429	0.9122
$i=3$	1.0245	2.3716	2.8496	1.8436	1.1634
$i=4$	0.6068	2.3673	3.3813	1.9162	0.8134
$i=5$	0.0657	0.8936	2.3588	0.8494	0.1547

Table 3.4: **Adjoint mode**. *Numerical values of sensitivities calculated by using adjoint mode, each cell corresponds to the sensitivity of P with respect to one grid point $\sigma_{i,j}$.*

$\frac{\partial P}{\partial \sigma_{i,j}}$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.1003	0.9651	2.6930	0.8885	0.1567
$i=2$	0.7350	2.4097	3.3258	1.9641	0.9109
$i=3$	1.0196	2.3806	2.8352	1.8567	1.1500
$i=4$	0.6067	2.3666	3.3780	1.9072	0.8014
$i=5$	0.0670	0.8950	2.3642	0.8413	0.1319

Notice that a difference compared to the results of the simple example in section 2.3 is that here the numerical values computed using forward and adjoint mode are not identically. This is because of the randomness incurred by the Monte Carlo simulation. However, the results computed using the different methods are all agreeing on at least one decimal digits.

To give some insight of the computational cost, the plot in Figure 3.2 shows the computation times using forward, adjoint and finite difference to compute sensitivities with respect to all nodes for grids of different sizes. The grid sizes are varied from 2×2 to 20×20, i.e. 4 to 400 grid points.

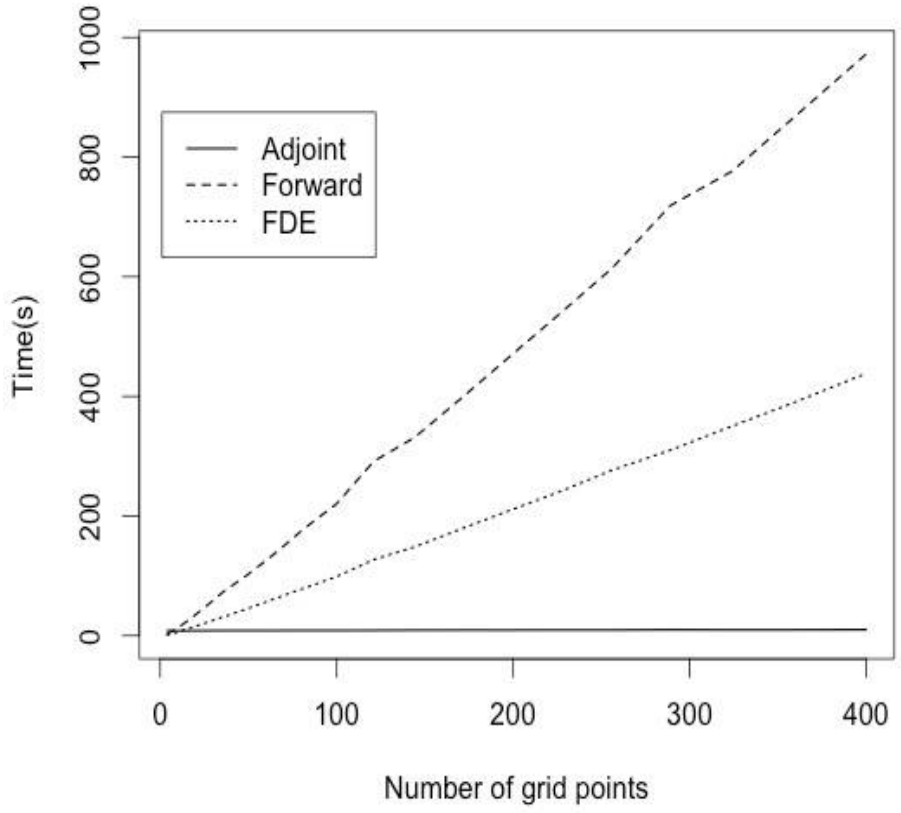

Figure 3.2: *The figure shows the computation time as a function of the number of grid points on the volatility surface, for all three methods.*

It can be seen that in this case the finite difference method outperforms the forward mode, but this performance difference might be due to the way it has been implemented in code. However, this plot clearly shows the performance benefits of using the adjoint mode compared to both forward mode and finite difference, especially when the size of the grid is large.

3.2 Basket option

In this section it will be shown how an algorithmic differentiation approach can be used to evaluate the sensitivities of an basket option. A basket option

is a financial derivative where there are multiple underlyings, for example stocks or commodities. The option gives the owner the right, but not the obligation, to buy the underlying assets (assuming it is a call option). A basket option which has stocks as underlyings has the following payout,

$$P = \max\left(\sum_{i=1}^{m}(w^{(i)}S_N^i) - K, 0\right), \tag{3.13}$$

where m is the total number of stocks, w is the weight for each stock which sums to one, N is the options maturity and K is the strike. Each underlying stock is an independent stochastic process having the following dynamics,

$$dS_t^i = rS_t^i dt + \sigma^{(i)} dW_t^i.$$

where $\sigma^{(i)}$ is the volatility which is independent of S and t, r is the risk-free interest rate, and dW_t^i the Wiener process increment of stock i. There is no explicit solution for the price of a basket option, assuming there are at least two underlying assets, instead Monte Carlo simulation can be used to estimate the option price. The following will show how to implement both the forward and adjoint mode for the case of a basket option.

3.2.1 Forward implementation

Now the calculation of the delta sensitivities for the basket option in forward mode will be shown. Considering Equation (1.5), in the case of deterministic interest rate and volatility, the step function for each stock S^i takes the following form,

$$S_{n+1}^i = S_n^i(1 + r\Delta t + \sigma^{(i)}\Delta W_n^i). \tag{3.14}$$

In this example the stocks are uncorrelated with each other, so they will not interfere with each others paths. To begin, consider only the derivative with respect to one stock S^i, i.e. $\frac{\partial P}{\partial S_0^i}$. Using the theory of algorithmic differentiation and starting with the initial vector of dot values,

$$\begin{bmatrix}\dot{S}_0^1 & \dots & \dot{S}_0^m\end{bmatrix}^T,$$

notice that only one of these enteries are equal to one and the rest are zero. Then the first matrix vector multiplication takes the following form,

$$\begin{bmatrix} \dot{S}_0^1 \\ \vdots \\ \dot{S}_0^m \\ \dot{S}_1^1 \end{bmatrix} = \begin{bmatrix} & \mathbb{1} & \\ & & \\ \frac{\partial S_1^1}{\partial S_0^1} & \cdots & \frac{\partial S_1^1}{\partial S_0^m} \end{bmatrix} \begin{bmatrix} \dot{S}_0^1 \\ \vdots \\ \dot{S}_0^m \end{bmatrix}.$$

similar operations then has to be done m times calculating all $\dot{S}_1^1, \dot{S}_1^2 \dots \dot{S}_1^m$. Then the $\dot{S}$ values of the following time step has to be calculated, starting with $\dot{S}_2^1$ which gives the following matrix vector multiplication,

$$\begin{bmatrix} \dot{S}_0^1 \\ \vdots \\ \dot{S}_0^m \\ \dot{S}_1^1 \\ \vdots \\ \dot{S}_1^m \\ \dot{S}_2^1 \end{bmatrix} = \begin{bmatrix} & & \mathbb{1} & & & \\ & & & & & \\ \frac{\partial S_2^1}{\partial S_0^1} & \cdots & \frac{\partial S_2^1}{\partial S_0^m} & \frac{\partial S_2^1}{\partial S_1^1} & \cdots & \frac{\partial S_2^1}{\partial S_1^m} \end{bmatrix} \begin{bmatrix} \dot{S}_0^1 \\ \vdots \\ \dot{S}_0^m \\ \dot{S}_1^1 \\ \vdots \\ \dot{S}_1^m \end{bmatrix}.$$

This will then have to be repeated until all S_N^i values are determined. As in the previous case of the local volatility surface, following this method blindly, a lot of matrix vector operations then has to be carried out and this is obviously not feasible. Instead this can be solved by analyzing which derivatives are essential to determine the desired sensitivities. Again, consider the derivative with respect to one specific stock, for example, $\frac{\partial P}{\partial S_0^1}$. Now, considering the equation $\dot{P} = \frac{\partial P}{\partial S_N^1}\dot{S}_N^1$, then $\frac{\partial P}{\partial S_N^1}$ will be known at the end of a forward sweep, but $\dot{S}_N^1$ will be more complex to evaluate. Although, using Equation (3.14) yields the recursive relations for all $\dot{S}^1$,

$$\dot{S}_{n+1}^1 = \frac{\partial S_{n+1}^1}{\partial S_n^1} = (1 + r\Delta t + \sigma^{(1)}\Delta W_n^1)\dot{S}_n^1.$$

As mentioned, the $\frac{\partial P}{\partial S_N^1}$ part of $\dot{P}$ is trivial, because P only depends on the values S_N^i which are known, i.e.

$$\frac{\partial P}{\partial S_N^1} = \mathbf{1}_{\sum_{i=1}^m (w^{(i)} S_N^i) > K}.$$

To get a correct estimate of the sought value, the Monte Carlo method is

applied and the algorithm used to determine $\dot{P} = \frac{\partial P}{\partial S_0^1}$ has to be run a large number of times. Then to determine all deltas i.e. $\frac{\partial P}{\partial S_0^i}$ for all $i = 1 \dots m$, this whole process has to be done for each of the underlying stocks in the basket.

3.2.2 Adjoint implementation

By the same arguments made in the forward implementation, it is more efficient to extract the important derivatives and make the necessary calculations manually. Consider the equation for $\bar{S}_n^i$ and expand it as the following,

$$\bar{S}_n^i = \frac{\partial P}{\partial S_n^i} = \frac{\partial S_{n+1}^i}{\partial S_n^i} \frac{\partial P}{\partial S_{n+1}^i}$$

then, using equation (3.14), yields the following relation,

$$\bar{S}_n^i = \frac{\partial S_{n+1}^i}{\partial S_n^i} \bar{S}_{n+1}^i = (1 + r\Delta t + \sigma^{(i)} \Delta {W_n}^i) \bar{S}_{n+1}^i$$

This means that when the initial forward sweep is run, all the values corresponding to the expression $(1 + r\Delta t + \sigma^i \Delta {W_n}^i)$ has to be stored which will later be used to recursively calculate all $\bar{S}_0^i$. But to initialize the recursion in the adjoint mode the values $\bar{S}_N^i$ has to be evaluated, which is done the following way [1],

$$\bar{S}_N^i = w^{(i)} \exp(-rT) \mathbf{1}_{\sum_{i=1}^m (w^{(i)} S_N^i) > K}.$$

When these values are known the recursive process of evaluating $\bar{S}_0^i$ can begin, and if following the adjoint calculations all of these should be evaluated during one sweep.
The returned vector $\bar{\mathbf{S}}$ contains derivatives with respect to all stocks. But as in the case of forward mode, to get satisfactory results the Monte Carlo method is applied.

3.2.3 Numerical results

In the following table, Table 3.5, numerical results, computed using forward mode, adjoint mode and finite difference, are presented. The calculations were made for a basket option with five underlying stocks, all with equal

initial stock value. The input parameters were given the following values, $K = 100$, $S_0 = 100$, $r = 0.05$, $T = 1$, with the stocks weights as $w^{(1)} = \frac{1}{15}$, $w^{(2)} = \frac{2}{15}$, $w^{(3)} = \frac{3}{15}$, $w^{(4)} = \frac{4}{15}$ and $w^{(5)} = \frac{5}{15}$ and the volatilities $\sigma^{(1)} = 0.30$, $\sigma^{(2)} = 0.35$, $\sigma^{(3)} = 0.40$, $\sigma^{(4)} = 0.45$ and $\sigma^{(5)} = 0.50$. The number of steps in the discretisation in time was set $n = 100$ and the number of Monte Carlo paths were set to $M = 10^6$.

Table 3.5: **Basket option**. *Numerical results of sensitivities calculated by finite difference, forward mode and adjoint mode.*

	Finite difference	Forward	Adjoint
$\frac{\partial P}{\partial S_0^1}$	0.036	0.036	0.036
$\frac{\partial P}{\partial S_0^2}$	0.075	0.075	0.075
$\frac{\partial P}{\partial S_0^3}$	0.118	0.118	0.118
$\frac{\partial P}{\partial S_0^4}$	0.168	0.167	0.167
$\frac{\partial P}{\partial S_0^5}$	0.222	0.222	0.223

As can be seen, all methods produces equal estimates of the sensitivities up to the third decimal digit.

To give some insight of the computational cost, the plot in Figure 3.3 shows the computation times using forward, adjoint and finite difference to compute sensitivities with respect to all stocks for basket options containing different number of underlings. The number of underlyings are varied from two to twenty stocks. Here the volatilities of the stocks were set to the following values $\sigma^{(1)} = 0.20$, $\sigma^{(2)} = 0.21 \ldots \sigma^{(20)} = 0.40$, i.e. for the basket with two stocks $\sigma^{(1)}$ and $\sigma^{(2)}$ were used and for the case with three stocks $\sigma^{(1)}$, $\sigma^{(2)}$ and $\sigma^{(3)}$ were used and so on.

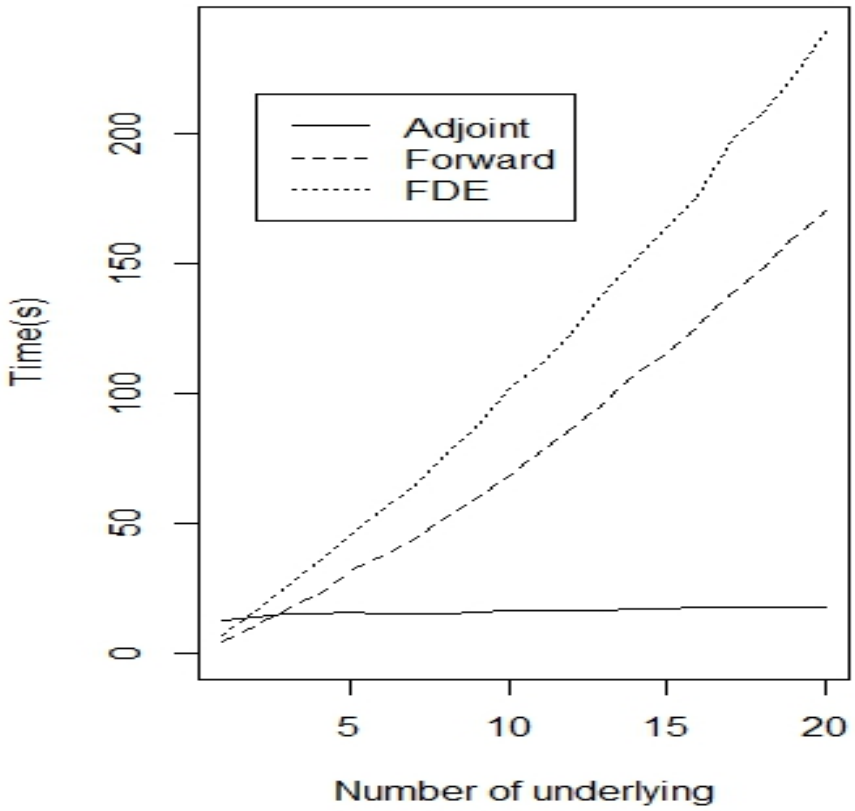

Figure 3.3: *The figure shows the computation time as a function of the number of underlying stocks in the basket option, for all three methods.*

Here, in comparison to the previous case of the volatility surface, it can be seen that using forward mode is faster than using finite difference. Though, similar to the previous case, adjoint mode outperforms both forward mode and finite difference, especially when the number of underlyings is large.

3.3 LIBOR Market Model

The LIBOR Market Model is a collection of different LIBOR forward rates, each having the following dynamics

$$\frac{dL_i(t)}{L_i(t)} = \mu_i(t)dt + \boldsymbol{\sigma}_i(t)^T \boldsymbol{dW}(t). \tag{3.15}$$

Here i is the index of one of the LIBOR forward rates, $L_i(t)$ is the LIBOR forward rate with the index i observed at time t, $\boldsymbol{dW}(t)$ is from a m dimensional Wiener process where m is the number of LIBOR forward rates, $\boldsymbol{\sigma}_i(t)$ is a column vector of volatilities, μ_i is the drift which is calculated the following way

$$\mu_i(\mathbf{L}(t), t) = \sum_{j=\eta(t)}^{i} \frac{L_j(t)\boldsymbol{\sigma}_i(t)^T \boldsymbol{\sigma}_j(t)}{1 + L_j(t)\Delta t}\Delta t, \tag{3.16}$$

where $\eta(t)$ will return the time index of the next maturity of the LIBOR forward rates as of time t. To price a derivative with LIBOR rates as underlyings, one has to calibrate the LIBOR Market Model to the market. To begin the LIBOR rates seen from today can be determined using today's yield rates. This can be calculated using the following formula [1],

$$F(t, T_1, T_2) = \frac{1}{T_2 - T_1}\left(\frac{Z_t^{T_1} - Z_t^{T_2}}{Z_t^{T_2}}\right), \tag{3.17}$$

where t is the time of today, T_1 and T_2 are the initiation and termination time of the forward rate, respectively, and $Z_t^{T_i}$ is the yield rate from t to T_i. Note that in this report the time interval will always be half a year, so the third argument in the LIBOR forward rate is omitted. Then using the Black's formula and currently traded derivatives (for example caplets) on the market, the implied volatility for different maturities on the caplets can be determined [1]. In this report a simplification is made for the volatilities $\boldsymbol{\sigma_i(t)}$, a parametric scalar volatility function usually referred as a "hump function" is used, which has the following appearance,

$$\begin{cases} \sigma_i(t) = (a(i-t) + b)\exp(-c(i-t)) + d, & \text{if } i \geq \eta(t) \\ 0, & \text{otherwise,} \end{cases}$$

where the term $(i-t)$ describes how much time there is left until the i:th forward rate matures. To clarify, for example $\boldsymbol{\sigma_i(t^*)}$ contains the values $\sigma_i(t)$ for all t so that $t^* \leq t \leq t^{max}$ (here t^{max} corresponds to the maturity of the last LIBOR rate being evaluated) and the values $\sigma_i(t)$ with $t < t^*$ are set to zero. The parameters a, b, c, d should be calibrated from the market, but no calibration has been made in this report, instead previously calibrated values has been used. These values are $a = 0.3, b = -0.02, c = 0.7, d = 0.14$ [12]. This simplification was justified since no market data could be retrieved by reasonable means. Also, it does not effect the analysis of the methods being evaluated in this project. However, now the simulation of the LIBOR forward rates can begin, applying the Euler scheme to the logarithm of the LIBOR rates the step function becomes,

$$\begin{aligned} &L_i(t+1) = \\ &= L_i(t) \exp\left((\mu(L_i,t) - \frac{||\boldsymbol{\sigma}_i(t)||^2}{2})\Delta t + \boldsymbol{\sigma}_n^T(t)\boldsymbol{W}(n+1)\sqrt{\Delta t} \right). \end{aligned} \tag{3.18}$$

Here $||\boldsymbol{\sigma}_i(t)||$ denotes the euclidean norm of $\boldsymbol{\sigma}_i(t)$. Note that after a LIBOR forward rate reaches maturity, the step equation for that LIBOR forward rate becomes $L_n(t+1) = L_n(t)$. All the values for σ_n are known from the calibration and the drift term has to be calculated for each step because it depends on the current LIBOR forward rates. It is calculated using Equation (3.16). When the simulation is done the paths can now be used to price interest rate derivatives, such as a caplet. A caplet option has the following payoff function,

$$\left(\prod_{j=0}^{m} \frac{1}{1 + L_j(j)\Delta t} \right) \max(0, L_m(m) - K)\Delta t. \tag{3.19}$$

Note that in this report where $\Delta T = \frac{1}{2}$, there will be as many maturities indices as LIBOR forward rate indices. Therefore the discount part of (3.19) will correspond to the correct time.

3.3.1 Forward implementation

Now the calculation of the delta sensitivities for the caplet option in forward mode will be shown. Using the theory of algorithmic differentiation the

initial vector of dot values are,

$$\begin{bmatrix} \dot{L}_0(0) & \dots & \dot{L}_m(0) \end{bmatrix}^T,$$

notice that only one of these entries are equal to one and the rest are zero. The first step is to calculate the derivative of μ_0, hence the first matrix vector multiplication takes the following form,

$$\begin{bmatrix} \dot{L}_0(0) \\ \vdots \\ \dot{L}_m(0) \\ \dot{\mu}_0(0) \end{bmatrix} = \begin{bmatrix} & \mathbb{1} & \\ & & \\ \frac{\mathfrak{d}\mu_0(0)}{\mathfrak{d}L_0(0)} & \dots & \frac{\mathfrak{d}\mu_0(0)}{\mathfrak{d}L_m(0)} \end{bmatrix} \begin{bmatrix} \dot{L}_0(0) \\ \vdots \\ \dot{L}_m(0) \end{bmatrix}.$$

The derivative of (3.16) with respect to LIBOR forward rates is the following,

$$\frac{\mathfrak{d}\mu_i(t)}{\mathfrak{d}L_j(t)} = \frac{\sigma_i(t)\sigma_j(t)\Delta t}{(1+L_j(t)\Delta t)^2}\mathbf{1}_{i\geq\eta(t)}.$$

Similar operations then has to be done m times calculating all $\dot{\mu}_1(0)$, $\dot{\mu}_2(0)$ $\dots$ $\dot{\mu}_m(0)$. Then the $\dot{L}$ values of the following time step has to be calculated, starting with $\dot{L}_0(1)$ which corresponds to the following matrix vector multiplication,

$$\begin{bmatrix} \dot{L}_0(0) \\ \vdots \\ \dot{L}_m(0) \\ \dot{\mu}_0(0) \\ \vdots \\ \dot{\mu}_m(0) \\ \dot{L}_0(1) \end{bmatrix} = \begin{bmatrix} & & \mathbb{1} & & & \\ & & & & & \\ \frac{\mathfrak{d}L_0(1)}{\mathfrak{d}L_0(0)} & \dots & \frac{\mathfrak{d}L_0(1)}{\mathfrak{d}L_m(0)} & \frac{\mathfrak{d}L_0(1)}{\mathfrak{d}\mu_0(0)} & \dots & \frac{\mathfrak{d}L_0(1)}{\mathfrak{d}\mu_m(0)} \end{bmatrix} \begin{bmatrix} \dot{L}_0^1 \\ \vdots \\ \dot{L}_0^m \\ \dot{\mu}_0(0) \\ \vdots \\ \dot{\mu}_m(0) \end{bmatrix}.$$

The derivative of (3.18) with respect to previous LIBOR forward rate is only nonzero if it is the same LIBOR forward rate, then it is derivative is dependent on the next maturity in the following way

$$\frac{\mathfrak{d}L_i(t+1)}{\mathfrak{d}L_i(t)} = \frac{\mathfrak{d}\log(L_i(t+1))}{\mathfrak{d}L_i(t)}L_i(t+1) = \begin{cases} \frac{L_i(t+1)}{L_i(t)}, & \text{if } i \geq \eta(t) \\ 1, & \text{otherwise.} \end{cases}$$

The drift derivatives are calculated in the following way

$$\frac{\mathfrak{d}L_i(t+1)}{\mathfrak{d}\mu_j(t)} = \begin{cases} \frac{\boldsymbol{\sigma}_i(t)^T \boldsymbol{\sigma}_j(t)\Delta t}{(1+L_j(t)\Delta t)^2}, & \text{if } i \geq \eta(t) \\ 0, & \text{otherwise.} \end{cases}$$

This will then have to be repeated until all $L_i(m)$ values are determined. As in the previous case of the local volatility surface, following this method blindly, a lot of matrix vector operations then has to be carried out and this is obviously not feasible. Instead this can be solved by analyzing which derivatives that are essential to determine the desired sensitivities. Again, consider the derivative with respect to one specific LIBOR forward rate, for example, $\frac{\partial P}{\partial L_1(m)}$. Now, considering the equation $\dot{P} = \frac{\partial P}{\partial L_1(m)}\dot{L}_1(m)$, then $\frac{\partial P}{\partial L_1(m)}$ will be known at the end of a forward sweep, but $\dot{L}_1(m)$ will be more complex to evaluate. One way to do it is by first evaluating a drift step, then evaluating a LIBOR forward rate step. Using Equation (3.16) yields the drift step relation,

$$\dot{\mu}_i(n+1) = Q_{\eta(n)}(n,i)\dot{L}_{\eta(n)}(n) + Q_{\eta(n)+1}(n,i)\dot{L}_{\eta(n)+1}(n) + ... + Q_i(n,i)\dot{L}_i(n),$$

where

$$Q_n(t,i) = \frac{\boldsymbol{\sigma}_n(t)^T \boldsymbol{\sigma}_i(t)\Delta t}{(1+L_n(t)\Delta t)^2}$$

Note that $\dot{\mu}$ is zero if the LIBOR forward rate has already matured. When all the drifts have been calculated for one time step, the next step is to determine the LIBOR forward rates u,sing the following relation,

$$\dot{L}_i(n+1) = \dot{\mu}_i(n)\Delta t + \frac{\partial L_i(t+1)}{\partial L_i(t)}\dot{L}_i(n). \tag{3.20}$$

This has to be repeated until all of the $\dot{\mathbf{L}}(m)$ are calculated. The $\frac{\partial P}{\partial \mathbf{L(m)}}$ part of $\dot{P}$ depends on LIBOR rates at different maturities seen at different times. But because of the step function for matured LIBOR rates, the payout function in Equation (3.19) can be expressed as only being dependent on the final LIBOR rates, instead of LIBOR rates for different maturities seen at

different times. Hence, $\frac{\partial P}{\partial \mathbf{L(m)}}$ can be expressed as the following,

$$\frac{\partial P}{\partial L_i(m)} = \begin{cases} \Delta t \prod_{j=0}^{m} \frac{1}{1+L_j(m)\Delta t}(\mathbf{1}_{L_m(T_m)>K} & \text{if i = m} \\ -(\max(L_m(T_m)-K),0)\frac{\Delta t}{1+L_i(T(m))\Delta t}) & \\ \Delta t \prod_{j=0}^{m} \frac{1}{1+L_j(m)\Delta t} & \text{otherwise} \\ (-\max(L_m(T_m-K),0)\frac{\Delta t}{1+L_i(T(m))\Delta t}). & \end{cases} \tag{3.21}$$

3.3.2 Adjoint implementation

By the same arguments made in the forward implementation, it is more efficient to extract the important derivatives and make the necessary calculations manually. Consider the equation for $\bar{\mathbf{L}}(n)$ and expand it as the following,

$$\begin{aligned} \bar{\mathbf{L}}(n) &= \left(\frac{\partial \mathbf{P}}{\partial \mathbf{L}(n)}\right)^T = \left(\frac{\partial \mathbf{L}(n+1)}{\partial \mathbf{L}(n)}\right)^T \left(\frac{\partial \mathbf{P}}{\partial \mathbf{L}(n+1)}\right)^T \\ &= \left(\frac{\partial \mathbf{L}(n+1)}{\partial \mathbf{L}(n)}\right)^T \bar{\mathbf{L}}(n+1). \end{aligned}$$

Then it can be seen that what has to be saved from the forward sweep is $\frac{\partial \mathbf{L}(n+1)}{\partial \mathbf{L}(n)}$ for each step n. This is a matrix which values can be determined by taking the derivative of Equation (3.18). The matrix will be a square matrix since the number of maturities is the same as LIBOR rates in this example. Hence, the diagonal elements can be determined the following way,

$$\frac{\partial L_i(n+1)}{\partial L_i(n)} = \begin{cases} \frac{L_i(n+1)}{L_i(n)} + \frac{L_i(n+1)||\boldsymbol{\sigma}_i(n)||^2\Delta t^2}{(1+L_i(n)\Delta t)^2} & \text{if } i \geq \eta(n) \\ 1 & \textit{otherwise.} \end{cases}$$

For the other indices in the matrix the following equation can be derived,

$$\frac{\partial L_i(n+1)}{\partial L_j(n)} = \begin{cases} \frac{L_i(n+1)\boldsymbol{\sigma}_i(n)^T\boldsymbol{\sigma}_j(n)\Delta t^2}{(1+L_j(n)\Delta t)^2} & \text{if } i > j \geq \eta(n) \\ 0 & \textit{otherwise.} \end{cases}$$

What is left to calculate is the initial value of $\bar{\mathbf{L}}(N)$, this can be done using Equation (3.21). When these values are known the recursive process of evaluating $\bar{\mathbf{L}}(0)$ can begin and, because of the nature of the adjoint calculations, this will be done in one sweep.

3.3.3 Numerical results

In Table 3.8, numerical results, computed using forward mode, adjoint mode and finite difference, are presented. The calculations were made for a caplet on the LIBOR rate two years from now, i.e. there are five underlying LIBOR rates to consider. The initial five LIBOR rates were determined using yield rates from the market, these yield rates are given in Table 3.6. The corresponding LIBOR rates, calculated by using Equation (3.17), are given in Table 3.7. Also, the strike was set to $K = 0.001$ and the number of Monte Carlo paths was set to $M = 10^6$.

Table 3.6: **Yield rates**. *Yield rates collected from the market.*

Time [Years]	1/12	3/12	6/12	1	2	3	5	7	10
Yield rate [%]	0.02	0.02	0.07	0.22	0.60	0.95	1.53	1.92	2.19

Notice that these yield rates also were interpolated so that the following forward rates could be computed using Equation (3.17).

Table 3.7: **Forward rates**. *Forward rates starting on different times in the future, all lasting for 6 months.*

Initiation time [Years]	0	0.5	1	1.5	2	2.5
Forward rate [%]	0.07	0.02	0.02	0.07	0.22	0.60

Table 3.8: **Caplet**. *Numerical results of sensitivities calculated by finite difference, forward mode and adjoint mode.*

	Finite difference	Forward	Adjoint
$\frac{\partial P}{\partial L_1(0)}$	-0.0035	-0.0034	-0.0034
$\frac{\partial P}{\partial L_2(0)}$	-0.0041	-0.0041	-0.0041
$\frac{\partial P}{\partial L_3(0)}$	-0.0047	-0.0049	-0.0047
$\frac{\partial P}{\partial L_4(0)}$	-0.0051	-0.0053	-0.0051
$\frac{\partial P}{\partial L_5(0)}$	0.4805	0.4804	0.4809

As can be seen, all methods produces equal estimates of the sensitivities up to the third decimal digit. Notice that the derivatives with respect to $L_1(0)$, $L_2(0)$, $L_3(0)$, $L_4(0)$ are all negative while the derivative with respect to $L_5(0)$ is positive. This is because of the fact that the four first LIBOR rates are only used to discount the payoff, i.e. the value of the caplet decreases if the rates increases. Also, the caplet is written on the fifth LIBOR rate which means that if that increases the value of the caplet will also increase, which apparently has a larger effect on the price than the fact that the fifth rate is used for discounting as well.

As in the previous examples, to give some insight of the computational cost, the plot in Figure 3.4 shows the computation times using forward, adjoint and finite difference to compute sensitivities. Here, the sensitivities are computed for caplets on different forward rates with respect to all initial forward rates affecting the specific caplet under consideration. The caplets considered, will be written on forward rates from half a year to seven years with increment half a year, i.e. the number of underlying forward rates will be varied from two to fifteen.

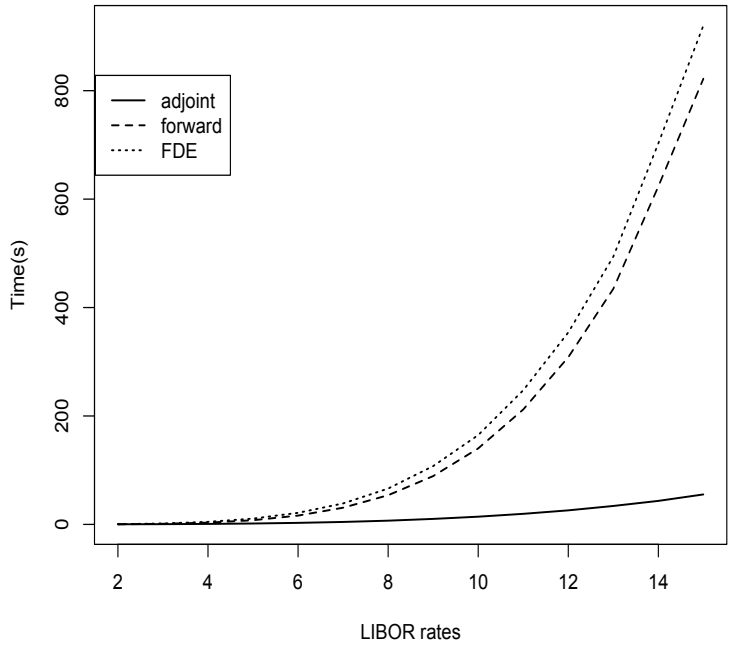

Figure 3.4: *This figure shows the computation time as a function of the number of underlying forward rates, for all three methods.*

Here, it can be seen that using forward mode is more or less equally efficient as using finite difference. Though, similar to the previous case, adjoint mode outperforms both forward mode and finite difference, especially when the number of underlyings is large. One difference in this case is the non-linear growth in computation times for all three methods. This is because of the fact that when increasing the number of forward rates the caplet payoff requires further simulation on each forward rate. In the previous examples the length of the simulations were always the same independent of how many grid points or stocks there were in the basket option. Also, in the LIBOR case, for each added forward rate, all previous forward rates will be influenced. This results in that when simulating each step of the forward rates additional operations has to be carried out, in comparison to the basket case, where adding an extra stock has a negligible effect on the step function.

When estimating the order of complexity with respect to the number of underlyings, other parameters such as Monte Carlo paths are neglected.

Considering the order of complexity of the adjoint mode, the code begins with a loop for all the underlyings except the first one, this adds a factor of $(N-1)$ to the complexity. Nested in this loop there is a loop going through all underlyings an additional time and finally this second loop also contains a nested loop going through all underlyings again, adding a factor (N^2) to the complexity. This yields the final order of complexity $\mathcal{O}((N-1)N^2)$ for the adjoint method. Using forward mode, operations with similar order of complexity as in the adjoint case has to be carried out. In addition to this, forward mode has to run through all operations for each delta sought, thus the order of complexity becomes $\mathcal{O}((N-1)N^3)$. To verify the theoretical claim of the order of the complexity, a log-log plot of the run time is shown in Figure 3.5.

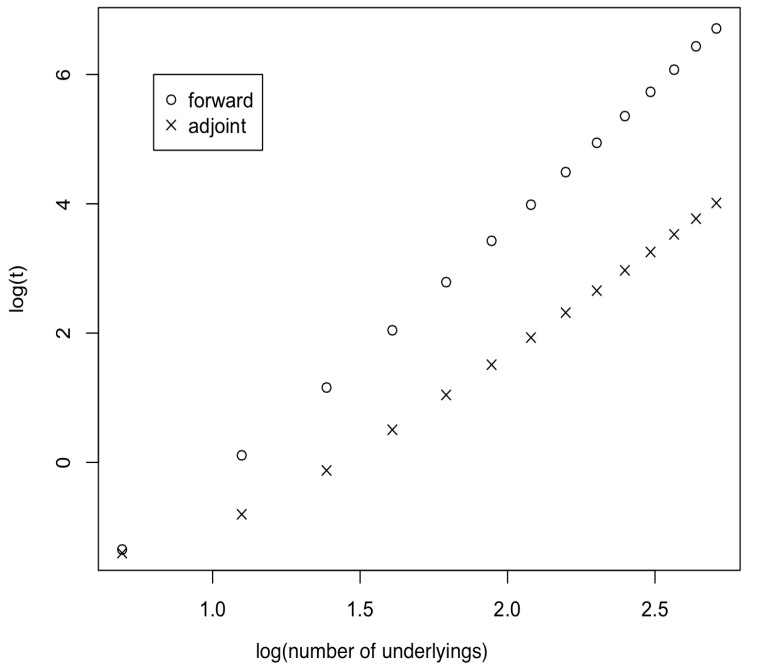

Figure 3.5: *This figure shows a log-log plot of the computation times versus the number of underlyings for both adjoint and forward mode.*

The slope for each method corresponds to their respective order of complexity, and as can be seen in the figure, the slope is approximately three for the

adjoint method and four for the forward method. This confirms the theoretical claim. In the previous case, the basket option, the order of complexity is $\mathcal{O}(1)$ for the adjoint case and $\mathcal{O}(N)$ for the forward. When comparing this to the LIBOR case, it is clear that the order of complexity gain that can be made using the adjoint method is, in both cases, equal to a factor as large as the number of underlyings.

Chapter 4

Conclusions

An overview of algorithmic differentiation has been presented and then applied to three cases commonly occurring in financial applications. The results from all three implementation cases strongly indicates the benefits of using the adjoint approach when calculating the sensitivities. The savings, considering the computational cost, that can be made using this approach is highly dependent on the number of sensitivities desired, i.e. if the number of sensitivities desired is high there are larger savings to be made. Also, the estimated sensitivities, produced from the two different methods, are both aligning with the results using finite difference and are therefore considered to be accurate.

However, writing this code by hand can be very time consuming and errors are easily made. Therefore further studies investigating and evaluating the available tools performing algorithmic differentiation can be of great interest. The requirement of the payoff function to be differentiable when using algorithmic differentiation makes it interesting to evaluate how the Likelihood ratio method, which does not have the same requirement, compliments and works in combination with algorithmic differentiation.

Bibliography

[1] Thomas Björk. *Arbitrage Theory in Continuous Time*. Oxford Finance, 3rd edition, 2009.

[2] Luca Capriotti. Fast Greeks by Algorithmic Differentiation. 2011.

[3] Bruno Duprie. Pricing with a Smile. 2004. Bloomberg.

[4] J. Gatheral and M. Lynch. Stochastic Volatility and Local Volatility. 2002.

[5] M. Giles and P. Glasserman. Smoking Adjoints: Fast Monte Carlo Greeks. 2006.

[6] Michael Giles. Monte Carlo evaluation of sensitivities in computational finance. 2012.

[7] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, 2003.

[8] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, 2nd edition, 2008.

[9] John C. Hull. *Options, Futures and Other Derivatives*. Prentice Hall, 8th edition, 2012.

[10] Rutkowski M. Musiela, M. *Martingale Mehtods in Financial Modelling*. Springer, 2 edition, 2005.

[11] Vytautas Savickas. Fast Greeks: Case of Credit Valuation Adjustments. 2011. Utrecht University.

[12] John Schoenmakers. *Robust Libor Modelling and Pricing of Derivative Products*. Chapman and Hall/CRC, first edition, 2005.

TRITA -MAT-E 2015:70
ISRN -KTH/MAT/E--15/70-SE