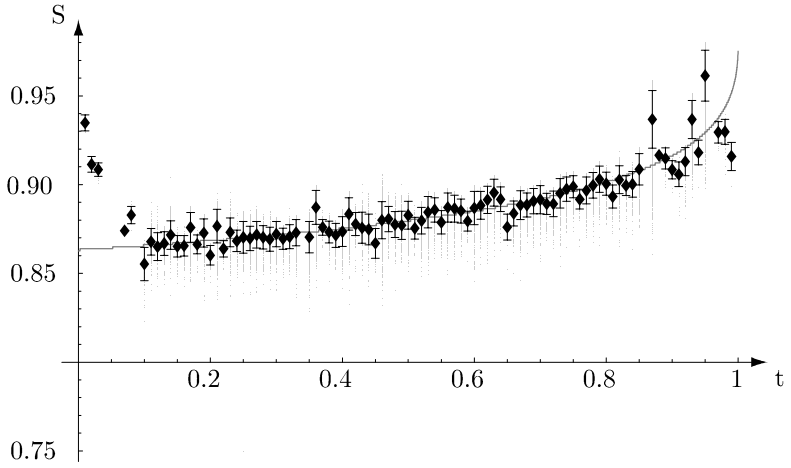
Handelshøjskolen i København
Institut for Finansiering
Erhvervsøkonomi/matematik-studiet
Cand.merc.(mat.) kandidatafhandling
December 2003

Forfatter: Niels Elken Sønderby
Vejleder: Carsten Sørensen

Simulationsbaseret prisfastsættelse af amerikanske optioner i en model med stokastisk volatilitet og spring

Niels Elken Sønderby



http://www.nielses.dk/stud/cand

Indhold

Simulation based pricing of American options in a model with stochastic volatility and jumps

Niels Elken Sønderby

Summary

This master's thesis presents a combination of the SVJD (stochastic volatility and jump diffusion) option pricing model introduced by Bates (1996) and the least-squares Monte Carlo method developed by Longstaff and Schwartz (2001). This makes it possible to price American options in the SVJD-model. The resulting combination is implemented in a computer program in order to make it fast and easy to do the calculations.

The rationale for studying the SVJD-model is that the famous Black-Scholes formula does not yield prices that are consistent with observed market prices. This is partly due to the fact that the Black-Scholes model assumes that the price of the underlying asset follows a geometric Brownian motion with constant volatility. For this reason, a number of extensions to the Black-Scholes model has been suggested. One of the suggested improvements has been to let the volatility follow a stochastic process and thus vary over time. Of these, especially the Heston (1993) model has been quite successful. Another suggested improvement is to extend the model by adding the observed phenomenon that asset prices exhibit jumps. One popular way to model this behaviour is by adding log-normally distributed jumps arriving according to a Poisson process to the geometric Brownian motion. This was first studied by Merton (1976). Bates (1996) combines these two extensions and this thesis refers empirical evidence, that the SVJD-model is a quite good candidate for a model describing market data.

Opposed to the Black-Scholes model it is not possible to make a perfect hedge in the SVJD-model. As a consequence it is neccessary to make assumptions about the utility functions of the investors in order to arrive at a pricing result. Doing this, the SVJD-model can be rewritten as a process under the risk-neutral probability measure. It is then possible to use the characteristic functions to arrive at a semi-closed pricing formula for European call and put options. The formula only requires the numerical integration of two real-valued integrands consisting of elementary functions. In this thesis the Gauss-Kronrod integration algorithm is described and implemented in order to evaluate the pricing formula.

Using other integration formulas similar to the pricing formula, it is pos-

sible to evaluate the probability density function. This possibility is used to graphically illustrate the effects of different choices of model parameters on the skewness and kurtosis of the asset return. The most important effects are as follows: Higher volatility of volatility results in higher kurtosis. Positive correlation between asset price changes and volatility changes results in positive skewness. Oppositely for negative correlation. Jumps results in skewness in the same direction as the mean of the jumps and also gives higher kurtosis. The pricing effects of changing the parameters are also studied together with the resulting volatility smiles. Kurtosis results in a symmetric smil, whereas skewness results in a smirk.

The semi-closed-form pricing formula can only be used for European options. However, as a large part of traded options are American, it is of great use to be able to price these. As the SVJD-model is rather complex, it is difficult to adapt traditional methods like binomial tree and finite difference algorithms for use in this model. The Monte Carlo simulation approach, however, is easy to adapt to new models. Until recently it was considered infeasible to use simulation techniques to price American options. But with the method called least-squares Monte Carlo (LSM) this becomes possible. This thesis shows how LSM can be used to price options in the SVJD-model.

First, a method for simulating the path of the price of the underlying asset in the SVJD-model is developed. For the purpose of variance reduction also the antithetic path is simulated. This makes it possible to price European options using a Monte Carlo technique. As a semi-closed-form solution exists, it is easily checked if the developed Monte Carlo approach prices the options correctly.

Secondly, the LSM algorithm is presented. This approach uses the information across the simulated paths to estimate the continuation value of the American option. This is done using ordinary least-squares regression, for which an algorithm using singular value decomposition (SVD) is used. By working recursively backwards the LSM-method can then be used to find the option price at time 0. In the LSM approach it is shown how using the European option price as a control variate can improve the estimates.

Using the implementation of the LSM algorithm an approximation to the optimal exercise boundary for the American option is shown. In the BS-model this boundary is two-dimensional, whereas in the case of the SVJD-model it is three-dimensional.

In order to assure the accuracy of the implementation, the output of the program is tested. The tests shows that the implementation is accurate. However, prices are biased if the correlation is different from zero. Also pricing in a model with jumps requires many time steps resulting in long calculation times.

All the discussed algorithms are implemented in the program delivered with the thesis. In order to make the calculation routines widely available, they are made as extensions to the open source financial C++ library QuantLib and will be distributed with future versions of the library. Also, as part of this thesis an interface making the routines from QuantLib available through Mathematica has been developed. This assures that the functionality is easily accessible through a popular user interface. The program and other files can be downloaded from http://www.nielses.dk/stud/cand

1 Indledning

Prisfastsættelse af optioner er et centralt emne i moderne finansieringsteori, og til dette formål har Black-Scholes-formlen fra 1973 været utrolig succesfuld. Den er stadig den mest udbredte metode, men da en simpel anvendelse af formlens priser ikke stemmer overens med markedets priser, bliver tallene justeret på forskellig vis. En hyppigt anvendt metode er at justere i forhold til "volatilitetssmilet", der foreskriver, at volatiliteten for det underliggende aktiv afhænger af aftalekursen.

Behovet for justeringer opstår, fordi de underliggende antagelser i Black-Scholes-modellen ikke holder. Dette gælder f.eks. antagelsen om konstant rente i optionens løbetid. Den antagelse, der giver størst problemer i praksis er imidlertid antagelsen om, at afkastet på det underliggende aktiv er log-normalfordelt. Denne fordeling følger af, at prisen på det underliggende aktiv antages at følge en geometrisk brownsk bevægelse. Der har derfor været gjort en række forsøg på at finde andre stokastiske processer, der mere realistisk beskriver prisens udvikling over tid. De mest prominente forsøg beskriver volatiliteten som stokastisk eller indfører muligheden for spring. David Bates har i 1996 præsenteret en model (benævnt SVJD-modellen), der indeholder begge disse elementer og ifølge empiriske undersøgelser er modellen velegnet til at beskrive aktivets prisudvikling. Ydermere udleder han en formel for udregning af optionsprisen givet modellens parametre. Dette giver nye muligheder for på en teoretisk tilfredstillende måde at prisfastsætte optioner.

Bates' lukkede formel kan imidlertid kun beregne prisen på europæiske optioner. Da mange handlede optioner er amerikanske, er det interessant også at kunne regne på disse. En nyudviklet metode til dette er least-squares Monte Carlo, der arver fleksibiliteten fra standard Monte Carlo simulationsteknikken, men giver mulighed for at tage højde for førtidig indfrielse (*early exercise*). Metoden er generel og har mange anvendelser, og i forhold til andre teknikker er det forholdsvist enkelt at anvende den til modeller med stokastisk volatilitet og spring.

Denne afhandling præsenterer SVJD-modellen og least-squares Monte Carlo metoden og kombinerer disse, således at det bliver muligt at udregne

prisen på en amerikansk option i en model med stokastisk volatilitet og spring ved hjælp af simulationsteknik.

Sideløbende med udviklingen af nye teoretiske modeller inden for afledte finansielle produkter er der sket en rivende udvikling i computerteknologi og metoder til at løse finansielle problemstillinger ved hjælp af numeriske metoder. Dette bunder naturligvis i, at de markedsdeltagere, der handler med de forskellige instrumenter, skal kunne regne på dem for at fastsætte en korrekt pris og kunne få tal til brug for risikostyring. Samtidigt er det vigtigt, at udregningen foregår tilstrækkeligt hurtigt, så det i løbet af kort tid er muligt at regne på forskellige scenarier, løbetider osv., så der ikke tabes vigtige sekunder, når markedet bevæger sig hurtigt. Desuden skal beregningerne optimalt set stilles til rådighed på en måde, så de er lettilgængelige, og derved kan benyttes af andre end den, der har udviklet beregningsrutinen.

Denne afhandling vil derfor også behandle de beregningsmæssige aspekter af de givne modeller og præsentere et program for dem.

1.1 Problemformulering

Hovedformålene med denne afhandling vil være at:

- præsentere og beskrive SVJD-modellen (Bates 1996) inkl. den lukkede optionsværdiformel, modellens egenskaber (fordelinger m.v.) og konsekvenser (volatilitetssmil)
- præsentere og beskrive least-squares Monte Carlo metoden (Longstaff & Schwartz 2001) samt omtale resultater omkring dens konvergensegenskaber
- kombinere de to ovennævnte metoder så det bliver muligt at prisfastsætte amerikanske optioner under SVJD-modellen
- udvikle og teste et computerprogram, der implementerer de to metoder

Som baggrund for ovenstående vil den bagvedliggende grundlæggende teori for optionsprisfastsættelse blive introduceret.

1.2 Afgrænsning

For at fastholde fokus på det væsentlige opstilles følgende afgrænsninger:

- Der vil ikke være nogen diskussion af antagelserne omkring markedet m.v.

- Der vil kun i mindre grad blive redegjort for konkurrerende modeller og metoder.
- Metoder til estimation af de modelparametre, der er nødvendige for at modellerne kan bruges i praksis, vil ikke blive beskrevet, men kun omtalt kortfattet.
- Opgaven vil fokusere på valutaoptioner, da disse bliver handlet i stort omfang på min arbejdsplads (Nordea Markets) og generelt har et stort volumen på de finansielle markeder. (De behandlede metoder vil dog umiddelbart kunne anvendes på optioner på dividendebærende aktier, og sammenhængen med disse vil blive beskrevet.)
- Der tages ikke højde for helligdagskalendre og diverse markedskonventioner.

1.3 Værktøjer og metode

En væsentlig del af formålet med denne opgave er at få bragt den finansielle teori frem til et punkt, hvor den er implementeret i et computerprogram og brugbar i praksis. Dette sker for at give opgaven et mere virkelighedsnært præg.

Opgaven vil tage udgangspunkt i det finansielle problem og komme med en matematisk beskrivelse heraf. Derefter ønskes dette implementeret i en beregningsmotor, der kan regne hurtigt og stilles til rådighed i forskellige sammenhænge. For at det skal være nemt for interesserede brugere at anvende programmets funktionalitet, ønskes en brugervenlig grænseflade til programmet.

Til beregningsmotoren er valgt programmeringssproget C++[1], da det giver mulighed for hurtige beregninger og en struktureret opbygning af koden. C++ er udbredt i den finansielle verden og giver gode integrationsmuligheder til mange forskellige andre sprog. For at undgå at starte på bar bund er koden lavet som udvidelser til det finansielle C++-toolkit QuantLib (Ametrano & Ballabio 2003). Denne "værktøjskasse" indeholder dels de grundlæggende rutiner til f.eks. generering af tilfældige tal, simulation af Monte Carlo stier, tilbagediskontering o.s.v., dels rutiner til prisfastsættelse af optioner m.v. ved hjælp af forskellige metoder (analytisk, endelig differens, Monte Carlo). Dette rutinebibliotek gør det muligt at komme ud over det grundlæggende niveau, hvor metoderne, der studeres allerede er beskrevet og implementeret adskillige gange, til et niveau, hvor det er muligt udvikle programmer af

[1] En introduktion til C++ findes i Koenig & Moo (2000).

mere nyskabende karakter. Derudover giver den tætte sammenkobling med QuantLib-biblioteket mulighed for at få distribueret koden sammen med dette og dermed opnå en større anvendelse af de udarbejdede beregningsrutiner.

Da en C++ implementation af en beregningsrutine ikke umiddelbart er let tilgængelig, er det normalt at gøre denne tilgængelig via et andet programmeringssprog eller en decideret brugergrænseflade. Den kunne f.eks. stilles til rådighed som funktion i Microsoft Excel, som objekt i Visual Basic eller som funktion i en matematik-programpakke. I denne afhandling er det valgt at eksportere C++-funktionerne til Mathematica (Wolfram 1999). Det samlede program får herved en stor lighed med et kommercielt produkt som UnRisk (2003), selvom omfanget af funktionaliteten selvsagt vil være markant mindre. Mathematica er valgt, fordi det er et stærkt matematikprogram, hvor den interaktive brugergrænseflade gør det nemt for slutbrugeren at ændre inddata og afprøve beregninger i forskellige scenarier. Det har meget fleksible værktøjer til graftegning og giver også mulighed for statistisk efterbehandling af resultaterne.

Hermed nås der tilbage til den finansielle problemstilling, hvor de beregnede resultater har en finansiel fortolkning.

1.4 Afhandlingens struktur

Strukturen i kandidatafhandlingen er som følger:

Kapitel 2 introducerer den velkendte Black-Scholes metode for prisfastsættelse af optioner. Begreberne "ingen arbitrage" og "det risikofrie sandsynlighedsmål" introduceres. Ækvivalensen mellem en valutaoption og en aktie med kontinuert dividende præsenteres. I kapitel 3 gives en motivation for valget af en udvidet model (SVJD-modellen). Modellen og dens egenskaber præsenteres, og det omtales hvilke metoder, der kan bruges for at estimere dens parametre. For at komme frem til en lukket formel for optionsprisen i SVJD-modellen introducerer kapitel 4 det risikoneutrale mål i denne model. Herefter præsenteres formlen og dens implementation i C++ – herunder Gauss-Kronrod integration.

Kapitel 5 beskriver den klassiske Monte Carlo procedure for europæiske optioner. Efter at have introduceret metoden i Black-Scholes setup'et beskrives, hvorledes den risikoneutrale process fra SVJD-modellen kan simuleres, og det bliver muligt at bruge Monte Carlo metoden til at prisfastsætte europæiske optioner i denne model. Kapitel 6 behandler prisfastsættelse af amerikanske optioner. Først bliver der argumenteret for, at least-squares Monte Carlo metoden i forhold til alternativerne er velegnet til opgaven. Herefter

beskrives metoden, og kapitlet slutter med, at SVJD-modellen kombineres med least-squares Monte Carlo algoritmen. Således bliver det muligt at prisfastsætte amerikanske optioner i denne model med stokastisk volatilitet og spring.

Kapitel 7 vil godtgøre, at det udviklede program regner korrekt samt se på effekten af de variansreducerende teknikker. Kapitel 8 konkluderer opgaven.

Bilag A indeholder en oversigt over de brugte symboler og variabelnavne.

En væsentlig del af denne kandidatafhandlings produkt er det udviklede program. For ikke at forstyrre fremstillingen af teorien er detaljer omkring implementeringen samt programmets kildekode imidlertid placeret i bilag.

Bilag B indeholder en oversigt over den vedlagte CD-ROM's indhold. Bilag C indeholder en brugervejledning med forklaring af de enkelte funktioner. Bilag D indeholder kildekoden til beregningsmotoren i C++. Bilag E indeholder kildekoden til bindingerne mellem C++ og Mathematica.

1.5 Tak til...

En stor tak til Carsten Sørensen (Ph.D., lektor, Institut for Finansiering) for kompetent vejledning og gode tips ved udarbejdelsen af denne afhandling. Desuden tak til Niels Henrik Børjesson (cand.merc.(mat.), senioranalytiker, Nordea Markets), Kim Allan Klausen (cand.merc.FIR, analytiker, Danske Bank) og Kristina Birch (cand.merc.(mat.), Ph.D.-stud., statistikgruppen, HHK) for gennemlæsning og kommentering af afhandlingen før aflevering. Også tak til Rikke Brøndel (cand.act.) for hjælp med Björks bog. Og endelig tak til Jesper Andreasen (Ph.D., Head of Derivatives Product Development, Nordea Markets) og Jens Lund (Ph.D., senioranalytiker, Nordea Markets) for gode input omkring Heston- og SVJD-modellerne. Ansvaret for de tilbageværende fejl, udeladelser og mangler påhviler naturligvis udelukkende forfatteren.

2 Prisfastsættelse af optioner

Dette kapitel beskriver grundlaget for den fremherskende Black-Scholes-Merton metode til prisfastsættelse af afledte instrumenter, og udleder et prisudtryk for standard (*vanilla*) valutaoptioner. De centrale koncepter og vigtigste resultater vil blive præsenteret, men der vil ikke være en grundig matematisk udledning af formlerne. En sådan vil eksempelvis kunne findes i Björk (1998), som dette kapitel i høj grad er baseret på. En anden gennemgang findes i Baxter & Rennie (1996), mens lettere tilgængelige, men mindre matematisk stringente fremstillinger, kan findes i Hull (2000a) eller Wilmott (1998, kapitel 3, 4, 5 og 7).

Udledningen vil være struktureret som følger: Først beskrives det, hvordan det underliggende aktivs opførsel kan modelleres, og begrebet kontrakt (det afledte instrument) defineres. Herefter defineres et valutamarked bestående af valutaen samt de to risikofrie rentebærende aktiver. Efter en omskrivning af dette valutamarked til et kunstigt defineret marked med kun to aktiver introduceres begrebet en selvfinansierende strategi. Idéen er nu, at det er muligt at danne en selvfinansierende porteføljestrategi af det risikofri aktiv og det andet aktiv, der ved udløb er lig med kontraktens værdi. I et arbitragefrit marked må prisen på kontrakten derfor være lig anskaffelsesværdien for den selvfinansierende portefølje. Dette giver en differentialligning, som optionens værdi skal opfylde. Løses denne fås, at optionens værdi er lig med den forventede værdi under et andet sandsynlighedsmål. Dette resultat anvendes nu og føres tilbage til det oprindelige valutamarked. Den forventede værdi kan også skrives som et integrale, og ved hjælp af dette nåes der frem til en lukket formel for valutaoptionens pris.

2.1 Model for det underliggende aktiv

Da et afledt aktivs pris afhænger af dets underliggendes aktivs kurs, er det nyttigt at have en model for, hvordan denne udvikler sig over tid. Den mest udbredte model er den, der ligger bag Black-Scholes formlen. Her modelleres kursen som en geometrisk brownsk bevægelse.

Definition 1 *(Björk 1998, p. 53) En geometrisk brownsk bevægelse (GBM) er en process S, der følger dynamikken*

$$\begin{aligned} dS_t &= \mu S_t dt + \sigma S_t dW_t \\ S_0 &= s_0 \end{aligned} \tag{2.1}$$

Her er S_t kursen på det underliggende aktiv på tidspunkt t, s_0 er kursen observeret på tidspunkt 0 , μ er driften hvorved kursen forventes at stige og σ er volatiliteten, der siger nogen om, hvor meget kursen svinger. Definitionen er baseret på en standard brownsk bevægelse, også kaldet en Wiener-proces W_t (Björk 1998, p. 27). dW_t indfører stokastikken i ligningen og er den infinitisimale tilvækst i Wiener-processen. Løst sagt er denne et tilfældigt tal udtrukket fra en normalfordeling med middelværdi 0, således at der for alle $s < t$ gælder at $W_t - W_s$ er normalfordelt $N(0, \sqrt{t-s})$.

Processen skrives også ofte på formen

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t \tag{2.2}$$

Idet der er tale om forholdsmæssige tilvækster, vil S_t udvikle sig eksponentielt, og det er derfor interessant at se på, hvordan $\ln S_t$ udvikler sig. Hvis vi definerer $Z_t = \ln S_t$, fås vha. Itôs lemma[1]:

$$\begin{aligned} dZ_t &= \left(\mu - \frac{1}{2}\sigma^2\right) dt + \sigma dW_t \\ Z_0 &= \ln s_0 \end{aligned} \tag{2.3}$$

En model af denne type medfører, at kursen for det afledte aktiv på tidspunkt T vil være log-normalt fordelt:

Proposition 1 *Værdien S_T for en geometrisk brownsk proces S på tidspunkt $T > t$ vil være log-normal fordelt:*

$$\ln(S_T) \sim N\left(\ln(s_t) + \left(\mu - \frac{\sigma^2}{2}\right)(T-t), \sigma\sqrt{(T-t)}\right)$$

hvor s_t er værdien af S_t observeret på tidspunkt t.

Denne model kan beskrive kursudviklingen for mange aktiver f.eks. aktiekurser, valutakurser og visse råvarer nogenlunde godt. Der er dog en del mangler ved modellen, og der vil derfor i kapitel 2 blive introduceret en række udvidelser, så den kan beskrive virkeligheden mere realistisk.

Til brug for at beskrive forventede værdier er der brug for et begreb, der beskriver den information, som den stokastiske proces har genereret. Der følger derfor her en løs definition af begrebet filtrering.

[1] Björk (1998, p. 38)

Definition 2 *(Björk 1998, p. 29) En* ***filtrering*** $\mathcal{F}_t^X$ *for en proces* X *betegner den information, som* X *har genereret op til tidspunkt* t*. Hvis det på baggrund af stien* $\{X_s \mid 0 \leq s \leq t\}$ *er muligt at afgøre om en given hændelse* A *er sket eller ej, skrives* $A \in \mathcal{F}_t^X$*. En stokastisk process* X *siges at være* ***tilpasset*** (adapted) *til filtreringen* $\mathcal{F}^X$*, hvis* X_t *altid kan bestemmes ud fra den sti, som* X *har fulgt op til tidspunkt* t*.*

Denne definition kan bruges til at indføre en notation for den forventede værdi af den fremtidige kurs af S betinget af, at kursen nu er kendt.

Definition 3 *En* ***forventet værdi*** *for processen* S *på tidspunkt* T *givet den information, der er genereret af* S *op til tidspunkt* t*, skrives*

$$\mathbb{E}_{\mathbb{P}}(S_T \mid \mathcal{F}_t)$$

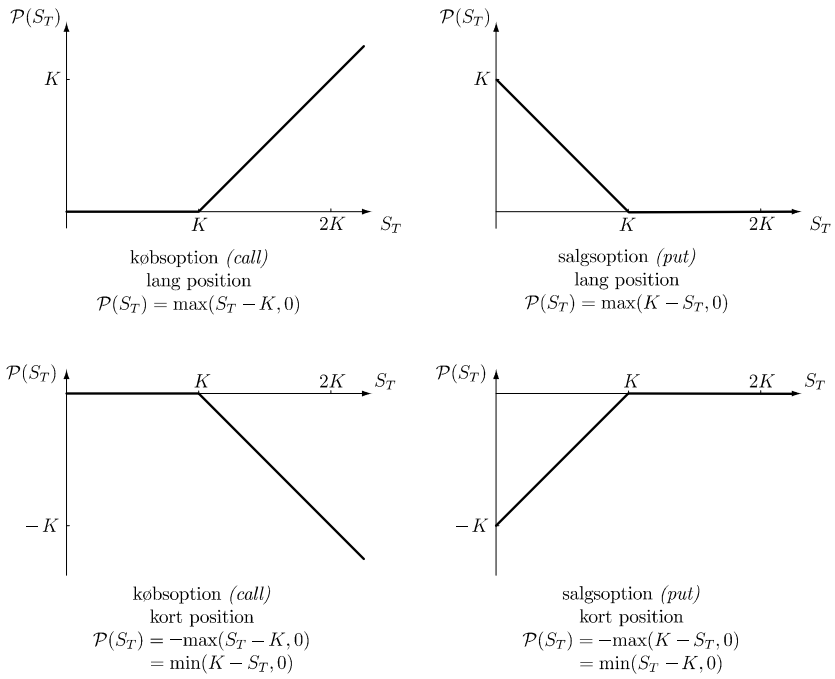
2.2 Optioner og afledte produkter

Formålet med denne afhandling er at prisfastsætte valutaoptioner, og det er således på sin plads at definere en sådan:

Definition 4 *En europæisk* ***købsoption*** (call option) *med aftalekurs* (exercise price) K *og løbetid* (time to maturity) T *på en underliggende valutakombination med kurs* S *defineres som en aftale melem to parter, der giver køberen en ret, men ikke en pligt, til at købe valutaen til aftalekursen* K *ved optionens udløb, tidspunkt* T*. En* ***salgsoption*** (put option) *defineres analogt som retten til at sælge. En* ***amerikansk option*** *giver (modsat en europæisk) mulighed for at indløse* (exercise) *optionen på alle tidspunkter op til og med tidspunkt* T*.*

For både købs- og salgsoptioner er der to parter i kontraktindgåelsen. Den der køber retten til at kunne indfri optionen, og den der sælger optionen, og således forpligter sig til at købe/sælge til aftalekursen ved evt. indfrielse. Man siger, at køberen har en **lang position** i optionen, mens sælgeren har en **kort position** i optionen.

Definition 5 *(Björk 1998, p. 78) Mere generelt defineres en* ***afledt fordring*** (contingent claim) *med* ***udløbstidpunkt*** T *som en stokastisk variabel* $\mathcal{X} \in \mathcal{F}_T^S$*. Såfremt* $\mathcal{X}$ *kan skrives på formen* $\mathcal{X} = \mathcal{P}(S_T)$ *kaldes den en* ***simpel fordring****. Funktionen* $\mathcal{P}$ *kaldes* ***kontraktfunktionen*** *eller* payoff-*funktionen.*



Figur 2.1: Afkastprofil ved udløb for korte og lange positioner af købs- og salgsoptioner.

En europæisk købsoption er altså en simpel fordring med kontraktfunktion $\mathcal{P}(S_T) = \max(S_T - K, 0)$. Kravet om at $\mathcal{X} \in \mathcal{F}_T^S$ betyder i praksis blot, at det rent faktisk skal være muligt at kunne bestemme, hvilket beløb fordringen udbetaler på tidspunkt T.

I figur 2.1 er kontraktfunktionerne for standardoptionerne illustreret.

2.3 Valutamarkedet

I Black-Scholes paradigmet for prisfastsættelse af afledte aktiver, der blev grundlagt med den banebrydende artikel Black & Scholes (1973), gøres der en række antagelser om, hvordan markedet ser ud. De vigtigste er:

- Der eksisterer et risikofrit aktiv, der har en konstant rente.
- Det underliggende aktiv følger en geometrisk brownsk bevægelse.
- Der er ingen arbitrage-muligheder i markedet.
- Der er ingen transaktionsomkostninger ved handel med det underliggende aktiv.

- Det er muligt at have en kort position (*short sales*).

Med udgangspunkt i disse antagelser, der ikke vil blive diskuteret nærmere[2], opbygges i det følgende en model for et valutamarked.

Det underliggende aktiv for valutaoptionen er valutakursen S_t, der er defineret som det beløb af indenlandsk valuta, som man på tidspunkt t skal betale for én enhed af den udenlandske valuta. Beløb, der haves i indenlandsk valuta kan investeres i et risikofrit aktiv til den konstante risikofri indenlandske rente r^d, og beløb i udenlandsk valuta forrentes tilsvarende med r^f. Til brug for modellen defineres:

Definition 6 *(Björk 1998, p. 76) Processen B repræsenterer prisen på et* ***risikofrit aktiv****, hvis dens dynamik kan beskrives med*

$$dB_t = rB_t dt$$

Antagelsen om eksistensen af et risikofrit aktiv retfærdiggøres som regel med, at der findes en risikofri obligation (*bond*), typisk en statsobligation, der giver renten r med sikkerhed.

Med antagelsen om at valutakursen følger en geometrisk brownsk bevægelse, kan modellen for det teoretiske marked opstilles.

Definition 7 *(Björk 1998, p. 167) Black-Scholes* ***modellen for et valutamarked*** *defineres som*

$$\begin{aligned} dB_t^d &= r^d B_t^d dt \\ dB_t^f &= r^f B_t^f dt \\ dS_t &= \mu S_t dt + \sigma S_t dW_t \end{aligned}$$

For at simplificere modellen udnyttes, at beløb investeret i den udenlandske obligation vil kunne omveksles til indenlandsk valuta, således at prisen på den udenlandske obligation B^f udtrykt i indenlandsk valuta kan skrives som

$$\widetilde{B}_t^f = S_t B_t^f \tag{2.4}$$

Ved brug af produktreglen (Baxter & Rennie 1996, p. 62) kan dette omskrives (fodtegnene er her udeladt):

$$\begin{aligned} & d\widetilde{B}^f \\ = \; & B^f dS + S dB^f = B^f(S\mu dt + S\sigma dW) + S(r^f B^f dt) \\ = \; & \widetilde{B}^f(\mu dt + \sigma dW) + r^f \widetilde{B}^f dt \\ = \; & \widetilde{B}^f(\mu + r^f)dt + \widetilde{B}^f \sigma dW \end{aligned}$$

[2]Beskrivelser af forsøg på at "reparere" på disse antagelser kan findes i Wilmott (1998, p. 75, kap. 19).

Lemma 2 *Black-Scholes modellen for et valutamarked kan alternativt skrives som*

$$\begin{aligned} dB_t^d &= r^d B_t^d dt \\ d\widetilde{B}_t^f &= \widetilde{B}_t^f(\mu + r^f)dt + \widetilde{B}_t^f \sigma dW \end{aligned} \tag{2.5}$$

Vi har hermed opnået, at det teoretisk definerede marked kun består af to aktiver, så beregningerne er lettere at håndtere.

2.4 Arbitragefrit marked

Opgaven er nu at finde frem til, hvad en rimelig pris på fordringen er. Som et værktøj til dette skal bruges begrebet en selvfinansierende strategi.

Definition 8 *(Björk 1998, p. 72) Givet en n-dimensionel prisproces* $S = (S_1, S_2, ..., S_n)$ *defineres en* ***portefølje-strategi*** (portfolio strategy) *som en vilkårlig* $\mathcal{F}_T^S$ *tilpasset proces* h_t, $t \geq 0$.

h_t er altså den n-dimensionelle proces, der angiver, hvor meget af hver enkelt aktiv, der besiddes på tidspunkt t.

Definition 9 *Givet en portefølje strategi* h *defineres en* ***værdiproces*** (value process) V^h *som*

$$V_t^h = \sum_{i=1}^{n} h_{i,t} S_{i,t}$$

eller med vektornotation

$$V_t^h = h_t S_t$$

V_t^h angiver således værdien på tidspunkt t af den portefølje, der vælges med portefølje-strategien h_t.

Definition 10 *En* ***selvfinansierende strategi*** (self-financing strategy) *er en portefølje-strategi* h*, der opfylder*

$$dV_t^h = \sum_{i=1}^{n} h_{i,t} dS_{i,t}$$

eller med vektornotation

$$dV_t^h = h_t dS_t$$

Dette betyder, at en strategi h netop vil være selvfinansierende, hvis salget af porteføljen h_t på tidspunkt t indbringer nøjagtigt det beløb, der skal bruges for at kunne købe porteføljen h_{t+dt} et infinitisimalt tidsrum dt senere.

Opgaven er nu at finde frem til en "rimelig" pris på valutaoptionen, forstået som den pris, der ikke giver arbitragemuligheder. Som nævnt i afsnit 2.3 er det jo netop en af de grundlæggende antagelser i Black-Scholes paradigmet, at der ikke er muligheder for arbitrage på markedet. **Prisprocessen for fordringen** (her optionen), der ønskes prisfastsat, benævnes $\Pi(t, \mathcal{X})$. Den arbitragefri pris skal således sikre, at der ikke findes arbitragemuligheder på markedet bestående af den indenlandske obligation B^d, den udenlandske obligation udtrykt i indenlandsk valuta $\widetilde{B}^f$ og fordringen selv Π.

Definition 11 *(Björk 1998, p. 80) Ved en* ***arbitragemulighed*** *på et marked forstås en selvfinansierende portefølje h, så*

$$V_0^h = 0$$
$$V_T^h > 0$$

med sandsynlighed 1. Markedet betegnes som ***arbitragefrit****, hvis der ikke findes arbitragemuligheder.*

En simpel anvendelse af kravet om ingen arbitrage fører frem til den såkaldte *put-call* paritet. På tidspunkt 0 dannes en portefølje bestående af en lang position af en *call*-option, Ke^{-rT} i kontanter, en kort position af en *put*-option og en kort position (et lån) af $e^{-r^f T}$ i den underliggende valuta. Disse positioner holdes i hele tidsrummet $0 \leq t \leq T$, og der er således tale om en konstant porteføljestrategi. Ved udløb (tidspunkt T) vil denne portefølje med sikkerhed have værdien $\max(S_T - K, 0) + K - \max(K - S_T, 0) - S_T$, idet vi forudsætter at det har kostet r^f i rente at låne valutaen. Dette giver $\max(S_T, K) - \max(K, S_T) = 0$. For at markedet nu kan være arbitragefrit må porteføljen dannet på tidspunkt 0 også have værdien 0. Hvis prisen på henholdsvis *call*- og *put*-optionen skrives c og p, betyder det at $c + Ke^{-r^d T} - p - s_0 e^{-r^f T} = 0$. Med en lille omskrivning har vi således følgende resultat:

Proposition 3 *(Björk 1998, p. 109) (Hull 2000b, p. 275) (****Put-call pariteten****) Givet to europæiske optioner, en call og en put, begge med aftalekurs K og tid til udløb T, vil følgende relation gælde i et marked uden arbitragemuligheder:*

$$p = Ke^{-r^d T} + c - s_0 e^{-r^f T}$$

Et andet centralt resultat siger, at alle selvfinansierende porteføljer nødvendigvis må give den risikofri rente:

Proposition 4 *(Björk 1998, p. 80) Hvis der findes en selvfinansierende porteføljе h, således at værdiprocessen følger*

$$dV_t^h = k_t V_t^h dt \tag{2.6}$$

så vil denne under antagelse af at markedet er arbitragefrit være lig

$$dV_t^h = rV_t^h dt$$

En selvfinansierende strategi skal altså give samme afkast som den risikofri rente, ellers vil der findes arbitragemuligheder.

2.5 Differentialligningen ved et arbitragefrit marked

På vej mod et funktionsudtryk for prisen på en simpel valutafordring præsenteres nu det generelle resultat for et marked med to aktiver, nemlig det underliggende stokastiske aktiv X_t og den risikofrie obligation B_t:

$$\begin{aligned} dB_t &= rB_t dt \\ dX_t &= X_t \mu dt + X_t \sigma W_t \end{aligned} \tag{2.7}$$

Vi skal finde en prisningsfunktion $F(t,x) = \Pi(t, \mathcal{X})$, der udelukker arbitrage på markedet bestående af X, B og Π. Fremgangsmåden er at danne en selvfinansierende porteføljestrategi, der så giver den risikofri rente r.

Det kan vises, at ved at lade en selvfinansierende portefølje h bestå af en lang position af $\frac{\delta F}{\delta x}$ i det underliggende aktiv og en kort position af fordringen selv fås en værdiproces V^h, der ikke indeholder et stokastisk led og altså er på formen (2.6). Ved at benytte resultatet fra proposition 4 om, at $dV_t^h = rV_t^h dt$ kan der nås frem til følgende differentialligning:

Theorem 5 *(Björk 1998, p. 84)* ***(Black-Scholes-Merton differentialligningen)*** *Givet et Black-Scholes marked (2.7) og en simpel fordring $\mathcal{X} = \mathcal{P}(X_T)$, vil den eneste prisfunktion $F(t,x)$, der opfylder kravet om et arbitragefrit marked, være givet ved løsningen på følgende differentialligning med tilhørende grænsebetingelse*

$$\begin{aligned} \frac{\delta F}{\delta t} + xr\frac{\delta F}{\delta x} + \frac{1}{2}x^2\sigma^2\frac{\delta^2 F}{\delta x^2} - rF = 0 \\ F(T,x) = \mathcal{P}(x) \end{aligned} \tag{2.8}$$

Udtrykket $\frac{\delta F}{\delta x}$, der bruges i dannelsen af den selvfinansierende porteføljе, kaldes også Δ (**delta**), og er det mest centrale nøgletal for en option. Det kan nemlig bruges til afdækning af risiko via såkaldt *delta hedging*, der er meget tæt relateret til argumentet, der bruges til udledning af BSM differentialligningen.

Hvis en investor har solgt en købsoption, kan han ved at købe Δ af det underliggende aktiv og evt. låne manglende eller placere overskydende penge i banken til renten r, afdække sin position i et kort tidsinterval. Hvis han herefter kontinuert udregner et nyt Δ og justerer sin beholdning af aktivet herefter, replikerer han nøjagtigt optionen. Hvis der ved udløb gælder $S_T > K$ (optionen er *in-the-money*), vil han netop have ét af det underliggende aktiv klar til levering, og en gæld på K, der bliver udlignet ved modtagelsen af aftalekursen fra optionskøberen. Hvis der omvendt gælder $S_T < K$ (optionen er *out-of-the-money*), vil han ikke eje noget af aktivet, og heller ikke have hverken gæld eller penge i banken. Hvis $S_T = K$ (optionen er *at-the-money*) vil han eje en halv af det underliggende aktiv, og have en gæld på $\frac{1}{2}K$, så de to beløb nøjagtigt udligner hinanden.

Denne perfekte afdækning forudsætter, at investoren kan handle kontinuert og uden transaktionsomkostninger. Dette er naturligvis ikke muligt, men hvis *delta-hedgingen* gennemføres med diskrete tidsintervaller opnås en afdækning, der tilnærmelsesvis er perfekt.

Læg mærke til, at variablen μ er faldet ud. Dette er ganske overraskende og betyder i praksis, at en afledt fordrings pris ikke afhænger af det forventede afkast for det underliggende aktiv – kun volatiliteten har betydning. Dette hænger sammen med, at det er muligt at lave en perfekt hedging, og at fordringen prisfastsættes relativt til det underliggende aktiv.

2.6 Risikoneutral prisfastsættelse

Med udgangspunkt i ligningen (2.8) skal der nu findes et prisudtryk for optionen. Dette opnås ved at introducere det risikoneutrale sandsynlighedsmål.

I denne udledning er begrebet en martingalproces centralt.

Definition 12 *(Björk 1998, p. 33) (Baxter & Rennie 1996, p. 74) En stokastisk proces M_t kaldes en martingal under sandsynlighedsmålet $\mathbb{P}$, såfremt der gælder*

$$\mathbb{E}_{\mathbb{P}}(M_t \mid \mathcal{F}_s) = M_s, \ \forall s \leq t$$
$$\mathbb{E}_{\mathbb{P}}(|M_t|) < \infty, \ \forall t$$

Her betyder første linie, at forventningen til en fremtidig værdi af processen er lig den nuværende værdi. Anden linie er en mere teknisk betingelse, der siger, at processen har begrænset varians.

Proposition 6 *(Björk 1998, p. 60, p. 86) (**Feynman-Kač**) Såfremt F er løsning til (2.8)*

$$\frac{\delta F}{\delta t} + xr\frac{\delta F}{\delta x} + \frac{1}{2}x^2\sigma^2\frac{\delta^2 F}{\delta x^2} - rF = 0$$
$$F(T,x) = \mathcal{P}(x)$$

kan F skrives som

$$F(t,x) = e^{-r(T-t)}\mathbb{E}(\mathcal{P}(Y_T) \mid \mathcal{F}_t) \tag{2.9}$$

hvor Y følger

$$dY = Yrdt + Y\sigma dW$$

Det ses nu, at $\mathcal{X} = \mathcal{P}(X_T)$ kan prisfastsættes vha. en proces Y, der er på samme form som X, men blot har udskiftet driftleddet μ med den risikofri rente r. Det kunne derfor være fristende simpelthen at påstå, at X følger dynamikken givet ved Y, og glemme den oprindelige X-dynamik. Det kan faktisk også gøres med fornuft, hvis blot processerne holdes begrebsmæssigt og matematikteknisk adskilt. Den oprindelige eller objektive proces X er defineret under det objektive sandsynlighedsmål $\mathbb{P}$. Det nye mål, der svarer til Y, kaldes det risikoneutrale sandsynlighedsmål $\mathbb{Q}$. Således siges det, at X har $\mathbb{Q}$-dynamikken

$$dX = Xrdt + X\sigma d\widetilde{W}$$

Tilden ($\widetilde{}$) over W angiver, at Wiener-processen er en $\mathbb{Q}$-Wiener-proces. Denne konvention vil blive brugt i resten afhandlingen.

Det interessante er nu, at sandsynlighedsmålet $\mathbb{Q}$ netop gør at den tilbagediskonterede proces $\frac{X_t}{B_t}$ viser sig at være en $\mathbb{Q}$-martingal. Dette er også forklaringen på, at $\mathbb{Q}$ udover at blive kaldt det risikoneutrale sandsynlighedsmål også kaldes det ækvivalente martingalmål. At målet er ækvivalent dækker over, at $\mathbb{Q}$ har positiv sandsynlighed på samme mængde som $\mathbb{P}$.

Proposition 7 *(Björk 1998, p. 87) (**Risikoneutral værdifastsættelse**) I Black-Scholes modellen, har prisprocessen Π_t for et vilkårligt handlet aktiv, den egenskab, at den tilbagediskonterede prisproces*

$$Z_t = \frac{\Pi_t}{B_t}$$

er en martingal under $\mathbb{Q}$.

At det kan lade sig gøre at prisfastsætte derivaterne ved at bruge det risikoneutrale sandsynlighedsmål skyldes, at det under Black-Scholes modellen er muligt at danne en selvfinansierende strategi, der replikerer derivatet. Denne portefølje må netop have den risikofri rente som afkast, og det gør, at der kan "lades som om", at agenterne er risikoneutrale. Det er imidlertid på ingen måde nødvendigt, at agenterne på markedet rent faktisk er risikoneutrale. Det er kun rent regneteknisk, at der prisfastsættes "som om" agenterne er det.

Med udgangspunkt i den omskrevne model for valutamarkedet fra lemma 2, følger det af sætning 5 og proposition 6, at $\mathbb{Q}$-dynamikken for $\widetilde{B}^f$ er givet ved

$$d\widetilde{B}^f = r^d \widetilde{B}^f dt + \widetilde{B}^f \sigma d\widetilde{W} \tag{2.10}$$

$\widetilde{B}^f$ er imidlertid et konstrueret aktiv, der ikke handles på markedet, så det er derfor nødvendigt med et udtryk for processen S_t. Til dette formål er der brug for en fler-dimensionel version af Itôs lemma.

Theorem 8 *(Björk 1998, p. 45) (**Itôs formel**) Givet en n-dimensionel proces X vil processen givet ved funktionen $f(t, X_t)$ have det stokastiske differentiale*

$$df(t, X_t) = \frac{\delta f}{\delta t}dt + \sum_{i=1}^{n} \frac{\delta f}{\delta x_i} dX_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\delta^2 f}{\delta x_i \delta x_j} dX_i dX_j$$

hvor følgende formelle multiplikationsregler er gældende

$$\begin{aligned} (dt)^2 &= 0 \\ dt \cdot dW &= 0 \\ (dW_i)^2 &= dt \\ dW_i \cdot dW_j &= 0, i \neq j \end{aligned}$$

Fra (2.4) har vi pr. definition $S_t = \frac{\widetilde{B}_t^f}{B_t^f}$. Hvis vi nu sætter $f(t, X_1, X_2) = S_t$, $X_1 = \widetilde{B}_t^f$ og $X_2 = B_t^f$, kan vi bruge ovenstående formel.

Idet $\frac{\delta S_t}{\delta t} = 0, \frac{\delta S_t}{\delta \widetilde{B}^f} = \frac{1}{B^f}, \frac{\delta S_t}{\delta B^f} = -\frac{\widetilde{B}^f}{B^{f2}}, \frac{\delta S_t}{\delta B^f} = 0$, og at de resterende 3 led reduceres til enten $dt \cdot dW = 0$ eller $(dt)^2 = 0$, fordi dB^f ikke indeholder et dW led, får vi

$$dS_t = \frac{1}{B^f} d\widetilde{B}^f - \frac{\widetilde{B}^f}{B^{f2}} dB^f$$

Sætter vi nu (2.10) og $dB^f = r^f B^f dt$ ind i denne formel, fås

$$dS_t = \frac{1}{B^f}(r^d \widetilde{B}^f dt + \widetilde{B}^f \sigma d\widetilde{W}) - \frac{\widetilde{B}^f}{B^{f2}}(r^f B^f dt) \tag{2.11}$$

$$
\begin{aligned}
&= \frac{\widetilde{B}^f}{B^f}(r^d dt + \sigma d\widetilde{W}) - \frac{\widetilde{B}^f}{B^f} r^f dt \\
&= S_t(r^d dt + \sigma d\widetilde{W}) - S_t r^f dt \\
&= (r^d - r^f) S_t dt + S_t \sigma d\widetilde{W}
\end{aligned}
$$

Og vi er herved nået frem til dynamikken for valutakursen S_t under det risikoneutrale sandsynlighedsmål $\mathbb{Q}$, og kan således bruge proposition 6 til at konkludere, at prisen på en valutafordring kan findes med formlen

$$F(t,s) = e^{-r^d(T-t)} \mathbb{E}_{\mathbb{Q}}(\mathcal{P}(S_T)|\mathcal{F}_t)$$

hvor S_t har $\mathbb{Q}$-dynamikken

$$S_t = (r^d - r^f) S_t dt + S_t \sigma d\widetilde{W}$$

Dette kan også udtrykkes på integraleform som

$$F(t,s) = e^{-r^d(T-t)} \int_{-\infty}^{\infty} \mathcal{P}(S_T) \mathbb{Q}(S_T) dS_T \tag{2.12}$$

hvor $\mathbb{Q}(S_T)$ er punktsandsynligheden for, at kursen ved udløb er S_T.

Ved hjælp af denne formel kan $F(t,s)$ for en vilkårlig $\mathcal{P}$ i princippet findes ved hjælp af numerisk integration, som den, der introduceres i kapitel 4. Ved at se på standard købs- og salgsoptioner og udnytte, at udløbskursen S_T er log-normalfordelt, kan man imidlertid nå frem til en lukket formel for prisen.

2.7 Analytisk formel

Proposition 9 *(Björk 1998, p. 90, p. 170) (**Black-Scholes formlen**) Prisen for en europæisk købsoption under Black-Scholes valutamarked-modellen er givet ved*

$$c(t,s) = se^{-r^f(T-t)}\Phi(d_1) - e^{-r^d(T-t)}K\Phi(d_2) \tag{2.13}$$

hvor

$$d_1(t,s) = \frac{1}{\sigma\sqrt{T-t}} \left(\ln\left(\frac{s}{K}\right) + \left(r^d - r^f + \frac{1}{2}\sigma^2\right)(T-t) \right)$$

$$d_2(t,s) = d_1(t,s) - \sigma\sqrt{T-t}$$

Ved brug af put-call pariteten $p(t,s) = Ke^{-r^d(T-t)} + c(t,s) - s_t e^{-r^f(T-t)}$, kan (2.13) let omskrives, så prisen for en salgsoption har formlen

$$p(t,s) = se^{-r^f(T-t)}(\Phi(d_1) - 1) - e^{-r^d(T-t)}K(\Phi(d_2) - 1) \tag{2.14}$$

En alternativ formulering af formlen kan fås ved at tage udgangspunkt i (2.9) (Bates 1996, p. 75). En købsoption vil således have prisen

$$\begin{aligned} c(t,s) &= e^{-r^d(T-t)}\mathbb{E}_{\mathbb{Q}}(\max(S_T-K,0)\mid\mathcal{F}_t) \\ &= e^{-r^d(T-t)}\left(\int_K^\infty S_T\mathbb{Q}(S_T\mid\mathcal{F}_t)dS_T - K\int_K^\infty \mathbb{Q}(S_T\mid\mathcal{F}_t)dS_T\right) \end{aligned}$$

$\mathbb{Q}(S_T \mid \mathcal{F}_t)$ er tæthedsfunktionen for værdien af S på tidspunkt T under sandsynlighedsmålet $\mathbb{Q}$ givet informationen $\mathcal{F}_t$ (dvs. i praksis kursen s). Vi laver en lille omskrivning (og udelader $\mathcal{F}_t$'erne):

$$c(t,s) = e^{-r^d(T-t)}\left(\mathbb{E}_{\mathbb{Q}}(S_T)\int_K^\infty \frac{1}{\mathbb{E}_{\mathbb{Q}}(S_T)}S_T\mathbb{Q}(S_T)dS_T - K\int_K^\infty \mathbb{Q}(S_T)dS_T\right)$$

Vi definerer nu et par variable: $F = \mathbb{E}_{\mathbb{Q}}(S_T) = S_t e^{(r^d-r^f)}$, der er *forward* prisen på valutaen. $P_2 = \int_X^\infty \mathbb{Q}(S_T)dS_T = \mathbb{Q}(X > S_T)$, der er sandsynligheden for, at optionen ender *in-the-money* og dermed indfries. Dette svarer til den forventede værdi ved udløb af en binær option, der udbetaler 1, hvis $X > S_T$, og 0 ellers. $P_1 = \int_K^\infty \frac{1}{\mathbb{E}_{\mathbb{Q}}(S_T)}S_T\mathbb{Q}(S_T)dS_T$, der kan opfattes som en sandsynlighed, idet integranden er ikke-negativ og integralet over $[0;\infty[$ er lig 1. Samtidigt har P_1 den fortolkning, at den er lig den forventede værdi ved udløb af en binær option, der betaler S_T, hvis $S_T > K$ og 0 ellers. Med disse definitioner kommer vi frem til:

$$c(t,s) = e^{-r^d(T-t)}(FP_1 - KP_2)$$

Fra (2.13) kan det let udledes, at i en Black-Scholes verden er $P_1 = \Phi(d_1)$ og $P_2 = \Phi(d_2)$. Som vi skal se senere, vil disse værdier være anderledes i andre modeller. Også her kan *put-call*-pariteten bruges, og vi får

$$p(t,s) = e^{-r^d(T-t)}(F(P_1-1) - K(P_2-1))$$

I bilaget afsnit C.2 er det vist, hvordan man kan bruge det medfølgende program til at udregne prisen vha. denne formel.

2.8 Sammenhæng med dividendebærende aktie

Ved omskrivningen af valutamarkedet i (2.11) blev det fundet, at valutakursen S_t under det risikoneutrale sandsynlighedsmål følger dynamikken $dS_t =$

$(r^d - r^f)S_t dt + \sigma S_t d\widetilde{W}$. Dette virker intuitivt rigtigt, idet pengene alternativt kan investeres i det riskofrie indenlandske aktiv og give renten r^d. Ved investering i valuta vil der blive forlangt det samme afkast. Den købte valuta behøver imidlertid ikke blot ligge i en skuffe, men kan investeres til den udenlandske rente r^f. Således må kursen udtrykt i indenlandsk valuta stige med renten $r^d - r^f$.

Med udgangspunkt i overnævnte dynamik og ved brug af proposition 6 fås Black-Scholes-Merton differentialligningen for valutamarkedet:

$$\begin{aligned} \frac{\delta F}{\delta t} + s(r^d - r^f)\frac{\delta F}{\delta s} + \frac{1}{2}s^2\sigma^2\frac{\delta^2 F}{\delta s^2} - r^d F &= 0 \\ F(T,s) &= \mathcal{P}(s) \end{aligned} \tag{2.15}$$

Hvis der istedet betragtes en aktie, der udbetaler en kontinuert dividende som en konstant andel q af aktien, kan dynamikken for aktiens kurs S_t og dividenden D_t skrives som

$$\begin{aligned} dS_t &= \mu S_t dt + \sigma S_t dW_t \\ dD_t &= qS_t dt \end{aligned}$$

Den indtægt, der modtages ved at investere i aktien, kan skrives som *gain* processen G_t[3]

$$\begin{aligned} dG_t &= dS_t + dD_t \\ &= (\mu + q)S_t dt + \sigma S_t dW_t \end{aligned}$$

Der antages desuden som tidligere eksistensen af en risikofri obligation B_t

$$dB_t = rB_t dt$$

Ved at bruge argumenter om at markedet er arbitragefrit i lighed med dem, der er nødvendige for udledningen af proposition 5, fås følgende differentialligning for prisfastsættelse af en aktie S_t[4]:

$$\begin{aligned} \frac{\delta F}{\delta t} + s(r - q)\frac{\delta F}{\delta s} + \frac{1}{2}s^2\sigma^2\frac{\delta^2 F}{\delta s^2} - rF &= 0 \\ F(T,s) &= \mathcal{P}(s) \end{aligned} \tag{2.16}$$

Det ses, at ligningen (2.16) for en dividendebærende aktie er på helt samme form, som ligningen (2.15) for valuta, hvor r^f blot er udskiftet med q

[3] Björk (1998, p. 161)
[4] Björk (1998, p. 162)

og r^d udskiftet med r. Prisningsresultaterne fra afsnit 2.7 vil således kunne benyttes ved en simpel udskiftning af variabelnavnene. Ligeledes vil teorien og programmet, der bliver beskrevet i det følgende, kunne benyttes såvel på valutaoptioner som på aktieoptioner.

3 Udvidelser til Black-Scholes

I dette kapitel behandles forskellige udvidelser til Black-Scholes modellen, der giver mulighed for at beskrive de observerede markedspriser bedre. Desuden refereres resultaterne fra en række empiriske undersøgelser, der understøtter valget af Bates' SVJD som en egnet model. Først introduceres volatilitetssmilet, der påviser, at Black-Scholes-formlen ikke prisfastsætter optioner i overensstemmelse med markedet. Herefter præsenteres forskellige muligheder for at udvide BS-modellen med stokastisk volatilitet samt en mulig måde at tilføje spring til den stokastiske proces. Disse to udvidelser kombineres til SVJD-modellen. Til sidst behandles nogle praktiske overvejelser, der kan være en grund til, at SVJD-modellen kun har fået begrænset udbredelse i praksis.

3.1 Volatilitetssmilet

Black-Scholes formlen er stadig udgangspunktet i prisfastsættelsen af optioner, men adskillige studier viser, at markedets priser afviger systematisk fra Black-Scholes priserne. Samtidigt er den geometriske brownske bevægelse ikke i stand til at forklare tidsrækkerne for prisen på det underliggende aktiv. Modellen er således ikke tilfredsstillende, og man må søge efter en, der er bedre.

Et standardværktøj til at påvise inkonsistenserne mellem markedspriser og BS-priser er det såkaldte volatilitetssmil. Til dette formål bruges den implicitte volatilitet, der er den volatilitet, der indsat i BS-formlen giver markedsprisen.

Definition 13 *Den **implicitte volatilitet** for et givet instrument og en given markedspris F_m er den volatilitet σ_{imp} for hvilket det gælder at*

$$F(S, t; \sigma_{imp}, r^d, r^f; K, T) - V_m = 0$$

I praksis findes værdien ved at holde alle andre parametre end σ_{imp} konstante og bruge en numerisk rodfindingsalgoritme. Idet prisen er en voksende funktion af volatiliteten vil løsningen være entydig, hvis den findes.

I afsnit C.3 er det demonstreret, hvordan σ_{imp} kan udregnes med det medfølgende program.

Det er ligegyldigt om der bruges priser for *put*-optioner eller *call*-optioner, da *put-call*-pariteteten gælder både for BS-priser og markedspriser. Således vil afvigelsen fra markedsprisen være den samme for *put* og *call* ($p_{bs} - p_m = c_{bs} - c_m$), og den σ_{imp}, der får afvigelsen til at være 0 for den ene type, vil også give 0 for den anden (Hull 2000b, p. 436).

Hvis den implicitte volatilitet σ_{imp} findes for handlede valutaoptioner for forskellige aftalekurser K og disse plottes op mod hinanden, fås i de fleste tilfælde en U-formet graf (se figur 4.12, p. 60), der kaldes volatilitetssmilet. Hvis markedspriserne stemte overens med Black-Scholes modellen ville "smilet" være en vandret linie, men da dette reelt aldrig observeres, kan man konkludere, at markedet ikke er enigt i, at Black-Scholes modellen beskriver virkeligheden korrekt. Volatilitetssmilet for forskellige aktivtyper (aktieindeks, råvarer, valuta osv.) vil have forskellig form og vil f.eks. for aktier typisk mere have lighed med et skævt smil (*smirk*) (Hull 2000b, p. 437 ff.). I denne afhandling vil begrebet volatilitetssmil imidlertid henvise til den funktionelle sammenhæng mellem aftalekurs og implicit volatilitet uanset dens form.

Eksistensen af volatilitetssmilet påviser, at BS-modellen ikke er fuldkommen. Samtidig kan smilet imidlertid bruges til at justere de priser man finder med BS-formlen (Wilmott 1998). Teknikken er, at man finder en række optioner med samme karakteristika, men med forskellig aftalekurs og ud fra disse konstruerer volatilitetssmilet. Herefter bruges der en interpolationsteknik, så smilet ikke består af punkter, men af en sammenhængende kurve. Nu kan en option af lignende type prisfastsættes ved hjælp af BS-formlen med den volatilitet, der kan aflæses på volatilitetssmilet.

For mange optionstyper vil volatilitetssmilet være forskelligt for forskellige restløbetider, og man kan derfor udvide smilet $\sigma_{imp}(K)$ til også at afhænge af T og derved få en volatilitetsoverflade $\sigma_{imp}(K, T)$ (Wilmott 1998, p. 291).

3.2 Underliggende stokastiske proces

Da observationen af volatilitetssmilet afslører, at Black-Scholes modellen er utilstrækkelig til at beskrive prisdannelsen, er det naturligt at prøve at ændre eller udvide den til en ny model, der kan forklare volatilitetssmilet. Inden for det fremherskende paradigme til prisfastsættelse af derivater tages der udgangspunkt i at kursen på det underliggende aktiv følger en stokastisk

differentialligning (SDE). I dette paradigme bliver opgaven så at finde en SDE, der giver sandsynlighedsfordelinger, der stemmer overens med de observerede optionspriser, og hvor den genererede proces stemmer overens med de observerede tidsrækker for det underliggende aktiv (Bates 1996, p. 69). Flere empiriske undersøgelser viser, at fordelingerne for kursen på det underliggende aktiv har tykke haler og er skæve. Opgaven er således at finde en process, der er plausibel og kan fange disse karakteristika.

3.3 Stokastisk volatilitet

En række forsøg på at forbedre Black-Scholes-modellen har taget udgangspunkt i, at selv en simpel analyse vil vise, at volatiliteten σ for kursen ikke er konstant, som det antages i Black-Scholes formlen. Det er uden de store modifikationer muligt at gøre volatiliteten tidsafhængig (Wilmott 1998, p. 121), men det kan være svært at vide, hvilken deterministisk funktion denne skal følge. Derfor er det mere realistisk at antage, at volatiliteten i lighed med kursen S_t følger en stokastisk proces. Variansen V_t lades derfor følge en stokastisk proces og udsvingene for S_t lades være bestemt af den nu stokastiske volatilitet $\sqrt{V_t}$ (svarende til σ):

$$dS_t = \mu dt + \sqrt{V_t} dW_t$$

Der er flere muligheder for, hvilken process variansen V_t. I det følgende vil 3 af de mest prominente modeller blive beskrevet, men der findes naturligvis andre (f.eks. $V_t^{\frac{3}{2}}$-modellen eller modellen fra Stein & Stein (1991)). Alle 3 modeller vil med de rigtige parametervalg resultere i leptokurtiske sandsynlighedsfordelinger og et valg af en korrelation ρ forskellig fra nul vil resultere i en skæv fordeling.

3.3.1 Hestons kvadratrodsmodel

En model, der blev introduceret af Heston (1993) og er den, der vil blive bygget videre på i denne afhandling, er den såkaldte kvadratrodsmodel[1]

$$dV_t = (\alpha - \beta V_t)dt + \sigma_v \sqrt{V_t} dW_{v,t}$$

Denne proces er *mean-reverting*, hvilket betyder, at V_t vil blive trukket tilbage mod et naturligt leje kaldet ligevægts-niveauet (*steady state level*),

[1] Her er brugt samme symboler som i Bates (1996). Det kan bemærkes, at den stokastiske proces er på samme form som den, der bruges i CIR rentemodellen.

der er givet ved $\frac{\alpha}{\beta}$. Volatiliteten $\sqrt{V_t}$ vil altså variere omkring værdien $\sqrt{\frac{\alpha}{\beta}}$, der således er sammenlignelig med σ i Black-Scholes modellen. $\beta > 0$ kaldes *mean reversion rate* og angiver hastigheden for tilbagevenden til ligevægtsniveauet. Udtrykket $\frac{\ln 2}{\beta}$ kaldes halveringstiden for volatilitetschok *(half-life of volatility shocks)* og betegner den tid, det i gennemsnit tager før en afvigelse fra ligevægts-niveauet er bragt halvvejs tilbage til ligevægten. Hvis tidsenheden er et år, vil $\beta = 1.3$ således betyde, at der går ca. 6.4 måneder før V er nået halvvejs tilbage mod $\frac{\alpha}{\beta}$.

Omskrives V-processen til $dV_t = \beta(\frac{\alpha}{\beta} - V_t)dt + \sigma_v\sqrt{V_t}dW_{v,t}$ bliver det tydeligere, hvorfor $\sqrt{\frac{\alpha}{\beta}}$ betegner ligevægten for $\sqrt{V_t}$, og hvorfor β betegner trækkraften. Hvis vi et øjeblik ser bort fra det stokastiske led, så vil $V_t > \frac{\alpha}{\beta}$ medføre, at dV bliver negativ og trækker V_t ned mod $\frac{\alpha}{\beta}$ med en hastighed afhængig af størrelsen af β. Hvis omvendt $V_t < \frac{\alpha}{\beta}$, vil dV blive positiv og trække V_t opad.

σ_v betegner volatilitetens volatilitet, der angiver, hvor meget støjdene fra Wiener-processen W indvirker på V. Desuden skal den nuværende volatilitet $\sqrt{V_0}$ bruges som input i modellen.

Endelig gives der i modellen mulighed for, at volatiliteten samvarierer med det underliggende aktivs kurs med korrelationskoefficienten $\text{corr}(W, W_v) = \rho$. For aktiekurser vil ρ typisk være negativ. Dette virker intuitivt rigtigt, da det betyder, at kursen bliver mere volatil, når den er på vej ned, eller omvendt at en mere volatil (usikker) kurs vil få den til at falde.

En stor fordel ved kvadratrodsmodellen er, at Heston (1993) har udviklet en metode, der fører frem til lukkede formeludtryk for både optionspriser og tæthedsfunktionen for fordelingen af det underliggende aktivs kurs. Den eneste numeriske beregning, der skal foretages er en integration af en funktion, der udelukkende består af elementære funktioner. Dette ser vi på i flere detaljer i kapitel 4.

3.3.2 GARCH diffusion / Hull-White

En anden mulig model er at tage udgangspunkt i den store erfaring, der er opsamlet omkring tidsrækkeanalyse af volatilitet med den såkaldte GARCH metode (Hull 2000b, p. 368). Denne bruges til studier af diskrete tidsrækker, men kan vises at have den kontinuerte grænse kaldet GARCH diffusionen (Lewis 2000, p. 3), (Wilmott 1998, p. 303)

$$dV_t = (\omega - \theta V_t)dt + \xi V_t dW_{v,t}$$

Som det ses er den eneste forskel på GARCH diffusion modellen og kvadratrodsmodellen, at koefficienten foran det brownske tilvækstled $dW_{v,t}$ in-

deholder V_t i stedet for $\sqrt{V_t}$. Dette bevirker, at estimation af parametrene ξ og σ_v på baggrund af den samme tidsrække vil være meget forskellige i de to modeller. De øvrige parametre har på trods af andre symboler samme fortolkning som i kvadratrodsmodellen.

Den samme proces studeres i Hull & White (1987, p. 289), hvor der gives en specielt tilpasset Monte Carlo metode til at finde optionsprisen i dette tilfælde. Hvis $\omega = 0$ kan prisen tilnærmes med en simplere Taylor-udvikling. Der gives imidlertid ikke mulighed for at W og W_v er korrelerede. Desuden er der først for nyligt fundet en analytisk løsning (Heston & Nandi 2000), hvilket har været en ulempe for modellens brug i praksis. Til gengæld findes der et stort antal af færdige programpakker, hvormed man umiddelbart kan estimere parametrene i GARCH modellen.

3.3.3 Log-varians modellen

En tredje model er log-varians modellen, der stemmer godt overens med standard modeller for stokastisk volatilitet i diskret tid og EGARCH modellen (Andersen, Benzoni & Lund 2002, p. 1243, 1245).

$$d \ln V_t = (\alpha - \beta \ln V_t)dt + \eta dW_{v,t}$$

Processen er *mean-reverting* på samme måde som kvadratrodsmodellen, hvor det her blot er $\ln V_t$, der søger tilbage mod ligevægts-niveauet. Umiddelbart ser det ud til, at størrelsen af det stokastiske led ikke afhænger af V_t, men ved omskrivning med Itôs lemma kan det vises, at V_t indgår i det stokastiske led for processen V_t (Lewis 2000, p. 5). Der findes endnu ikke en lukket formel for optionsprisfastsættelse under log-varians modellen.

3.4 Spring

Med stokastisk volatilitet kan de tykke haler i de observerede fordelinger for kurser forklares. Der er imidlertid en række empiriske undersøgelser, der tyder på, at modeller, hvor der samtidigt inkluderes spring i den stokastiske proces, bedre kan forklare kursudviklingen.

Jorion (1988) bruger en *maximum-likelihood* estimationsmetode til at sammenligne forskellige processers evne til at forklare sandsynlighedsfordelingerne for en række forskellige valutakryds (herunder \$/DM) og et markedsindeks bestående af alle NYSE og AMEX aktier. Han sammenligner processer, der indeholder et ARCH element og et Poisson springelement eller begge. (ARCH processen har egenskaber, der ligner de stokastisk volatilitetsprocesser, der blev introduceret i forrige afsnit.)

Han kommer frem til (p. 435), at der, selv efter at der med ARCH processen er taget højde for, at variansen varierer over tid, opnås en bedre forklaringsgrad ved at tage springkomponenten med i modellen. Dette viser sig i højere grad i valutamarkederne end i aktiemarkeder, hvilket kan skyldes, at der i regimer med mere eller mindre fast valutakurs opstår spring, når centralbanken de- eller revaluerer.

Bates (1996) introducerer en udvidelse til Hestons kvadratrodsmodel, der udover stokastisk volatilitet, indeholder en spring-komponent (kaldet SVJD-modellen, der beskrives i afsnit 3.5). Herefter undersøger han, hvilken prisproces for valutakrydset \$/DM, der er implicit givet ved valutaoptionspriser observeret fra 1984 til 1991. Han konkluderer, at modellen inkl. spring fitter bedre end undermodellerne (p. 87). Han analyser også, hvor godt tidsrækken for prisen på valutafutures stemmer overens med modellen givet de implicitte parametre. Her når han dog frem til, at modellen med spring ikke er signifikant bedre end modellen uden (p. 99). Denne sidste observation taler i princippet mod brugen af SVJD-modellen.

Andersen et al. (2002) estimerer parametre for en lang række modeller med og uden stokastisk volatilitet og spring (herunder Black-Scholes, kvadratrodsmodellen, log-varians modellen og SVJD-modellen) for en tidsrække for det amerikanske S&P 500 aktieindeks. I sammenligningen af modellerne, når de frem til, at "både stokastisk volatilitet og diskrete springkomponenter er kritiske ingredienser af den data-genererende mekanisme" (p. 1241).

Bakshi, Cao & Chen (1997) studerer den interne konsistens mellem de parametre, der er implicitte i optionspriserne, og den underliggende tidsserie for en række modeller. De konkluderer, at det for intern konsistens og til prisfastsættelsesformål, er vigtigt at inkludere både stokastisk volatilitet og spring i modellen.

Der er således en række empiriske tegn på, at det er en fordel at inkludere en springkomponent i modellerne. En af de første konkrete forslag til en model med en spring findes i Merton (1976). Her udvider han den geometriske brownske bevægelse, der drives af en Wiener-proces, med en springkomponent, der drives af en Poisson-proces. Intuitionen er, at kursen normalt følger en kontinuert proces, men at der så med en vis frekvens vil ankomme ny væsentlig information til markedet, der resulterer i et spring i kursen, der gør processen diskontinuert. Som i en normal Poisson-proces vil frekvensen af spring være angivet med en intensitet λ. En intensitet λ på f.eks. 4 vil betyde, at der i gennemsnit er et spring for hver 0.25 tidsenhed, altså hver 3. måned, hvis tidsenheden er et år.

Den geometriske brownske bevægelse fra (2.2) udvides således på følgende

måde[2]:

$$dS_t/S_t = (\mu - \lambda\overline{k})dt + \sigma dW_t + kdq$$

Her er q Poisson-processen, og således vil $dq = 1$, når Poisson-hændelsen (springet) indtræffer, og $dq = 0$, når der ikke er noget spring. k er størrelsen på springet, givet at der indtræffer et spring. Denne tillades at være stokastisk. $\overline{k}$ er middelværdien for k, og leddet $-\lambda\overline{k}$ sørger for, at den gennemsnitlige drift i processen stadig vil være lig μ på trods af springleddet kdq.

Merton (p. 135) giver et eksempel på en simpel springproces (kaldet *jump-to-ruin*), hvor kursen følger en geometrisk brownsk bevægelse, men hvor der er en positiv sandsynlighed for, at kursen pludselig går i nul (firmaet går fallit). Her er k konstant $= -1$, og således er $\overline{k}$ også lig -1. Denne model fører frem til en variant af Black-Scholes-formlen, hvor renten blot sættes til $r' = r+\lambda$. Denne model er interessant i forhold til prisfastsættelse af warrants og aktier på enkeltfirmaer, men er mindre relevant i forhold til aktieindeks og valutakurser.

En mere realistisk model er den, hvor den faktor $1 + k$, der ganges på S_t, når et spring indtræffer, er log-normal-fordelt. Merton studerer også denne såkaldte *jump diffusion model*, hvor

$$\ln(1 + k) \sim N(\ln(1 + \overline{k}) - \frac{1}{2}\delta^2, \delta^2)$$

og således $\mathbb{E}(1 + k) = 1 + \overline{k}$. Han når herved frem til, at optionsprisen kan skrives som en uendelig sum, hvis værdi kan beregnes med numeriske metoder.

3.5 Stokastisk volatilitet og spring (SVJD)

I Bates (1996) kombineres Hestons stokastiske volatilitetsmodel med Mertons springmodel. Denne kombinerede model bliver kaldt SVJD-modellen (*stochastic volatility and jump diffusion*) og kursen følger følgende proces

$$dS_t/S_t = (\mu - \lambda\bar{k})dt + \sqrt{V_t}dW + kdq \tag{3.1}$$

(Hele modellen opsummeres i afsnit 4.1).

Som nævnt i foregående afsnit er der gode indikationer på, at der udover stokastisk volatilitet med fordel kan inkorporeres spring i modellerne. Andersen et al. (2002) finder i deres analyse af en række forskellige modellers

[2]Notationen adskiller sig en anelse fra Mertons fremstilling.

evne til at beskrive S&P500-aktieindekset, at medtagelsen af spring er nødvendig for en tilstrækkelig god beskrivelse af prisprocessen. Dog finder de, at log-SVJD-modellen, hvor volatiliteten følger en log-varians proces, fitter marginalt bedre end kvadratrods-SVJD-modellen (Bates' version), hvor volatiliteten følger en kvadratrods proces. De konkluderer dog (p. 1263), at de ikke realistisk kan differentiere mellem de to modeller, og vælger herefter at bruge kvadratrods-SVJD til den videre analyse, da eksistensen af en lukket formel gør arbejdet meget lettere.

En berettiget kritik af Bates' model er, at det er klart, at den fitter bedst, da det også er den model, der har flest parametre at fitte med. Det er fristende at indføre mere og mere komplicerede modeller med mere eller mindre plausible udvidelser, der så kan beskrive de underliggende tidsrækker eller optionspriserne mere nøjagtig. Problemet er imidlertid, at disse fitninger på historiske data ikke nødvendigvis beskriver de fundamentale faktorer i markedsdynamikken særligt godt og derfor ikke giver et præcist billede af fremtiden.

Der er imidlertid gjort forsøg på at udvide SVJD-modellen med flere parametre. Andersen et al. (2002) lader springintensiteten være en funktion af volatiliteten: $\lambda(t) = \lambda_0 + \lambda_1 V_t$. De finder imidlertid (p. 1263), at værdierne for λ_1 er upræcise og insignifikante. De afprøver også udvidelse af modellen med et *volatility-in-mean* led, der tillader, at afkastet (normalt μ) afhænger af volatiliteten, så (3.1) får følgende form: $dS_t/S_t = (\mu + cV_t - \lambda\overline{k})dt + \sigma dW_t + kdq$. De finder ligeledes, at det er insignifikant, at dette led skal indgå.

Bakshi et al. (1997) udvider modellerne BS, SV og SVJD ved at gøre renten stokastisk efter en CIR-model. De finder, at deres resulterende SVSI-J-model (*stochastic volatility, stochastic interest rate and jumps*) ikke forbedrer performance væsentligt og udelader den som konsekvens fra præsentationen i artiklen (p. 2006). Ligeledes konkluderer de, at deres SVSI-model på trods af de tre ekstra parametre, der bruges til at beskrive dynamikken i rentens udvikling, ikke giver et bedre fit end SV-modellen og desuden giver uplausible værdier for volatilitetsparametren (p. 2019).

Der er således gode tegn på, at udvidelsen med spring er mere nyttig end mange andre tænkelige udvidelser.

3.6 Praktiske overvejelser

SVJD modellen har tilsyneladende endnu ikke fundet stor udbredelse blandt de finansielle institutioner, og det er interessant at se nærmere på, hvorfor dette ikke sket.

Selvom den virker som et godt bud på en model til at prisfastsætte op-

tioner, er der mange konkurrerende modeller, og da der ikke er én model, der er fuldstændig overbevisende, har praktikerne svært ved at gennemskue, hvilken model de skal basere deres beregninger på.

Samtidigt vil vanskelighederne ved estimation af parametrene altid begrænse den praktiske anvendelighed af en prisfastsættelsesmodel. En stor fordel ved Black-Scholes-modellen er, at der netop er én parameter, nemlig volatiliten, som man skal estimere og have en holdning til (Wilmott 1998, p. 333). Dette antal er helt perfekt. Hvis der var flere parametre, ville der være for meget arbejde ved det (og måske ville det være uoverskueligt for *traderen*). Hvis der ikke var nogle parametre, der skulle estimeres, kunne handlerne afvikles af en computer. Metoderne til estimation af parametrene i SVJD-modellen vil blive behandlet i afsnit 4.5, efter at parametrene for SVJD-modellen under det risikoneutrale mål er introduceret.

Indførelsen af spring betyder også, at det ikke længere er muligt at lave en (teoretisk) perfekt afdækning (*hedging*). Såfremt der reelt er spring i det underliggende aktivs kurs, vil en *delta-hedge* naturligvis ikke være en perfekt afdækning, men ved brug af Black-Scholes modellen kan man bevare illusionen om, at den er det. Bruger man derimod en model med spring, bliver det direkte synligt, at en perfekt hedge ikke er mulig, hvilket kan være en ubehagelig indrømmelse (Wilmott 1998, p. 334). Som det bemærkes af Andreasen (2003, p. 14) vil det imidlertid være naivt at bruge en model uden springrisiko til prisfastsættelse af optioner på aktier. Han sammenligner det med at løbe foran et damplokomotiv og samle mønter op fra skinnerne[3]. I lang tid kan det se ud som om du klarer dig rigtigt godt, men damplokomotivet rammer dig, før du ved det!

[3] "picking up pennies in front of a steam engine"

4 SVJD-modellen

I dette kapitel opsummeres SVJD-modellen, der er foreslået af Bates (1996). Derefter vises de ændringer, der skal til for at omskrive modellen til en risikoneutral verden, og den lukkede formel for prisen på en europæisk option præsenteres. Til brug for at kunne udregne integralet, der indgår i formlen, introduceres Gauss-Kronrod integration. Herefter vil metoder til estimation af de nødvendige modelparametre kort blive omtalt. Endelig vil modelparametrenes indvirkning på sandsynlighedsfordelingen for udløbskursen og prisen blive behandlet. Disse effekter illustreres grafisk, og tal for skævhed og kurtosis præsenteres.

4.1 SVJD-modellen og dens parametre

Som beskrevet i afsnit 3.5 kombinerer Bates Hestons kvadratrodsmodel med Mertons springmodel, hvilket resulterer i en model, der udvider Black-Scholes med både et stokastisk volatilitetsled og en springkomponent:

Definition 14 ***SVJD-modellen*** (stochastic volatility and jump diffusion) *defineres ved følgende ligninger*

$$\begin{aligned} dS/S &= (\mu - \lambda\bar{k})dt + \sqrt{V}dW + kdq \\ dV &= (\alpha - \beta V)dt + \sigma_v\sqrt{V}dW_v \\ \text{cov}(dW, dW_v) &= \rho dt \\ \text{prob}(dq = 1) &= \lambda dt \\ \ln(1+k) &\sim N(\ln(1+\bar{k}) - \frac{1}{2}\delta^2, \delta^2) \end{aligned}$$

hvor de enkelte symboler (processer og parametre) tolkes som vist i tabel 4.1.

For at illustrere SVJD-modellens store fleksibilitet, er der i tabel 4.2 en oversigt over andre modeller, der er specialtilfælde af SVJD-modellen. Disse er alle beskrevet i kapitel 3 og både de teoretiske og empiriske aspekter af dem er behandlet grundigt i den finansielle litteratur.

S	kurs på det underliggende aktiv (f.eks. valutakurs)	$S > 0$
μ	driften i kursen	
$\sqrt{V}$	kursens volatilitet	$\sqrt{V} > 0$
σ_v	volatilitetens volatilitet	$\sigma_v \geq 0$
$\sqrt{\frac{\alpha}{\beta}}$	ligevægtsniveau for volatiliteten (*steady state volatility*)	$\sqrt{\frac{\alpha}{\beta}} > 0$
β	"trækkraften" mod ligevægten (*mean-reversion rate*)	$0 < \beta \leq 1$
ρ	korrelationskoefficient mellem de to Wiener-processer for kursen W og volatiliteten W_v	$0 \leq \rho \leq 1$
λ	springintensiteten for Poisson-processen	
k	størrelsen af et spring, når springet indtræffer	
$\overline{k}$	middelværdi for springenes størrelse	
δ	standardafvigelsen for springenes størrelse	$\delta > 0$

Tabel 4.1: De enkelte parametre i SVJD-modellen. For at modellen giver mening skal værdierne holde sig indenfor de angivne intervaller.

Model	**Artikel**	$\boldsymbol{\mu}$	$\mathbf{V}_0$	$\boldsymbol{\sigma}_v$	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\rho}$	$\boldsymbol{\lambda}$	$\overline{\boldsymbol{k}}$	$\boldsymbol{\delta}$
Black-Scholes	Black & Scholes (1973)	μ	σ	0	0	0	0	0	0	0
SV kvadratrodsmodel	Heston (1993)	μ	V_0	σ_v	α	β	ρ	0	0	0
jump-to-ruin	Merton (1976)	μ	σ	0	0	0	0	λ	-1	0
JD *jump diffusion*	Merton (1976)	μ	σ	0	0	0	0	λ	$\overline{k}$	δ
SVJD	Bates (1996)	μ	V_0	σ_v	α	β	ρ	λ	$\overline{k}$	δ

Tabel 4.2: Specialtilfælde af SVJD-modellen.

Herudover kan SV og SVJD-modellerne varieres ved at lade $\rho = 0$ eller tillade $\rho \neq 0$. Faktisk kunne man for at yde fuld retfærdighed til SV- og SVJD-modellerne tilføje *and price-volatility correlation* til navnene, da korrelationen er en vigtig modelegenskab, der bl.a. kan forklare skævhed i sandsynlighedsfordelingerne for S_T.

Nogle af de øvrige modeller fra kapitel 3 kan fås med små modifikationer af SVJD-modellen, men er ikke specialtilfælde.

4.2 SVJD-modellen under det risikoneutrale mål

Som for Black-Scholes modellen præsenteret i kapitel 2 er det nemmere at prisfastsætte derivater, hvis man omskriver SVJD-modellen til det risikoneutrale mål. I tilfælde af SVJD-modellen kan man imidlertid ikke nøjes med et simpelt arbitrageargument, da der ikke findes instrumenter (og da slet ikke risikofrie), der repræsenterer den stokastiske volatilitet og spring-delen i processen. Det er derfor nødvendigt med antagelser omkring investorernes nyttefunktioner.

I det følgende argumenteres der for, at SVJD-modellen med visse antagelser kan omskrives til en model på samme form som "objektive" model[1]. Dog udskiftes μ med $r^d - r^f$ på samme måde som i Black-Scholes tilfældet. Parametrene skal imidlertid risikojusteres og får nye værdier, og de nye parametre mærkes med toptegn *.

Bates omskriver modellen under antagelse af, at alle investorer i økonomien opfører sig som en repræsentativ agent, der har marginalnytte J_w af penge[2]. Springparametrene skal under disse antagelser justeres på følgende måde:

$$\begin{aligned} \lambda^* &= \lambda \mathbb{E}\left(1 + \frac{\Delta J_w}{J_w}\right) \\ \bar{k}^* &= \bar{k} + \frac{\mathrm{cov}(k, \Delta J_w / J_w)}{\mathbb{E}(1 + \Delta J_w / J_w)} \end{aligned}$$

Her angiver $\Delta J_w / J_w$ den procentsats, hvormed kursen vil springe, når et spring indtræffer. Hvis man yderligere antager, at nyttefunktionen er tidsseperabel og isoelastisk, får man, at $1 + k^*$ er log-normalt fordelt med middelværdi $\ln(1 + \bar{k}^*) - \frac{1}{2}\delta^2$ og samme varians δ^2 som under den objektive model.

Den repræsentative investor vil også forlange en præmie for usikkerheden på volatiliteten ($\frac{dJ_w}{J_w}$ betegner det percentuelle chok ved fravær af spring.):

$$\Phi_v = \mathrm{cov}\left(dV, \frac{dJ_w}{J_w}\right)$$

Under den samme antagelse om tidsseperabel, isoelastisk nytte vil den volatilitetsrisikopræmien kun afhænge af V, $\Phi_v = f(V)$, og kan med rimelighed approksimeres som lineær, $\Phi_v(V) = \xi V$. Parameteren β vil således ændres til $\beta^* = \beta - \xi$.

Endelig skal driften μ, som i Black-Scholes-tilfældet, justeres, så den afspejler rentespændet mellem de to valutaer $r - r_f$, og de to brownske bevægelser samt Poisson-processen skifter mål og får således en tilde ($\widetilde{\ }$) analogt til parametrene, der fik en stjerne (*).

Den samlede model i en risikoneutral verden får følgende udseende:

$$\begin{aligned} dS/S &= (r - r_f - \lambda^* \bar{k}^*)dt + \sqrt{V} d\widetilde{W} + k^* d\widetilde{q} \\ dV &= (\alpha - \beta^* V)dt + \sigma_v \sqrt{V} d\widetilde{W}_v \end{aligned} \tag{4.1}$$

[1] Litteraturen er relativ enig om begrebet den "risikoneutrale model", men for modellen for de markedsobserverede kurser bliver der brugt flere udtryk som f.eks. "fysiske", "sande", "objektive", "virkelige" osv.

[2] Flere detaljer om økonomien med en repræsentativ agent og om antagelserne om nyttefunktioner kan findes i Bates (1988).

$$\begin{aligned}
\operatorname{cov}(d\widetilde{W}, d\widetilde{W}_v) &= \rho dt \\
\operatorname{prob}(d\widetilde{q} = 1) &= \lambda^* dt \\
\ln(1+k^*) &\sim N(\ln(1+\bar{k}^*) - \frac{1}{2}\delta^2, \delta^2)
\end{aligned}$$

4.3 Formel for optionspriser i SVJD-modellen

Efter at have opstillet modellen under det risikoneutrale mål, kan der nu prisfastsættes optioner under modellen. Som vist i afsnit 2.7 kan prisen på en købsoption skrives som $c = e^{-r^d T}(FP_1 - XP_2)$, hvor P_1 og P_2 fortolkes som sandsynligheder, og F er *forward*-prisen på valuta. Under Black-Scholes log-normale antagelse, kunne vi relativt let finde de to sandsynligheder, men i Bates-modellen er fordelingen af S_T ikke kendt. Derfor er det nødvendigt med en anden metode til udregning af de to sandsynligheder. Heston (1993) har udviklet en metode, der ved hjælp af de momentgenererende funktioner hørende til P_1 og P_2 gør det muligt at finde optionsprisen (næsten) uden brug af numeriske metoder.

De momentgenererende funktioner af $\ln(S_T/S_0)$ for de to sandsynligheder P_j $(j = 1, 2)$ bliver (Bates 1996, p. 76):

$$\begin{aligned}
F_j(u; V, T-t) &\equiv \mathbb{E}(e^{u \ln(S_T/S_t)} \mid P_j) \qquad (4.2) \\
&= \exp(C_j(u; T-t) + D_j(u; T-t)V + E_j(u; T-t)
\end{aligned}$$

hvor

$$\begin{aligned}
C_j(u; T-t) &= (r - r_f - \lambda^*\bar{k}^*)u(T-t) - \frac{\alpha(T-t)}{\sigma_v^2}(\rho\sigma_v u - \beta_j - \gamma_j) \\
&\quad -\frac{2\alpha}{\sigma_v^2}\ln\left(1 + \frac{1}{2}(\rho\sigma_v u - \beta_j - \gamma_j)\frac{1 - e^{\gamma_j(T-t)}}{\gamma_j}\right)
\end{aligned}$$

$$D_j(u; T-t) = -2\frac{\mu_j u + \frac{1}{2}u^2}{\rho\sigma_v u - \beta_j + \gamma_j \frac{1+e^{\gamma_j(T-t)}}{1-e^{\gamma_j(T-t)}}}$$

$$E_j(u; T-t) = \lambda^*(T-t)(1+\bar{k}^*)^{\mu_j + 1/2}((1+\bar{k}^*)^u e^{\delta^2(\mu_j u + u^2/2)} - 1)$$

$$\gamma_j(u) = \sqrt{(\rho\sigma_v u - \beta_j)^2 - 2\sigma_v^2(\mu_j u + \frac{1}{2}u^2)}$$

$$\mu_1 = +\tfrac{1}{2},\ \mu_2 = -\tfrac{1}{2},\ \beta_1 = \beta^* - \rho\sigma_v \text{ og } \beta_2 = \beta^*.$$

P_1 og P_2 bestemmes herefter som

$$P_j(u; S_t, T-t) = \mathbb{Q}(S_T > K \mid F_j) = \frac{1}{2} + \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{F_j(iu)e^{-iu\ln(K/S_t)}}{iu}du$$

hvor der er benyttet en invers Fourier transformation af den karakteriske funktion med komplekse værdier $F_j(iu; S_t, T-t)$. (i er det komplekse tal $\sqrt{-1}$.) Ved at benytte egenskaberne for karakteriske funktioner kan funktionen omskrives til

$$P_j(u; S_t, T-t) = \frac{1}{2} + \frac{1}{\pi}\int_{0}^{\infty} \frac{\text{imag}(F_j(iu)e^{-iu\ln(K/S_t)})}{u}du \tag{4.3}$$

Herefter kan prisen på en europæisk købsoption findes som

$$c(S, V, T-t; K, \theta) = e^{-r^d(T-t)}(FP_1 - KP_2) \tag{4.4}$$

hvor $\theta = \langle \lambda^*, \bar{k}^*, \delta, \alpha, \beta^*, \sigma_v, \rho \rangle$ og $F = S_t e^{(r^d - r_f)(T-t)}$.

Prisen på en salgsoption følger af *put-call*-pariteten som vist i afsnit 2.7:

$$p(S, V, T-t; K, \theta) = e^{-r^d(T-t)}(F(1-P_1) - K(1-P_2)) \tag{4.5}$$

Implementationen af denne formel er vist i afsnit D.2.2 og demonstreret i afsnit C.3.

Der findes en række alternative formler til den netop præsenterede på trods af, at de alle prisfastsætter den samme option i den samme model. Dette skyldes, at der i udledningen kan benyttes forskellige versioner af Fourier transformationen. Desuden kan der vælges én af to fremgangsmåder. Den ene består i at bruge den karakteristiske funktion til at udlede de to sandsynligheder P_1 og P_2, der derefter kan bruges i den Black-Scholes lignende formel (4.4). Den anden udleder formlen ved først at prisfastsætte en option med afkastet $\mathcal{P}(S_T) = \min(S_T, K)$ og derved nå frem til en formel med et enkelt integrale, der skal integreres for at finde den endelige pris. Denne benævnes den karakteristiske formel. En grundig samlet gennemgang af disse to fremgangsmåder er præsenteret i Sepp (2003).

Andre versioner af den Black-Scholes-lignende formel i SV-tilfældet kan findes i Heston (1993, p. 331) og Duffie (2001, p. 178 ff.). Nielsen (1999) udleder en formel for SVJD-modellen, der er baseret på samme Fourier transformation som i Hestons artikel.

Lipton (2002) kombinerer lokale volatilitetsmodeller (Dupire 1994) med SVJD-modellen og specialtilfælde heraf. Han præsenterer samtidigt en integrationsformel til prisfastsættelse, hvor u ikke på samme måde som i (4.3) optræder under brøkstregen. Dette betyder, at integranden ikke går mod

uendelig, når u går mod 0. Af denne grund er integralet mere velegnet til numerisk integration end integralet i de Black-Scholes-lignende formler. Omvendt fremhæver Sepp (2003, p. 13), at den Black-Scholes-lignende formel kan udregnes 3 gange så hurtigt som den karakteristiske formel.

Formlen (4.2) præsenteret ovenfor har det problem, at der opstår en division med nul, når $\sigma_v = 0$. Dette problem kan kun løses ved at bruge L'Hospitals regel til at finde grænseværdien, når $\sigma_v \to 0$. Dette problem lider formlen i Lipton (2002, p. 64) ikke af, idet leddene i integralet, der kan henføres til SV-komponenten og JD-komponenten, optræder separat[3]. De kan således simpelthen udelades, når henholdsvis σ_v eller λ er lig 0, og derved både eliminere problemet og reducere beregningstiden.

4.4 Gauss-Kronrod integration

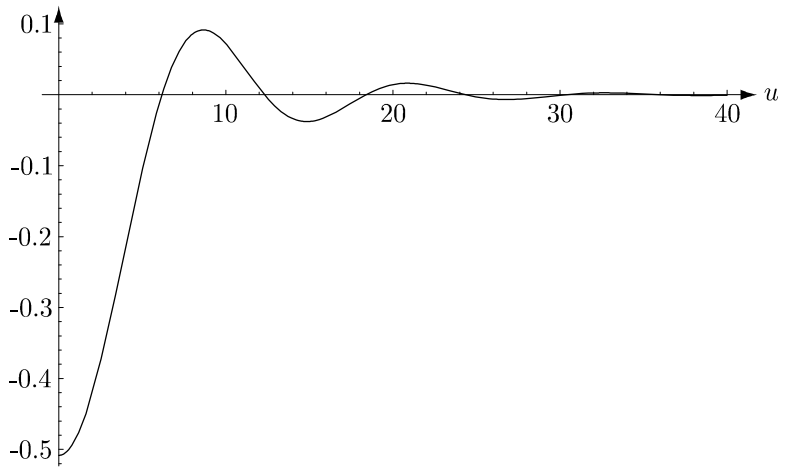
I den lukkede formel for optionsprisen indgår et integrale, der med den nuværende viden kun kan udregnes numerisk. Formlen kan derfor kun kaldes semi-lukket.

Selvom der indgår komplekse tal i formlen, er integranden reel, da den imaginære del udtrækkes med imag. Ydermere konvergerer integranden hurtigt til 0, som det ses i figur 4.1. Det er derfor i praksis ikke nødvendigt at integrere helt til uendeligt, men tilstrækkeligt at tage integralet ud til f.eks. 300. (Den nøjagtige optimale afskæring afhænger dog af integrandens form og således af parametrene.) Problemet er derfor et sædvanligt reeltals integrations-problem i én dimension på et endeligt interval. Til dette findes der en lang række metoder (f.eks. trapez-formlen og Simpsons formel). En af de mest effektive metoder er imidlertid Gauss-Kronrod, der giver en god præcision i forhold til beregningstid. Det er denne metode, der anbefales i Bates (1996) og er standardmetoden i eksempelvis programpakken Mathematica.

[3]Formel (7) fra Lipton (2002) kan udvides med springkomponenten på følgende måde:

$$\begin{aligned} & C^{(SVJD)}(0,S,v,T,K) \\ = \; & e^{-r^f T}S - \frac{e^{-r^d T}K}{2\pi} \\ & \int_{-\infty}^{\infty} \frac{e^{(-iu+1/2)\ln(\frac{S}{K}+(r^d-r^f)T)+\alpha^{(SV)}(T,u)+(u^2+1/4)\beta^{(SV)}(T,u)v+\alpha^{(JD)}(T,u)}}{u^2+1/4}du \end{aligned}$$

Desuden kan der gøres opmærksom på, at fortegnet foran $\beta^{(SV)}$-funktionen er forkert i Liptons formel. Det skal være et plus som vist i formlen ovenfor.



Figur 4.1: Integranden brugt i prisfastsættelsesformlen til SVJD-modellen. y-aksen viser integranden fra (4.3). $K = 100$, $S_t = 60$, $\rho = 0.3$. Øvrige parametre som i tabel 4.3, p. 52.

4.4.1 Integration med gaussisk kvadratur

Idéen med en effektiv metode til numerisk integration er at få så god en tilnærmelse til det rigtige integrale som muligt med udregning af så få funktionsværdier som muligt. Den berømte matematiker Gauss indså, at man ved hjælp af kun tre værdier af funktionen kunne få en præcision af integralet svarende til en tilnærmelse med et femtegradspolynomium. Metoder, der tager udgangspunkt i hans princip kaldes gaussisk kvadratur-formler (*Gaussian quadrature formulas*) (O'Neil 2002).

Opgaven lyder på at finde integralet $I = \int_a^b f(x)dx$, men for at gøre udledningen mere håndterlig omskrives dette til $I = \int_{-1}^{1} g\,(t)\,dt$, hvor $g(t) = \frac{b-a}{2} f\left(\frac{(b-a)}{2}t + \frac{a+b}{2}\right)$.

Vi starter simpelt med Gauss' tilnærmelse med 2 punkter:

$$\int_{-1}^{1} g(t) \approx w_1 g(t_1) + w_2 g(t_2) \tag{4.6}$$

Denne formel skal gælde med lighedstegn for polynomier op til grad 3, så den må specielt gælde, når $g(t)$ er et af mononomierne 1, t, t^2og t^3.

$$\int_{-1}^{1} t^3 dt = 0 \quad = \quad w_1 t_1^3 + w_2 t_2^3 \tag{4.7}$$

$$\begin{aligned}\int_{-1}^{1} t^2 dt = \frac{2}{3} &= w_1 t_1^2 + w_2 t_2^2 \\ \int_{-1}^{1} t dt = 0 &= w_1 t_1 + w_2 t_2 \\ \int_{-1}^{1} dt = 2 &= w_1 + w_2\end{aligned}$$

Ved at multiplicere 3. ligning med t_1^2 og derefter subtrahere den første fås:

$$0 = w_2 \left(t_2^3 - t_2 t_1^2\right) = -w_2 t_2 (t_1 + t_2)(t_1 - t_2)$$

Denne ligning er opfyldt, hvis $w_2 = 0$, $t_2 = 0$, $t_1 = -t_2$ eller $t_1 = t_2$. Den eneste af disse, der giver fornuft, er $t_1 = -t_2$, da løsningen ellers kun indeholder et enkelt punkt. Med denne løsning giver 3. og 4. ligning, at $w_1 = w_2 = 1$, og herefter 2. og 3. ligning $t_2 = -t_1 = \sqrt{\frac{1}{3}} \approx 0.5773$. Ved at indsætte disse værdier i formlen (4.6) fås tilnærmelsen, der altså vil være præcis op til tredjegradspolynomier.

Den generelle formel svarende til (4.7) for estimation ved brug af mange punkter bliver

$$w_1 t_1^k + \cdots + w_n t_n^k = \begin{cases} 0, & \text{for } k = 1, 3, 5, \ldots, 2n-1 \\ \frac{2}{k+1}, & \text{for } k = 0, 2, 4, \ldots, 2n-2 \end{cases} . \tag{4.8}$$

Denne formel er på ingen måde let at løse. Det er imidlertid så heldigt, at t'erne er rødder i det n'te grads Legendre-polynomium, der er rekursivt defineret som følger (Press, Teukolsky, Vetterling & Flannery 1992, p. 253):

$$\begin{aligned}L_0(x) &= 1 \\ L_1(x) &= x \\ L_n(x) &= \frac{x(2n-1)L_{n-1} - (n-1)L_{n-2}}{n}\end{aligned}$$

Legendre-polynomierne af 5. og 6. grad bliver således f.eks.

$$\begin{aligned}L_5(x) &= \frac{15\,x}{8} - \frac{35\,x^3}{4} + \frac{63\,x^5}{8} \\ L_6(x) &= -\frac{5}{16} + \frac{105\,x^2}{16} - \frac{315\,x^4}{16} + \frac{231\,x^6}{16}\end{aligned}$$

Når rødderne i et af disse polynomier er fundet, kan løsningerne indsættes i (4.8), og vægtene w_i kan findes. For $n \geq 6$ er problemet imidlertid ikke så ligetil, da rødderne bliver komplekse (dog med forsvindende lille imaginær

del) og (4.8) derfor besværlig at løse. Derfor er der i implementationen brugt de tabellerede værdier fra O'Neil (2002).

Den endelige formel bliver

$$\begin{aligned} I &= \int_a^b f(x)dx \\ &= \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(b-a)}{2}t + \frac{a+b}{2}\right) \\ &= \int_{-1}^{1} g(t) \approx G_n = \sum_{i=1}^{n} w_i g(t_i) \end{aligned}$$

Her betegner G_n det tilnærmede integrale ved brug af n punkter og vil være præcist, hvis f er et polynomium op til grad $2n-1$. $I \approx G_7$ gælder altså f.eks. med lighedstegn, hvis f er et 13.-gradspolynomium.

For at kunne opnå en ønsket præcision, når der om lidt bruges tilpassende integration, er der brug for et mål for den maksimale afvigelse fra den virkelige værdi. Et konservativt bud på fejlen, der begås, kan udregnes ved $\epsilon \leq |G_n - G_{n+1}|$. Herved er der imidlertid brugt $2n+1$ funktionsværdiudregninger, og præcisionen af $I \approx G_{n+1}$ er kun op til grad $2(n+1)-1 = 2n+1$. Det må kunne gøres bedre.

4.4.2 Kronrods udvidelse

Den russiske datalog Kronrod fandt ud af, at man kan udvide Gauss-metoden ved at vælge $n+1$ flere punkter, så estimationen bliver[4]

$$K_{2n+1} = \sum_{i=1}^{n} u_i g(t_i) + \sum_{j=1}^{n+1} v_j g(s_j)$$

Her er t_i de eksisterende punkter fra G_n med tilhørende nye vægte u_i, og s_j er de $n+1$ nye punkter[5] med tilhørende vægte v_j. K_{2n+1} og G_n deler således n punkter, og dette giver en fordel, når den maximale fejl skal udregnes. Denne kan nemlig findes med $\epsilon \leq |K_{2n+1} - G_n|$. Således har vi med $2n+1$ funktionsudregninger tilnærmet værdien af I med præcision op til grad $2(2n+1)-1 = 4n+1$ og samtidigt udregnet et øvre loft for fejlen!

Kildekoden kan ses i afsnit D.2.3.

[4] O'Neil (2002)

[5] De $n+1$ nye punkter er imidlertid ikke rødder i Legendre-polynomierne, men må findes med en anden metode.

4.4.3 Tilpassende integration

En mulighed for at få tilstrækkelig præcision er at vælge et n, der er tilstrækkeligt stort. Det ville imidlertid kræve at punkterne til K_{2n+1} var tabellagt i forvejen. Desuden ville problemet med at vælge n så lille som muligt samtidigt med at en ønsket præcision opnås stadig være tilbage. Det er derfor mere effektivt at opdele intervallet a til b i en række mindre intervaller og summe integralerne af disse. Men ofte er visse områder af funktionen mere ujævn end andre, og derfor er der brug for forskellig tæthed af punkterne i forskellige områder. En god strategi til at tage højde for dette er at udregne det ønskede integrale, og hvis præcisionen ikke er god at opdele intervallet i to. Denne metode kaldes tilpassende integration (*adaptive integration*) (Gerald & Wheatley 1999, p. 394).

Algoritmen kan formuleres rekursivt på følgende måde (ε er tolerancen for præcision):

$I([a;b],\varepsilon)$: Udregn K_{15} og G_7. Hvis $|K_{15} - G_7| < \varepsilon$, sættes $I = K_{15}$, ellers sættes $I = I\left(\left[a;\frac{(a+b)}{2}\right],\frac{\varepsilon}{2}\right) + I\left(\left[\frac{(a+b)}{2};b\right],\frac{\varepsilon}{2}\right)$.

Som det ses er der ikke tale om atomfysik, og der findes da også mere effektive metoder til at udvide Gauss-Kronrod metoden. Til vores formål er denne rutine imidlertid tilstrækkelig, og det er denne, der bruges i det medfølgende program.

4.5 Estimation af parametre

Som nævnt i afsnit 3.6 er det en nødvendig forudsætning for brug af SVJD-modellen, at man kan estimere parametrene. Der er overordnet to metoder til at bestemme parameterestimaterne. Man kan finde de parametre, der er implicitte i optionspriserne, eller man kan finde parametrene ud fra tidsrækken for kursen på det underliggende aktiv.

Metoden til implicit parameterestimation tager udgangspunkt i observerede optionspriser. Typisk bruges en variant af en mindste kvadraters metode, der finder de parametre, der minimerer den kvadrerede forskel mellem den observerede optionspris og modellens optionspris. Dette gøres så for samme type option for forskellige værdier af t, og de gennemsnitlige værdier for parametrene kan så bruges som modelparametre. Beskrivelser af konkrete metoder kan findes i Bakshi et al. (1997, p. 2016) og Bates (1996, p. 83).

Ved at tage udgangspunkt i optionspriserne vil parametrene afspejle markedets forventninger til den fremtidige kursudvikling modsat tidsrækkeanalyse, der vil resultere i parametre for den historisk realiserede kursudvikling. Dette har vist sig at forbedre evnen til at forudsige optionspriser. Samtidig

har estimation med udgangspunkt i optionspriser den fordel, at de estimerede parametre er under det risikoneutrale sandsynlighedsmål og således kan bruges direkte i prisfastsættelsesformlen, uden at det er nødvendigt med justeringer for risikopræmier.

Ved at tage udgangspunkt i tidsrækken kan parametrene findes ved hjælp af økonometriske teknikker som f.eks. *maximum likelihood* eller *generalized method of moments* (GMM). En videreudvikling af disse teknikker, der baserer sig på *simulated method of moments* (SMM), er *efficient method of moments* (EMM), der bruges af bl.a. Andersen et al. (2002) . Denne teknik er simulationsbaseret og beslægtet med Monte Carlo metoden, der gennemgås i kapitel 5. Da denne metode tager udgangspunkt i den observerede tidsrække af kursen vil parametrene afspejle kursudviklingen under det objektive sandsynlighedsmål.

Bates (2003) udvikler en *maximum likelihood* metode til parameterestimation, der baserer sig på en udvidet form for Bayes' regel. Denne gør det muligt rekursivt at opdatere en karakteristisk funktion for den simultane fordeling af latente variable (tilstandsvariable såsom volatiliteteten) og data betinget af historiske data. Herefter kan *likelihood* funktionerne udregnes ved hjælp af invers Fourier transformationsformler. Denne metode bruges så til at estimere parametre for en udvidet SVJD-model, hvor springintensiteten afhænger af volatiliteten. På samme måde som i Andersen et al. (2002) er disse parametre estimeret under det objektive sandsynlighedsmål $\mathbb{P}$. Bates bruger det samme datasæt fra S&P500-indekset som Andersen et al, men kommer frem til væsentligt anderledes parameterestimater. Specielt er der i den udvidede model markant højere springintensitet. Frühwirth-Schnatter & Sögner (2003)undersøger en lignende teknik for Hestons model.

Ved teknikkerne, hvor de estimerede parametre beskriver den "virkelige" prisproces, skal de justeres for risikopræmier før, de kan bruges i en optionsprisfastsættelsesformel. En praktiker, der skal bruge udregnede optionspriser til at finde en fair pris på mere avancerede derivater, der kan beskrives som kombinationer af *vanilla* optioner, vil således typisk være mest interesseret i parametrene under det risikoneutrale mål $\mathbb{Q}$, hvorimod parametrene under det objektive mål $\mathbb{P}$ vil være af mindre betydning. Dette vil gøre, at metoder til implicit parameterestimation vil blive foretrækket.

4.6 Sandsynlighedsfordeling i SVJD-modellen

En af fordelene ved SVJD-modellen er, at den kan give en rigere beskrivelse af fordelingen af afkast end Black-Scholes modellen. I Black-Scholes modellen er fordelingen af afkast log-normal, og man kan kun justere på de to før-

ste momenter (middelværdi og varians). I SVJD-modellen kan man derimod påvirke de højere momenter (skævhed og kurtosis) ved at skrue på modelparametrene. Da sandsynlighedsfordelingen af afkastene fortæller meget om modellens egenskaber og har en direkte konsekvens for prisfastsættelse, vil disse fordelingsegenskaber blive behandlet i dette afsnit.

4.6.1 Momenter

For at kunne beskrive, hvordan parametervalgene i SVJD-modellen påvirker fordelingens form, får vi brug for en række definitioner og resultater fra sandsynlighedsregningen (Weisstein 2003).

Definition 15 ***Det n'te centrale moment*** *for en sandsynlighedsfordeling defineres som det n'te moment taget omkring middelværdien μ.*

$$\mu_n = \int (x-\mu)^n \mathbb{P}(x) dx$$

μ_2 er samtidig lig variansen σ^2.

Definition 16 ***Skævheden*** *er et mål for asymmetri i tæthedsfunktionen og defineres som*

$$\gamma_1 = \frac{\mu_3}{\sqrt{\mu_2}^3}$$

For normalfordelingen er skævheden lig 0. Positiv skævhed indikerer en fordeling, hvor den højre hale er lang (fordelingen er venstreskæv). Negativ skævhed indikerer en fordeling, hvor den venstre hale er lang (fordelingen er højreskæv).

Definition 17 ***Overskydende kurtosis*** (kurtosis excess) *for en fordeling defineres som*

$$\gamma_2 = \frac{\mu_4}{\mu_2^2} - 3$$

Overskydende kurtosis for en normalfordeling er 0*. En fordeling med overskydende kurtosis* > 0 *siges at have overkurtosis eller være topstejl og vil være spidsere og have tykkere haler end en normalfordeling med samme varians. En overskydende kurtosis* < 0 *vil omvendt betyde, at fordelingen er fladere eller har bredere skuldre end normalfordelingen. Selvom kurtosis strengt taget er defineret som ovenstående tal uden at trække* 3 *fra, vil vi i denne afhandling bruge ordet kurtosis for tallet γ_2.*

Hvis vi har en moment-genererende funktion, kan vi bruge følgende resultat til at finde de nødvendige centrale momenter.

Definition 18 *Den **kumulant-genererende funktion** $R(u)$ defineres som logaritmen til den moment-generende funktion $R(u) = \ln M(u)$, og **kumulanterne** κ_1 defineres som dennes afledte:*

$$\begin{aligned}\kappa_1 &= R'(0)\\ \kappa_2 &= R''(0)\\ \kappa_3 &= R'''(0)\\ \kappa_4 &= R''''(0)\end{aligned}$$

Proposition 10 *Middelværdien μ og de første centrale momenter μ_n kan udtrykkes på følgende måde ved hjælp af kumulanterne*

$$\begin{aligned}\mu &= \kappa_1\\ \sigma^2 = \mu_2 &= \kappa_2\\ \mu_3 &= \kappa_3\\ \mu_4 &= \kappa_4 + 3\kappa_2^2\end{aligned}$$

Således når vi frem til, at de fire mål for fordelingens position og form kan beskrives ved hjælp af $R(u)$.

Proposition 11 *Middelværdi, varians, skævhed og kurtosis kan udregnes vha. den kumulantgenererende funktion som*

$$\begin{aligned}\mu &= R'(0)\\ \sigma^2 &= R''(0)\\ \gamma_1 &= \frac{R'''(0)}{\sqrt{R''(0)}^3}\\ \gamma_2 &= \frac{R''''(0) + 3R''(0)^2}{R''(0)^2} - 3 = \frac{R''''(0)}{R''(0)^2}\end{aligned}$$

Da F_2 som defineret i (4.2) netop er den moment-genererende funktion for fordelingen af $\ln(S_T/S_0)$, kan vi ved at definere $R(u) = \ln F_2(u) = C_j(T-t;u) + D_j(T-t;u)V + E_j(T-t;u)$ finde skævhed og kurtosis for afkastet i SVJD-modellen med ovenstående formler.

Det kan af sammenligningsmæssige årsager være nyttigt at kunne sammenligne SVJD-modellens fordeling med en fordeling fra Black-Scholes modellen, der har samme varians. Denne kan findes ud fra variansen $\sigma^2 = R''(0)$. Variansen i Black-Scholes er $\sigma^2(T-t)$ og volatiliteten σ, der skal bruges som parameter i Black-Scholes modellen for at få en sandsynlighedsfordeling, der er sammenlignelig med SVJD-modellens, vil således være givet ved $\sigma = \sqrt{R''(0)/(T-t)}$.

Kildekoden til disse udregninger findes sidst i afsnit E.1.

$T-t$	S_t	r^d	r^f	$\sqrt{V_t}$	σ_v	$\sqrt{\frac{\alpha}{\beta^*}}$	β^*	$\frac{\ln 2}{\beta^*}$	ρ	λ	$\overline{k}^*$	δ
0.5	100	0	0	0.1	0.1	0.1	2	0.347	0	0	0	0

Tabel 4.3: Parameterværdier for SV-model

4.6.2 Tæthedsfunktion

En effektiv metode til at beskrive, hvordan parameterværdierne påvirker fordelingerne rent kvalitativt, er at illustrere fordelingerne grafisk. Dette kan bidrage til en intuitiv forståelse af modellens egenskaber. Vi vil derfor i det følgende se på grafer i lighed med Nielsen (1999, kap. 5), Heston (1993, p. 337 ff.) og Hull & White (1987, p. 293). For at kunne tegne disse grafer har vi brug for et funktionsudtryk for tæthedsfunktionen.

Når S følger en geometrisk brownsk bevægelse som i Black-Scholes modellen, følger det af proposition 1, at det kontinuert tilskrevne afkast $\ln(S_T/S_t)$ er normalfordelt. Tæthedsfunktionen for $z = \ln(S_T/S_t)$ under det risikoneutrale sandsynlighedsmål vil have formen:

$$\mathbb{Q}(z) = \frac{1}{\sqrt{2\pi}\sigma\sqrt{(T-t)}} \exp(-\frac{1}{2}\left((z - (r^f - r^d - \frac{\sigma^2}{2})(T-t))/\sigma\sqrt{(T-t)}\right)^2)$$

For SVJD-modellen fås et noget mere kompliceret udtryk. Her kan sandsynlighedsfordelingen for $z = \ln(S_T/S_0)$ findes med udgangspunkt i den moment-genererende funktion F_2 (Bates 1996, p. 77):

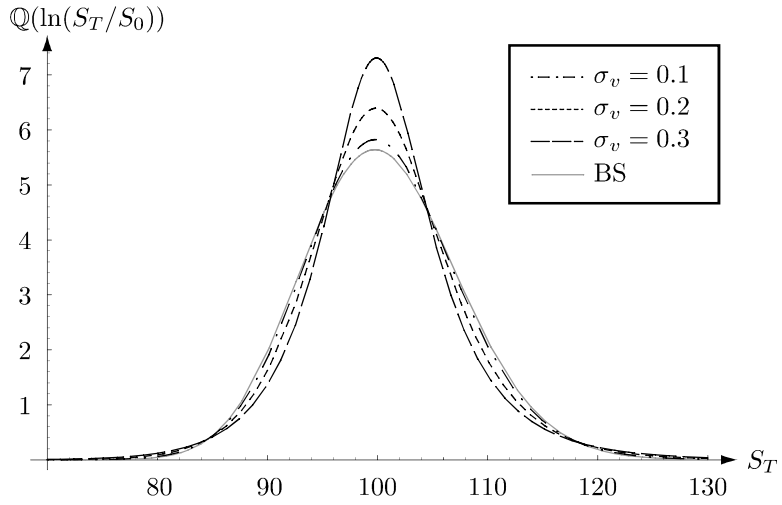
$$\mathbb{Q}(z) = \frac{1}{\pi}\int_0^\infty \text{real}(F_2(iu)e^{-iuz})du \tag{4.9}$$

I afsnit C.3 er det vist, hvordan skævhed og kurtosis kan udregnes og tæthedsfunktionen tegnes vha. det medfølgende program.

4.6.3 Effekt af parametervalg

For at kunne isolere effekten af hver enkelt parameter, vil vi først se på modellen uden spring, altså Hestons SV-model. Til brug for analysen vil vi tage udgangspunkt i parameterværdierne i tabel 4.3, der er de samme som i Nielsen (1999, p. 52) og Heston (1993). Hver enkelt parameter vil så blive varieret undervejs.

For lettere at kunne sammenligne med prisningseffekterne senere, vises S_T fremfor $\ln(S_T/S_0)$ på x-aksen. Dog er punktsandsynlighederne stadig dem, der knytter sig til $\ln(S_T/S_0)$ og en integration over hele x-aksen vil ikke give 1.



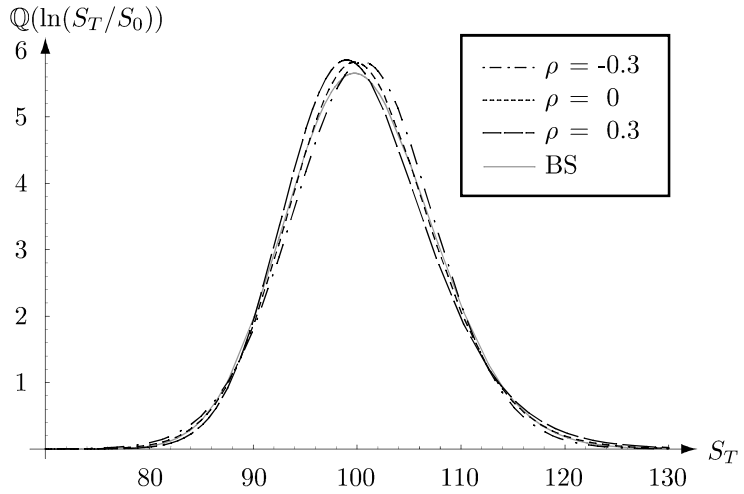
Figur 4.2: Effekten af volatilitetens volatilitet på udløbskursens sandsynlighedsfordeling. Illustreret for $\sigma_v = 0.1, 0.2$ og 0.3, øvrige parametre som i tabel 4.3. BS-volatiliteten ved $\sigma_v = 0.1$ er 0.100005. For $\sigma_v = 0$ vil SV-modellen og BS-modellen være sammenfaldende. Skævhederne er hhv. -0.0089, -0.0356 og -0.0802. Kurtosis er hhv. 0.2522, 1.0096 og 2.2744.

En ændring af $r^d - r^f$ vil blot forårsage en horisontal parallelforskydning af grafen og er derfor ikke så interessant. En ændring af $\sqrt{V_t}$ og $\sqrt{\frac{\alpha}{\beta^*}}$, der for forenklingens skyld er sat lig hinanden i denne analyse, vil give effekter svarende til at ændre σ i Black-Scholes modellen. Spredningen vil altså blive større. Hvis de to parametre ikke er lig hinanden, kan det give kurtosis-effekter, men disse vil dog ikke blive behandlet nærmere her. Ligeledes vil vi ikke se nærmere på effekten af β^*.

Ser vi nærmere på volatilitetens volatilitet σ_v, vil en højere σ_v give en højere kurtosis og en lille effekt på skævheden (Heston 1993, p. 338). Dette er illustreret i figur 4.2. Når $\sigma_v = 0$ er volatiliteten deterministisk, og fordelingen vil svare til fordelingen i Black-Scholes modellen. Når $\sigma_v > 0$ bliver volatiliteten stokastisk og bevæger sig omkring ligevægtsniveauet som følge af stød fra Wiener-processen W_v. Dette giver større sandsynlighed for store udsving i kursen S og dermed tykkere haler.

Hvis $\sigma_v \neq 0$, vil en korrelation $\rho \neq 0$ resultere i, at halen bliver spredt ud, og der vil komme skævhed i fordelingen (Heston 1993, p. 336). Dette er illustreret i figur 4.3. Kurtosis-effekten, der er tydeligst for $\rho = 0$, følger af, at $\sigma_v = 0.1$. Ved $\sigma_v = 0$ vil der jo netop heller ikke forekomme en skævheds-effekt, da W_v og dermed korrelationen med W ikke får nogen betydning.

Det vil nu blive illustreret, hvordan springparametrene indvirker på sand-



Figur 4.3: Effekten af korrelationen mellem volatiliteten og kursen på udløbskursens sandsynlighedsfordeling. Illustreret for $\rho = -0.3$, 0 og 0.3. BS-volatiliteten for $\rho = -0.3$ er 0.0997, men 0.1003 for $\rho = 0.3$. Skævheden er hhv. -0.2420, -0.0089 og 0.2261. Kurtosis er hhv. 0.3114, 0.2522 og 0.3049.

$T-t$	S_t	r^d	r^f	$\sqrt{V_t}$	σ_v	$\sqrt{\frac{\alpha}{\beta^*}}$	β^*	$\frac{\ln 2}{\beta^*}$	ρ	λ	$\overline{k}^*$	δ
0.5	100	0	0	0.1	0	0	0	–	0	0.5	0	0.1

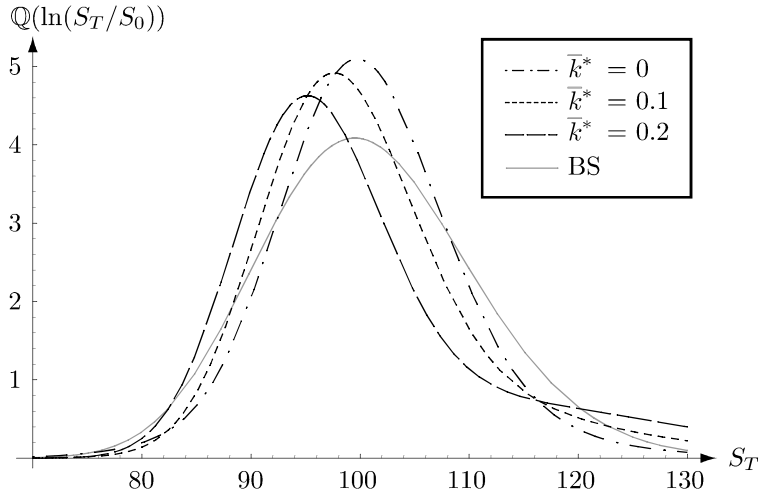
Tabel 4.4: Parameterværdier for JD-model

synlighedsfordelingen. For at kunne studere effekterne uafhængigt, ses der på JD specialtilfældet af SVJD-modellen. SV-parametrene sættes således lig 0. De anvendte parametre er vist i tabel 4.4.

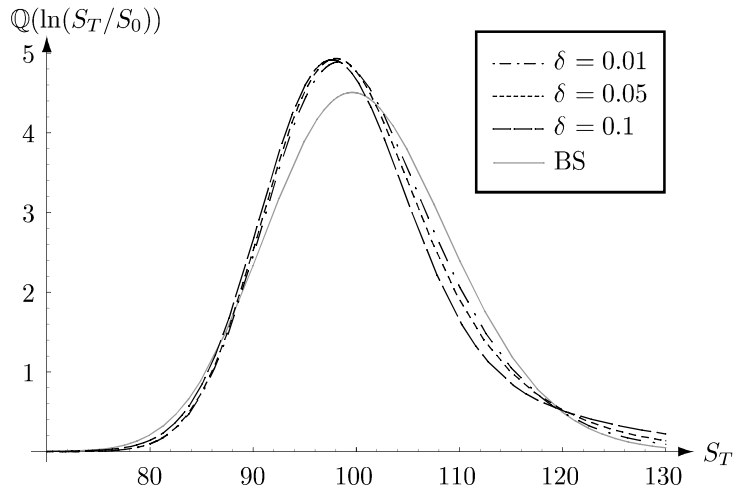
Gennemsnitsstørrelsen på springet $\overline{k}^*$ vil generere skævhed i fordelingen. En positiv springstørrelse $\overline{k}^* > 0$ vil betyde, at springene i gennemsnit er positive, og dette vil betyde en tykkere højre hale. Der vil således være positiv skævhed og fordelingen vil altså være venstreskæv. Samtidigt vil den tykkere hale give overkurtosis. Effekten af negative springstørrelser vil være tilsvarende, blot modsat. En $\overline{k}^* = 0$ vil primært give en kurtosis-effekt og kun lidt skævhed. Dette ses i figur 4.4.

En højere standardafvigelse på springets størrelse δ vil gøre, at springene spredes mere ud, og dette vil give højere kurtosis. Dette er illustreret i figur 4.5.

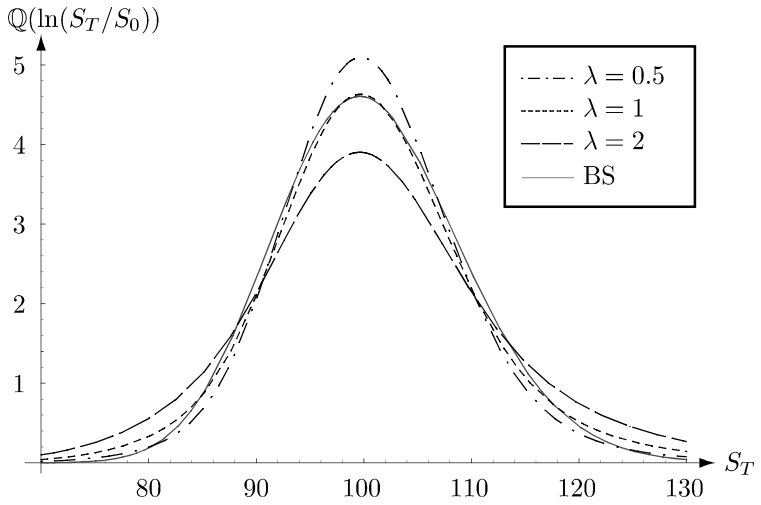
En højere intensitet af springene λ vil forstærke effekten af skævhed, idet springene vil forekomme oftere. Dette er illustreret i figur 4.6. Kurtosis vil stige et vist stykke, men vil derefter falde. Dette er illustreret i figur 4.7. De øvrige grafer af denne type er ikke vist, da de alle viser en monoton



Figur 4.4: Effekten af den gennemsnitlige springstørrelse på udløbskursens sandsynlighedsfordeling. Illustreret for $\overline{k}^* = 0$, 0.1 og 0.2, øvrige parametre som i tabel 4.4. BS-volatiliteten for $\overline{k}^* = 0.1$ er 0.1381. Skævheden er hhv. -0.0577, 0.9247 og 1.4307. Kurtosis er hhv. 1.3378, 2.3515 og 3.3643.



Figur 4.5: Effekten af springenes standardafvigelse på udløbskursens sandsynlighedsfordeling. Illustreret for $\delta = 0.01$, 0.05 og 0.1, $\overline{k}^* = 0.1$, øvrige parametre som i tabel 4.4. BS-volatiliteten for $\delta = 0.01$ er 0.1252. Skævheden er hhv. 0.3584, 0.5541 og 0.9247. Kurtosis er hhv. 0.4127, 0.9352 og 2.3515.



Figur 4.6: Effekten af springintensiteten på udløbskursens sandsynlighedsfordeling. Illustreret for $\lambda = 0.5$, 1 og 2, øvrige parametre som i tabel 4.4. BS-volatiliteten for $\lambda = 0.5$ er 0.1225. Skævheden er hhv. -0.0577, -0.0749 og -0.0815. Kurtosis er hhv. 1.3378, 1.5037 og 1.3356.

sammenhæng mellem parameterværdien og skævhed/kurtosis.

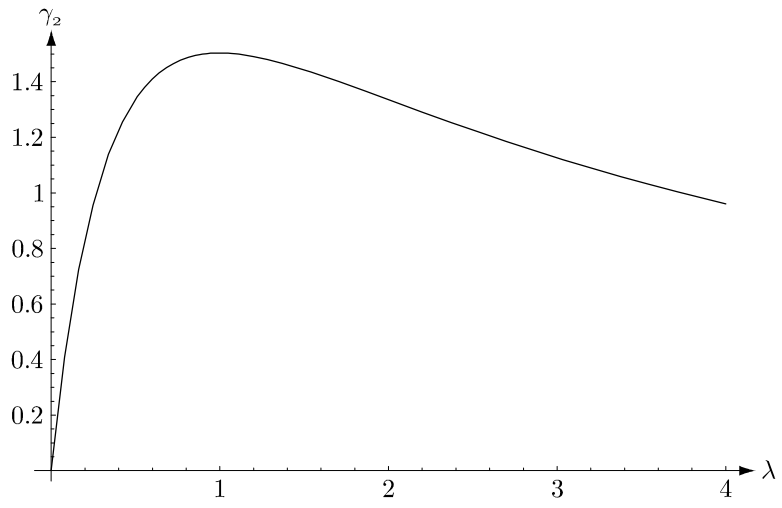
Hvis vi går tilbage til at se på effekten af stokastisk volatilitet og ser denne i sammenhæng med effekten af restløbetiden $T-t$, kan vi se, at en længere løbetid vil forøge effekten af den stokastiske volatilitet, der som nævnt tidligere giver overkurtosis. Denne effekt ses i figur 4.8. Dette betyder, at det er mest relevant at inddrage SV i modellen, når de optioner, der skal prisfastsættes har en lang løbetid.

Hvis vi derimod ser på effekten af spring for forskellige restløbetider $T-t$, vil effekten af spring også være væsentlig for korte løbetider. Dette fremgår af figur 4.9. Således kan anvendelse af en model med spring være ganske relevant, hvis man ønsker at prisfastsætte optioner med kort løbetid.

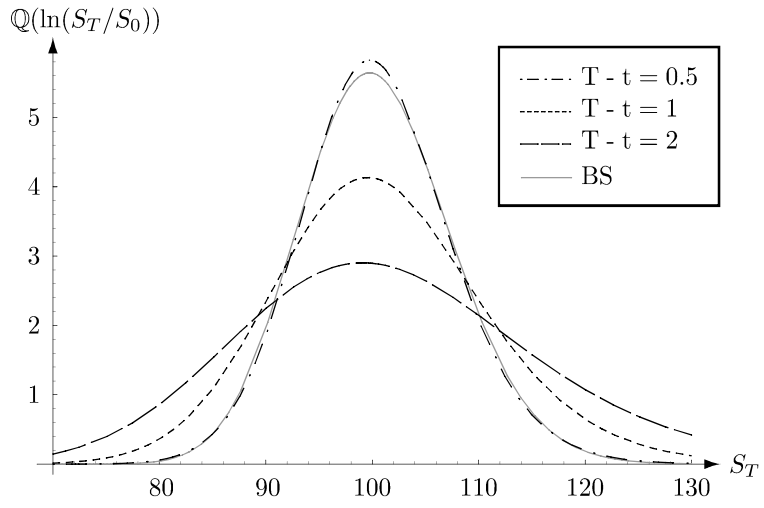
En grundigere analyse af de forskellige effekter af parametervalg for SVJD-modellen, herunder bl.a. effekten af at kombinere en JD-model med negativ skævhed og SV-model med positiv skævhed, kan findes i Nielsen (1999, p. 48 ff.).

4.7 Konsekvenser for prisfastsættelse

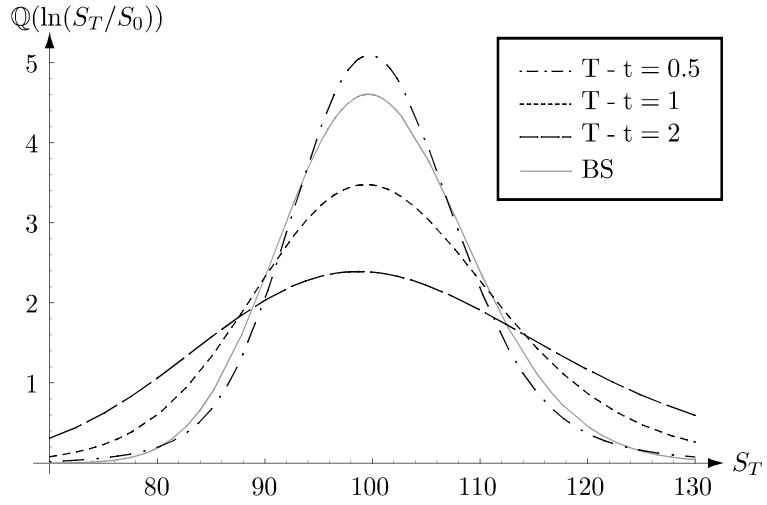
Da det interessante i sidste ende er, hvordan modellen prisfastsætter optioner, vil konsekvenserne af skævhed og kurtosis for priserne blive behandlet med udgangspunkt i to eksempler.



Figur 4.7: Effekten af springintensiteten på kurtosis for udløbskursen afkast. λ er varieret som vist, øvrige parametre som i tabel 4.4.



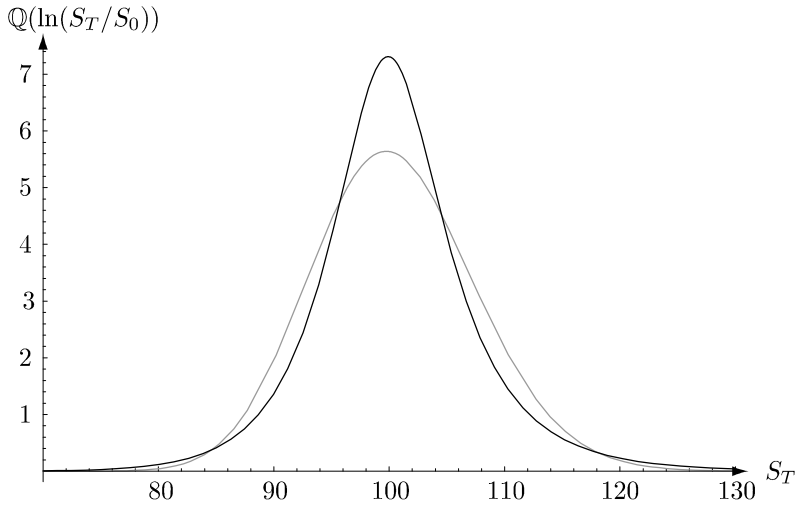
Figur 4.8: Effekten af restløbetiden på udløbskursens sandsynlighedsfordeling ved tilstedeværelse af stokatisk volatilitet. Illustreret for $T - t = 0.5$, 1 og 2, øvrige parametre som i tabel 4.3. BS-volatiliteten for $T - t = 0.5$ er 0.100005. Skævheden er hhv. -0.0089, -0.0143 og -0.0168. Kurtosis er hhv. 0.2522, 0.2858 og 0.2381.



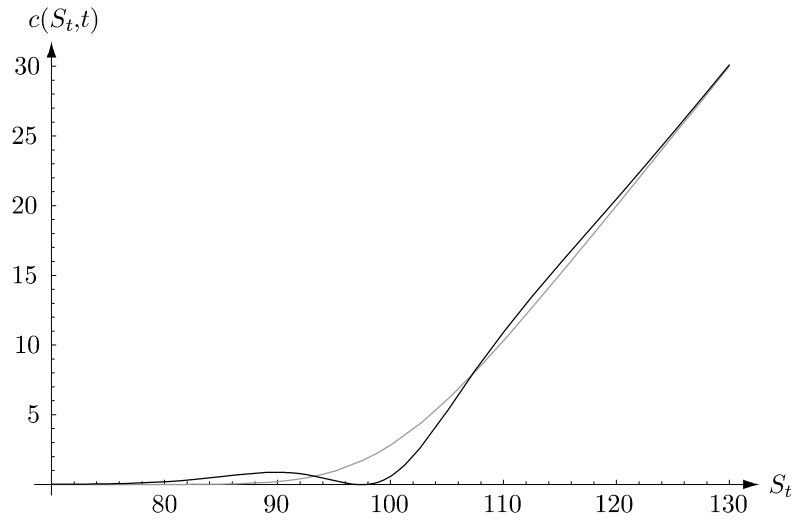
Figur 4.9: Effekten af restløbetiden på udløbskursens sandsynlighedsfordeling ved tilstedeværelse af spring. Illustreret for $T - t = 0.5$, 1 og 2, øvrige parametre som i tabel 4.4. BS-volatiliteten for $T - t = 0.5$ er 0.1225. Skævheden er hhv. -0.0577, -0.0408 og -0.0289. Kurtosis er hhv. 1.3378, 0.6689 og 0.3344.

Hvis fordelingen for afkastet $\ln(S_T/S_0)$ har overkurtosis (fig. 4.10) vil optioner *near-the-money* (optioner, hvor S er tæt på K) være mindre værd end under BS-modellen. Til gengæld vil *far-from-the-money* (optioner, hvor S er langt fra K) være mere værd. (Se fig. 4.11.) Men da der ikke er skævhed i fordelingen, giver overkurtosis næsten den samme effekt på *in-the-money* og *out-of-the-money* optioner (Heston 1993, p. 338). I figur 4.12 ses det, at en fordeling med overkurtosis giver en U-formet sammenhæng mellem aftalekurs og implicit volatilitet, det såkaldte volatilitetssmil.

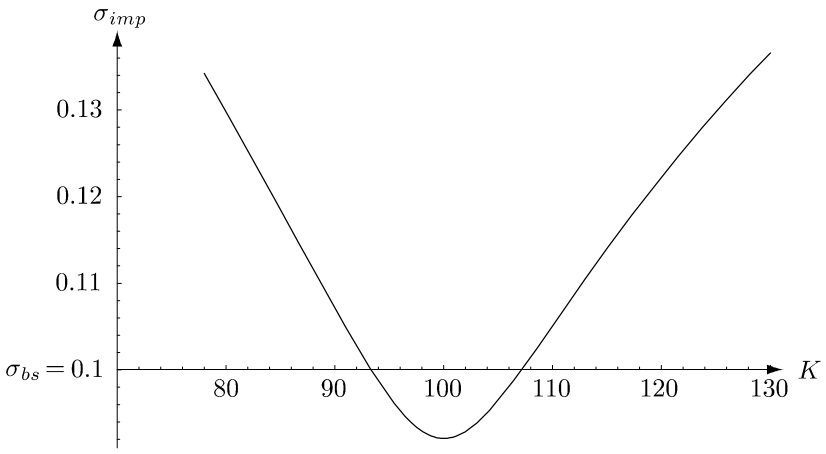
Er der positiv skævhed i afkastet for det underliggende aktiv som vist i eksemplet i figur 4.13, vil købsoptioner *out-of-the-money* være mere værd end i BS tilfældet, idet den længere højre hale gør det mere sandsynligt, at optionen ender *in-the-money*. Omvendt vil en *in-the-money* option være mindre værd. (Se figur 4.14.) Som det ses i figur 4.15 fås en anden sammenhæng mellem implicit volatilitet og aftalekurs, der mere minder om et skævt grin.



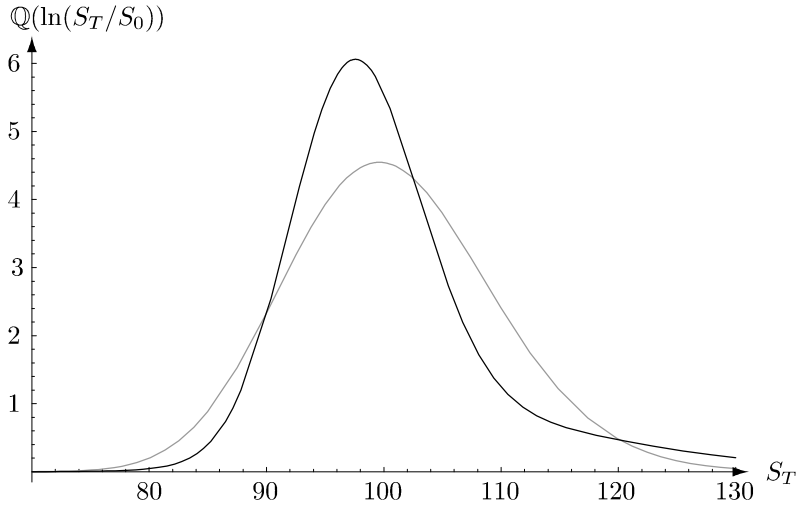
Figur 4.10: Sandsynlighedsfordeling for udløbskursen ved overkurtosis. $\sigma_v = 0.3$, øvrige parameterværdier som i tabel 4.3.



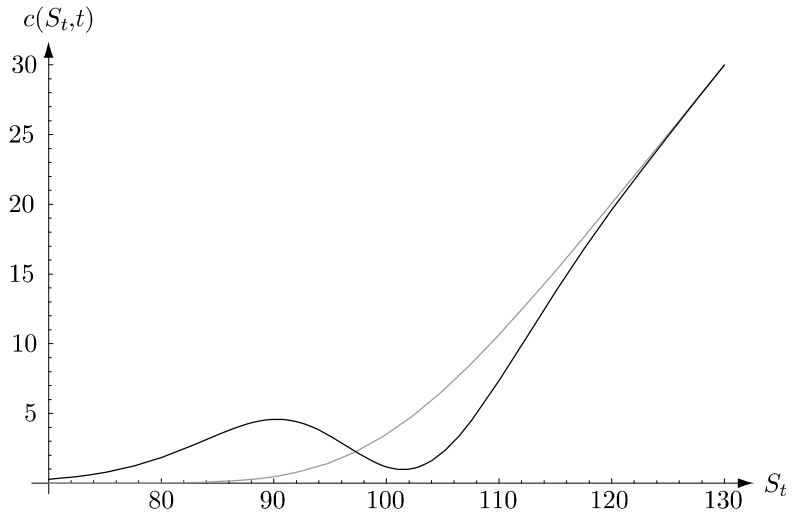
Figur 4.11: Prisningsforskelle ved overkurtosis. Prisforskellene er overdrevet 10 gange. $\sigma_v = 0.3$, øvrige parameterværdier som i tabel 4.3.



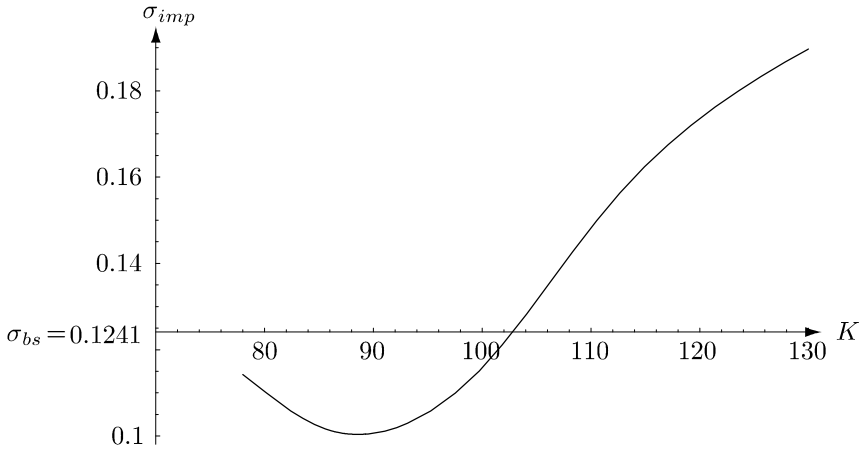
Figur 4.12: Volatilitetssmil ved overkurtosis. $\sigma_v = 0.3$, øvrige parameterværdier som i tabel 4.3. Skæringen med y-aksen indikerer BS-volatiliteten.



Figur 4.13: Sandsynlighedsfordeling for udløbskursen ved skævhed. $\overline{k}^* = 0.1$, øvrige parameterværdier som i tabel 4.4.



Figur 4.14: Prisningsforskelle ved skævhed. Prisforskellen er overdrevet 10 gange. $\overline{k}^* = 0.1$, øvrige parameterværdier som i tabel 4.4.



Figur 4.15: Volatilitetssmil ved skævhed. $\overline{k}^* = 0.1$, øvrige parameterværdier som i tabel 4.4. Skæringen med y-aksen indikerer BS-volatiliteten.

5 Monte Carlo for europæisk option

I dette kapitel introduceres en metode til prisfastsættelse af optioner ved hjælp af simulation, den såkaldte Monte Carlo metode. Først gives en begrundelse for valget af denne frem for andre numeriske metoder, f.eks. endelig differens eller binomial træer. Derefter gives en abstrakt introduktion til metoden, hvorefter den mere konkrete metode for en europæisk option under Black-Scholes modellen gives. Konfidensintervallet præsenteres og der gives metoder, der kan indskrænke dette. Til sidst præsenteres, hvordan stien og dens antitetiske i SVJD-modellen kan simuleres, og det overvejes om en kontrolvariabelteknik kan bruges i dette tilfælde.

Den første anvendelse af Monte Carlo metoden til optionsprisfastsættelse findes i Boyle (1977), mens lærebogstilgange kan findes i Hull (2000b, p. 407–415) og Wilmott (1998, p. 671 ff.). En oversigt over nyere fremskridt for metoden kan findes i Boyle, Broadie & Glasserman (1997).

5.1 Monte Carlo fremfor andre metoder

Der findes andre numeriske metoder end Monte Carlo simulation til at prisfastsætte afledte instrumenter, men idet vi i denne opgave ønsker at prisfastsætte optioner i en model med stokastisk volatilitet og spring, er Monte Carlo metoden lettere at anvende.

De to mest udbredte metoder udover Monte Carlo er binomial træer og endelig differens (*finite difference*), og begge disse kan relativt let udvides til at tage højde for førtidig indfrielse. Både konstruktionen af binomialtræet (der hurtigt vil blive til et flerdimensionalt træ, en *lattice*) og konstruktionen af nettet (*grid*'et) i endelig differens metoden vil imidlertid kompliceres både af elementet af stokastisk volatilitet og springelementet.

SVJD-modellen giver desuden mulighed for spring af vilkårlig størrelse, hvilket ved en umiddelbar anvendelse af de to metoder vil gøre dimensionaliteten uendelig stor, så det vil være nødvendigt med en approksimativ

teknik.

Desuden har Monte Carlo metoden den store fordel, at den er meget nem at tilpasse til nye modeller (f.eks. log-SV modellen) for det underliggende aktiv eller andre optionstyper, herunder stiafhængige optioner (f.eks. asiastiske optioner).

Ulemperne ved Monte Carlo metoden består primært i, at den kræver mange udregninger og derfor er langsom, og at en forbedring af præcisionen kræver mange flere beregninger (se afsnit 5.6). Herudover vil to resultater fra simulationen aldrig være præcist den samme (men forhåbentlig ligge tæt), idet metoden er baseret på tilfældige tal. Disse ulemper er dog af mindre betydning, især på grund af at hurtige computere bliver billigere og billigere, mens finansanalytikere og programmører til at designe og implementere en algoritme ikke gør det.

5.2 Integralet til prisfastsættelse

Til brug for udregning af prisen på et derivat ved hjælp af Monte Carlo simulation kan det være nyttigt at se på udviklingen af det underliggende aktiv som udfald i et sandsynlighedsrum $(\Omega, \mathcal{F}, \mathbb{P})$.

Definition 19 *Ved et* ***sandsynlighedsrum*** *for et underliggende aktiv forstås triplen* $(\Omega, \mathcal{F}, \mathbb{P})$*, hvor* Ω *er mængden af de mulige stier aktivet kan tage.* $\mathcal{F}$ *er mængden af alle mulige filtreringer som defineret i Definition 2.* $\mathbb{P}$ *er et* ***sandsynlighedsmål****, der tildeler enhver* $\mathcal{F}$ *en sandsynlighed:* $\mathbb{P} : \mathcal{F} \to [0, 1]$*, så* $\mathbb{P}(\varnothing) = 0$ *og* $\mathbb{P}(\Omega) = 1$*. En* ***sti*** $\omega \in \Omega$ *angiver en specifik udvikling for kursen på det underliggende aktiv.*[1]

Som det almindeligvis gøres i paradigmet for derivatprisfastsættelse (Cox & Ross 1976) skal prisen findes ved at bruge det risikoneutrale sandsynlighedsmål $\mathbb{Q}$.

Det afledte instruments pris kan derfor udregnes som (Ametrano & Ballabio 2003, Monte Carlo framework)

$$F = \int_{\Omega} f(\omega)\mathbb{Q}(\omega)d\omega$$

hvor f betegner den tilbagediskonterede pris af derivatet givet stien, og $\mathbb{Q}(\omega)$ er sandsynligheden for at stien ω realiseres.

[1] En grundig og kompakt introduktion til disse begreber kan findes i Duffie (2001, Appendix C).

Denne generelle formel giver mulighed for prisfastsættelse af amerikanske optioner samt eksotiske optioner, der afhænger af hele forløbet for kursen på det underliggende aktiv. Et eksempel på disse er de asiatiske optioner, der afhænger af gennemsnittet af kursen over en givet periode.

5.3 Tilnærmelse til integralet med simulation

Det ovenstående integrale kan findes med forskellige numeriske metoder, men den vi behandler her er Monte Carlo metoden. I Monte Carlo metoden findes en approksimation til integralet ved at simulere M stier $(\omega_1, \omega_2, ..., \omega_M)$, der udtrækkes fra $\mathbb{Q}(\omega)$, så de simulerede stier har sandsynlighedsfordelingen repræsenteret ved $\mathbb{Q}$. Herefter findes derivatets værdi som et gennemsnit af den tilbagediskonterede værdi $f(\omega)$ af de enkelte stier:

$$\widehat{F} = \frac{1}{M} \sum_{i=1}^{M} f(\omega_i) \tag{5.1}$$

I tilfældet af, at derivatet er en simpel fordring, og der antages en konstant rente fås formlen:

$$F = \int_{\Omega} e^{-rT} \mathcal{P}(S_T(\omega)) \mathbb{Q}(\omega) d\omega \tag{5.2}$$

Her skal $S_T(\omega)$ ses som den pris, der fremkommer på udløbstidspunktet T ved simulationen ω. Formlen ses at være praksis set ækvivalent med (2.12), der blev præsenteret under udledningen af Black-Scholes formlen.

Integralet fra (5.2) kan i stil med (5.1) approksimeres med

$$\widehat{F} = \frac{1}{M} \sum_{i=1}^{M} e^{-rT} \mathcal{P}(S_T(\omega_i)) \tag{5.3}$$

5.4 Simulation af stien

For at kunne udregne (5.3) skal vi finde de enkelte $S_T(\omega_i)$. Dette skal gøres, så de M S_T'er approksimativt får fordelingen givet ved $\mathbb{Q}$. En simpel måde at gøre dette på er Eulers tilnærmelse.

Definition 20 *(Duffie 2001, p. 298), (Wilmott 1998, p. 672)* ***Euler tilnærmelsen*** *til en geometrisk brownsk bevægelse som defineret i Definition 2.1 er*

$$\begin{aligned} \widehat{S}_{k+1} - \widehat{S}_k &= \mu \widehat{S}_k \Delta t + \sigma \widehat{S}_k \sqrt{\Delta t} \epsilon_k \\ \widehat{S}_0 &= s \end{aligned} \tag{5.4}$$

Her er de T tidsenheder delt op i n intervaller, hver af længde $\Delta t = \frac{T}{n}$, og ϵ_k'erne er tilfældige tal udtrukket fra en standardnormalfordeling.

Præcisionen af denne tilnærmelse afhænger af antallene af tidsintervaller n. For prisfastsættelsen af en europæisk option er det kun nødvendigt med ét tidsinterval, idet kun S_T behøves. Brugen af Euler tilnærmelsen med $n = 1$ ville imidlertid give en for stor upræcision, og det er derfor heldigt, at der for GBM findes et nøjagtigt udtryk.

Integration af udtrykket for $Z_t = \ln S_t$ fra (2.3) efterfulgt af exp taget på begge sider giver

$$S_t = S_0 \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma \int_0^t dW_t\right)$$

Idet vi har $\int_0^t dW_t = \sqrt{t}$ fra *stochastic calculus*, får vi når vi ser på dette over et tidsinterval:

Proposition 12 *En GBM kan simuleres eksakt ved følgende udtryk*

$$\widehat{S}_{k+1} - \widehat{S}_k = \widehat{S}_k \exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\epsilon_k\right) \tag{5.5}$$
$$\widehat{S}_0 = s$$

Idet en simpel fordring kan simuleres med et enkelt tidstrin, vil værdien af en sådan fordring med en valuta som underliggende aktiv kunne findes med følgende formel (med udgangspunkt i (5.3))

$$\widehat{F} = \frac{1}{M}\sum_{i=1}^{M} e^{-rT}\mathcal{P}\left(s_0 \exp\left((r^d - r^f - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\epsilon_1^{(i)}\right)\right) \tag{5.6}$$

Her er $\epsilon_1^{(i)}$ det første tal i den i'te række tilfældige tal (fra standardnormalfordelingen), der netop har et element, idet der jo ikke er brug for flere.

I afsnit C.2 er det vist, hvordan denne formel kan bruges af det medfølgende program.

5.5 Generering af tilfældige tal

At finde rækker af gode tilfældige tal er en videnskab i sig selv (Press et al. 1992, kapitel 7). Når man finder tilfældelige tal med computeralgoritmer vil de næsten altid være produceret på baggrund af deterministiske metoder

og tallene bliver derfor kun pseudo-tilfældige og ikke rigtigt tilfældelige. De mest udbredte algoritmer virker ved, at man giver et frø (*seed*) til den tilfældige talgenerator (*random number generator, RNG*), og RNG'en genererer herefter et for et en lang række af pseudo-tilfældelige tal, der har den ønskede fordeling. Udfordringen er at lave en simpel algoritme, der kan udføres hurtigt, og samtidigt genererer tal, der følger den ønskede fordeling meget præcist. Samtidigt er det vigtigt at rækken af tilfældelige tal ikke begynder at gentage sig selv efter et lille antal genererede tal.

Vi vil ikke i denne opgave dykke dybere i problematikken med om de genererede tal er tilstrækkeligt gode, men blot stole på, at den metode, der er valgt som standard (*default*) ud af en række tilgængelige metoder fra QuantLib toolkit'et, er tilstrækkelig.

5.6 Konfidensinterval og variansreduktion

Det er naturligvis interessant at vide, hvor præcis approksimationen i (5.6) er, og der er faktisk et præcist mål for dette (Boyle 1977, p. 325).

Proposition 13 *Standardafvigelsen for* $\widehat{F}$ *vil (for* M *stor) være*

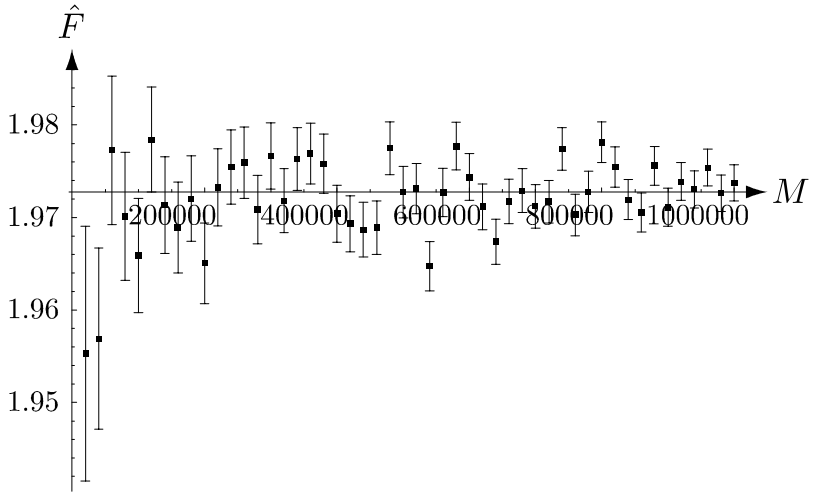
$$\frac{\widehat{s}}{\sqrt{M}}, \tag{5.7}$$

hvor $\widehat{s}$ *er bestemt ud fra den empiriske varians* $\widehat{s}^2 = \frac{1}{M-1}\sum_{i=1}^{M}(f(\omega) - \widehat{F})^2$.

Denne standardafvigelse kan bruges til at bestemme konfidensintervallet for $\widehat{F}$. Et 95% konfidensinterval vil være givet ved $\widehat{F} \pm 1.95996\frac{\widehat{s}}{\sqrt{M}}$, idet 1.96 er 97.5% fraktilen i standardnormalfordelingen.

Konfidensintervallet kan altså indsnævres på to måder. Den ene mulighed er at forøge antallet M af simulerede stier, hvilket er forholdsvist dyrt målt i beregningstid. Ydermere bliver gevinsten ved at tilføje flere stier mindre jo flere stier man har brugt, idet forholdet mellem præcision og antallet af stier er $\frac{1}{\sqrt{M}}$. (Se figur 5.1.) For at få 10 gange så god præcision skal der simuleres 100 gange så mange stier. Den anden mulighed er at reducere størrelsen af variansen $\widehat{s}^2$. Til dette findes en række såkaldte variansreducerende teknikker.

Den letteste at implementere af disse teknikker er antitetiske stier. Dette foregår ved, at man for hver sti samtidigt udregner værdien for derivatet, hvis den "modsatte" sti var blevet fulgt, og herved reducerer variansen uden at skulle generere flere tilfældelige tal. Ved den "modsatte sti" forstås den sti, der fremkommer ved brug af de samme tilfældelige tal men med modsat fortegn, altså formelt rækken $-\epsilon_k^{(i)}$. Hvis vi med en lidt hjemmelavet notation



Figur 5.1: Konvergens for Monte Carlo. Grafen viser optionsværdier udregnet for forskellige antal simulationer samt en standardafvigelse på hver side. Det ses, at der i starten vindes meget ved at forøge antallet af simulationer M, mens der ved mange simulationer kun opnås en lille forbedring ved at forøge antallet. Skæringen med y-aksen indikerer den analytiske pris, 1.97277.

angiver den "modsatte sti" med $-\omega_i$, kan formlen (svarende til (5.3)) således skrives

$$\widehat{F} = \frac{1}{M} e^{-rT} \sum_{i=1}^{M} \frac{\mathcal{P}(S_T(\omega_i)) + \mathcal{P}(S_T(-\omega_i))}{2} \tag{5.8}$$

Blot optionens payoff er monotont, vil en simulation, hvor de antitetiske stier bruges, give mindre varians end en simulation, der bruger dobbelt så mange rå stier (Boyle et al. 1997, p. 1271). Hvor stor forbedringen er afhænger af, hvor høj negativ korrelation, der er mellem de to afkast (Stentoft 2001, p. 32):

$$\mathrm{var}(\widehat{F}) = \frac{\mathrm{var}(\mathcal{P}(S_T(\omega_i)))(1 + \mathrm{corr}(\mathcal{P}(S_T(\omega_i)), \mathcal{P}(S_T(-\omega_i))))}{2M}$$

Ved brug af antitetiske stier skal den empiriske varians udregnes på baggrund af gennemsnittene af payoff for stien og dens antitetiske (altså de enkelte led i summen fra (5.8)) fremfor de $2M$ payoff, der fremkommer ved at tage payoff for de to stier hver for sig. Dette skyldes, at payoff for stien og dens antitetiske taget hver for sig ikke er uafhængige, hvorimod gennemsnittet af de to er det (Boyle et al. 1997, p. 1273), (Stentoft 2001, p. 12, 32).

En anden mulighed er en af de såkaldte kontrolvariabelteknikker (*control-variate techniques*)[2]. Her dannes en portefølje, der er lang i det komplekse derivat, der nødvendigvis skal prisfastsættes med simulation, og kort i et andet derivat, der er simplere og derfor har en analytisk løsning. Dette simple derivat skal samtidig samvariere med det komplekse derivat, således at det opnås, at porteføljen har en lavere varians end de to derivater hver for sig. Herefter beregnes værdien af porteføljen med simulation og den estimerede værdi af det komplekse derivat findes ved at lægge den analytiske løsning for det simple derivat til porteføljeværdien: $(\widehat{F}_K - \widehat{F}_S) + F_S$.

5.7 Monte Carlo for SVJD-modellen

Da der i denne opgave ønskes at nå frem til at kunne prisfastsætte amerikanske optioner i SVJD-modellen vha. simulation, vil der i det følgende blive beskrevet en metode til at simulere en sti i denne model. Denne sti vil blive brugt til at prisfastsætte en europæisk option i modellen. Som for europæiske optioner i BS-modellen, er det som sådant unødvendigt at bruge simulation til at prisfastsætte europæiske optioner i SVJD-modellen. Det er imidlertid en stor fordel at kunne afprøve teknikken i en situation, hvor der findes et fast holdepunkt i form af en analytisk løsning. Med dette som udgangspunkt kan metoden evt. modificeres til modeller, hvor der ikke findes en lukket formel. Desuden kan korrektheden af simulationen testes før den anvendes i algoritmen til prisfastsættelse af amerikanske optioner.

5.7.1 Simulation af SVJD-stien

Da det i forhold til prisfastsættelse er den risikoneutrale sti, der er relevant at simulere, er opgaven at simulere modellen fra (4.1), hvoraf de centrale linier er:

$$\begin{aligned} dS/S &= (r - r_f - \lambda^* \bar{k}^*)dt + \sqrt{V}d\widetilde{W} + k^* d\widetilde{q} \\ dV &= (\alpha - \beta^* V)dt + \sigma_v \sqrt{V} d\widetilde{W}_v \end{aligned} \tag{5.9}$$

Da der ved at simulere $Z_t = \ln S_t$ kan opnås en mere præcis simulation i lighed med (5.5) skal Itôs lemma benyttes til at finde denne.

$$\begin{aligned} dZ_t &= (r^d - r^f - \lambda^* \bar{k}^* - \frac{1}{2}V_t)dt + \sqrt{V_t}dW_t + dJ_t \\ dJ_t &= \ln(1 + k^*)d\widetilde{q} \\ dV_t &= (\alpha - \beta^* V_t)dt + \sigma_v \sqrt{V_t} dW_{v,t} \end{aligned}$$

[2] Wilmott (1998, p. 648), Ametrano & Ballabio (2003, Monte Carlo framework)

Springdelen er her behandlet seperat. En proces, der kun består af en Poisson-proces $dX = Xkdq$ med værdi X^- lige før et spring, vil have værdien $X^+ = X^-(1+k)$ lige efter et spring. Processen $J = \ln X$ vil så have værdien $J^+ = J^- + \ln(1+k)$, og processen får således formen $dJ = \ln(1+k)dq$. Dette led indgår så i Z_t processen.

Hvis vi kigger nærmere på ovenstående processer er der jo både en stokastisk proces V_t samt en Z_t proces, der afhænger af værdien af V_t og desuden skal tillægges springkomponenten J_t.

Først skal V_t altså findes og dette kan gøres med en Euler-tilnærmelse i stil med (5.4):

$$\begin{aligned} \widehat{V}_{k+1} - \widehat{V}_k &= (\alpha - \beta^* \widehat{V}_k)\Delta t + \sigma_v \sqrt{\widehat{V}_k}\sqrt{\Delta t}\epsilon_{v,k} \\ \widehat{V}_0 &= V_0 \end{aligned} \tag{5.10}$$

Her svarer $\epsilon_{v,k}$ til $dW_{v,t}$ og er som tidligere en værdi udtrukket fra en standardnormalfordeling. Idet der ved en simulation af denne type risikeres, at $\widehat{V}_k$ bliver mindre end 0 og derfor giver problemer med negative kvadratrødder, indføres den justering, at $\widehat{V}_k$ sættes lig ε, hvis $\widehat{V}_k < \varepsilon$, hvor ε er et lille positiv tal, f.eks. $\varepsilon = 10^{-5}$.

Herefter skal værdien af springkomponenten findes og den simpleste måde at gøre dette på er med følgende udtryk:

$$\begin{aligned} \widehat{J}_{k+1} - \widehat{J}_k = 0 \quad &, \quad \text{når } \epsilon_{U,k} \geq \lambda^* dt \\ \widehat{J}_{k+1} - \widehat{J}_k = \ln(1+\overline{k}^*) - \frac{\delta^2}{2} + \delta\epsilon_{J,k} \quad &, \quad \text{når } \epsilon_{U,k} < \lambda^* dt \\ J_0 = 0 & \end{aligned} \tag{5.11}$$

Her er $\epsilon_{U,k}$ et tilfældigt tal fra ligefordelingen (U'et står for *uniform*), mens $\epsilon_{J,k}$ er udtrukket fra standardnormalfordelingen (J'et står for *jump*).

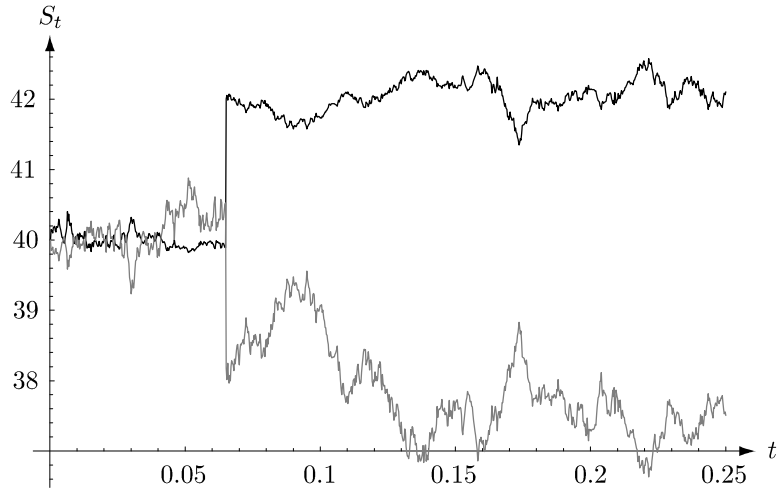
Med $\widehat{V}_k$ og $\widehat{J}_k$ klar kan vi finde Euler-tilnærmelsen til $Z_t = \ln S_t$.

$$\begin{aligned} \widehat{Z}_{k+1} - \widehat{Z}_k &= ((r^d - r^f) - \lambda^* \bar{k}^* - \frac{1}{2}\widehat{V}_k)\Delta t + \sqrt{\widehat{V}_k}\sqrt{\Delta t}\epsilon_{S,k} + \widehat{J}_{k+1} - \widehat{J}_k \\ \widehat{Z}_0 &= \ln S_0 \end{aligned} \tag{5.12}$$

Her svarer $\epsilon_{S,k}$ til dW. Vi kan herefter finde $\widehat{S}$ ud fra $\widehat{Z}$:

$$\widehat{S}_{k+1} - \widehat{S}_k = \widehat{S}_k \exp(\widehat{Z}_{k+1} - \widehat{Z}_k)$$

Alternativt kan S_t i den simulerede sti findes som tilnærmelsesvis lig $\exp(\widehat{Z}_k)$, når $k\Delta t = t$. Simulationen er imidlertid ikke eksakt, som den var



Figur 5.2: En simulation af en sti og dens antitetiske i SVJD-modellen. Parametre som mjump i tabel 7.1, p. 88. Læg mærke til, at når volatiliteten for den ene sti går op, så går volatiliteten for den anden sti ned.

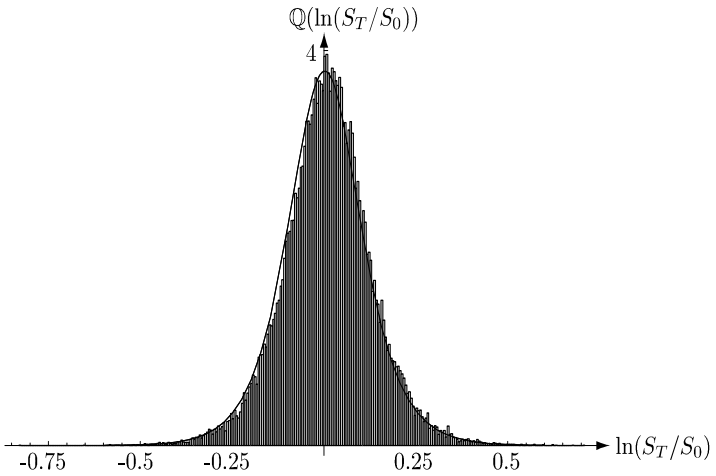
i (5.5). Dette skyldes, at $\widehat{Z}$ afhænger af $\widehat{V}$, og således kommer præcisionen til at afhænge af, hvor findelt simulationen af volatiliten er. Konvergensen for Monte Carlo kommer på denne måde til at afhænge af både M og Δt (Wilmott 1998, p. 675).

Den antitetiske sti simuleres på tilsvarende vis. Som beskrevet i afsnit 5.6 bruges til dette formål de samme genererede tilfældige tal, men de negeres på følgende måde: $-\epsilon_{S,k}$, $-\epsilon_{v,k}$, $-\epsilon_{J,k}$. Derimod negeres $\epsilon_{U,k}$ ikke, hvilket betyder, at springene indtræffer på samme tidspunkt i den normale sti og den antitetiske sti (se figur 5.2).

I afsnit C.4 beskrives, hvordan programmet kan bruges til at prisfastsætte europæiske optioner i SVJD-modellen. Kildekoden kan ses i afsnit D.3.1 og D.3.2.

5.7.2 Mulige forbedringer af simulationen

Såfremt simulationen af SVJD-stien er korrekt, vil fordelingen af de simulerede værdier $\ln(\widehat{S}_T/S_0)$ tilnærmelsesvist følge den analytiske tæthedsfunktion for $\ln(S_T/S_0)$ givet i 4.9. Dette er illustreret for realistiske parameterværdier med $\rho \neq 0$ i figur 5.3. Det ses, at fordelingen for de simulerede værdier er en anelse mere skæv end fordelingen givet ved den analytiske formel. Denne effekt er mere udtalt for højere værdier af ρ eller lavere antal tidsinddelinger, og tyder på, at Monte Carlo simulationen bliver unøjagtig for $\rho \neq 0$. Det



Figur 5.3: Sammenligning af histogrammet for de simulerede værdier $\ln(\widehat{S}_T/S_0)$ med tæthedsfunktionen for $\ln(S_T/S_0)$. De brugte parametrene er estimerede implicitte parametre fra Bates (1996, p. 86) $S_0 = 40$, $r^d = 0.08$, $r^f = 0.06$, $\sqrt{V_0} = 0.155$, $\sigma_v = 0.343$, $\sqrt{\frac{\alpha}{\beta^*}} = 0.155$, $\beta^* = 0.78$, $\rho = 0.078$, $\lambda^* = 15.01$, $\overline{k}^* = -0.001$, $\delta = 0.019$. Der er simuleret 50000 stier med 1000 tidstrin.

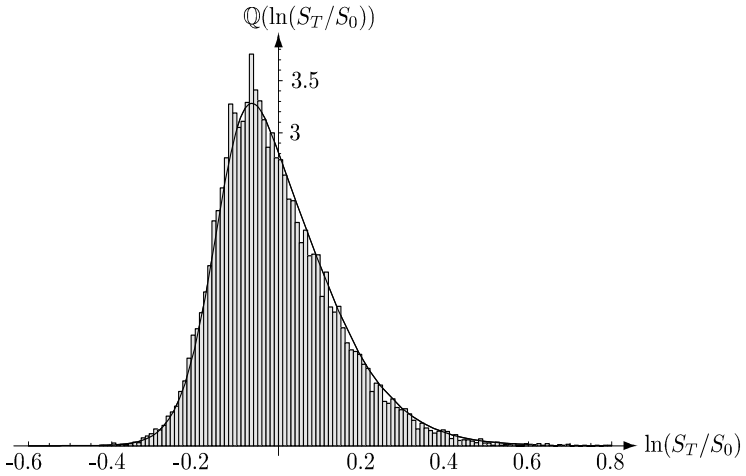
viste fit ville kunne forbedres en anelse ved at inkludere de antitetiske stier, mens flere tidsinddelinger tilsyneladende ikke hjælper.

Simulationen af SV-stien kan forbedres ved at bruge en højere ordens tilnærmelsesformel, f.eks. Milshteins tilnærmelse (Duffie 2001, p. 298), i formel (5.10) og (5.12). Simulationen af volatiliteten kan forbedres ved at bruge en Euler-tilnærmelse af processen $Y_t = V_t \exp(-\beta^* t)$ som vist i Frühwirth-Schnatter & Sögner (2003, p. 14, formel (29)). Om disse forbedringer løser problemet med, at $\rho \neq 0$ giver unøjagtige simulationer, er dog uvist.

I figur 5.4 er vist den tilsvarende graf for en model med spring. Her er det helt nødvendigt at bruge 1000 tidsinddelinger eller mere for at de to fordelinger matcher nøjagtigt. Desuden er de antitetiske stier inkluderet, hvilket giver en stor forbedring af fittet.

Den simple metode i formel (5.11) til simulation af spring kræver et stort antal tidsintervaller for at være præcis. Simulationen kan imidlertid forbedres ved at udnytte at tiden mellem to spring er eksponentialfordelt med parameter λ. Hvis diskretisering ydermere planlægges, så punkterne falder sammen med springtidspunkter, kan en ret præcis simulering findes uden brug af mange tidsintervaller (Cyganowski, Grüne & Kloeden 2002, p. 7).

En lovende mulighed for en samtidig forbedring af både volatilitet- og spring-delen af processen ligger i at bruge den simulatane fordeling af kurs og volatilitet til at simulere udviklingen af tilstandsvariablene. Bates (2003,



Figur 5.4: Sammenligning af histogrammet for de simulerede værdier $\ln(\widehat{S}_T/S_0)$ med tæthedsfunktionen for $\ln(S_T/S_0)$. Parametre som mjump i tabel 7.1, p. 88, dog med $k^* = 0.1$. Der er simuleret 10000 stier samt de antitetiske med 1000 tidstrin.

p. 21)beskriver i forbindelse med udarbejdelse af en metode til parameterestimation, hvordan den karakteristiske funktion for den simultane fordeling af kursen og volatiliteten ser ud. Denne vil formentlig kunne udnyttes til at finde de fremtidige værdier på baggrund af to tilfældige tal udtrukket fra ligefordelingen. I lighed med prisningsformlerne i 4.9 er det dog nødvendigt med en numerisk integration for at finde frem til sandsynligheden. Dette vil medføre en længere beregningstid, og det er ikke sikkert, at den vundne præcision kan opveje dette.

5.7.3 Kontrolvariabelteknik

For en europæisk option prisfastsat på en SVJD-sti findes der ikke en oplagt alternativ pris, man kan bruge som kontrolvariabel. Selvfølgelig kunne man bruge den analytiske pris som udregnet tidligere, men dette ville være snyd, idet det jo netop er den pris, vi forsøger at finde. Med notationen fra afsnit 5.6 ville $\widehat{F}_K = \widehat{F}_S$, og således $(\widehat{F}_K - \widehat{F}_S) + F_S = F_S$. Resultatet bliver altså blot den analytiske pris og den udførte Monte Carlo simulation ville hverken gøre fra eller til. Man kunne forestille sig, at man kunne bruge prisen på en europæisk option i BS-modellen som kontrolvariabel. Dette ville imidlertid ikke være metodisk korrekt, da denne ikke ville være prisfastsat på samme sti.

I Lewis (2000, p. 104) og Srikant (1998) er der beskrevet andre metoder,

hvorpå man kan forbedre Monte Carlo metoden i SV-tilfældet.

6 Monte Carlo for amerikansk option

I dette kapitel præsenteres least-squares Monte Carlo metoden til prisfastsættelse af amerikanske optioner. Som baggrund for denne introduceres først de generelle metoder til prisfastsættelse af amerikanske optioner, herunder begreberne indfrielsesstrategi og indfrielsesgrænse. Herefter omtales hvilke alternative metoder, man kan bruge til prisfastsættelsen, og hvorfor de fleste af disse kun vanskeligt kan gøres praktiske i brug til SVJD-modellen. Least-squares Monte Carlo teknikken gennemgås og resultater, der beviser metodens konvergens til den sande optionsværdi, præsenteres kort. Da mindste kvadraters metode er en central del af algoritmen gennemgås denne. De ændringer til standardalgoritmen, der er foretaget for effektivt at kunne bruge den i SVJD-tilfældet, præsenteres herefter, og der vises metoder, der kan reducere algoritmens hukommelsesforbrug. Til sidst illustreres den indfrielsesgrænse, der er implicit givet ved Longstaff-Schwartz algoritmen.

6.1 Indfrielsesstrategi for amerikansk option

I dette afsnit introduceres de generelle koncepter til prisfastsættelse af amerikanske optioner. Gennemgangen er baseret på Duffie (2001, p. 31, p. 182).

En amerikansk option (se definition 4, p. 17) adskiller sig fra en europæisk option ved, at den kan indfries på ethvert tidspunkt $0 \leq t \leq T$. Med udgangspunkt i S_t, defineres den tilbagediskonterede afkastproces ved $Z_t = e^{-rt}\mathcal{P}(S_t)$. Optionen vil give køberen et afkast med nutidsværdi Z_τ, hvor τ er valgt af køberen. Tidspunktet τ kaldes en indfrielsesstrategi *(exercise strategy)*. Køberen skal ikke lægge sig fast på en indfrielsesstrategi på forhånd, men kan vente til tidspunkt τ med at sige, at han nu ønsker at indfri optionen.

Givet en indfrielsesstrategi τ kan der på samme måde som for europæiske optioner argumenteres for, at afkastprocessen Z_t for den amerikanske option kan replikeres med en selvfinansierende porteføljestrategi h (Duffie 2001,

p. 183). h's værdiproces er givet ved

$$V_t^\tau = \mathbb{E}_\mathbb{Q}(Z_\tau \mid \mathcal{F}_t)$$

Dette medfører, at betalingen til køber på tidspunkt τ kan replikeres ved at følge porteføljestrategien h^τ, der er h indtil tidspunkt τ og 0 herefter. Denne strategi h^τ vil netop give en betaling med nutidsværdi Z_τ på tidspunkt τ.

En rationel indfrielsesstrategi τ^* (*rational exercise strategy*) defineres nu som det τ, der maksimerer den arbitragefri værdi af optionen på tidspunkt 0. Denne værdi findes ved at løse følgende optimale stopproblem (*optimal-stopping problem*), der netop finder supremum (altså maksimum) over de mulige stoptider af værdien af den porteføljestrategi, der replikerer Z_τ. Dette er samtidigt den maksimale forventede værdi af Z_τ.

$$V_0^* = \sup_{\tau \in [0,T]} V_0^\tau = \sup_{\tau \in [0,T]} \mathbb{E}_\mathbb{Q}(Z_\tau \mid \mathcal{F}_0) \tag{6.1}$$

For at finde løsningen τ^* på ovenstående problem introducerer vi en *Snell envelope* defineret som[1]

$$\begin{aligned} U_T &= Z_T \\ U_t &= \operatorname{ess\,sup}_{\tau \in [t,T]} \mathbb{E}_\mathbb{Q}(Z_\tau \mid \mathcal{F}_t) \end{aligned} \tag{6.2}$$

U_t vil således være værdien (tilbagediskonteret til tidspunkt 0) af optionen på tidspunkt t givet at den bliver indfriet optimalt i fremtiden. U_0 vil være optionsværdien på tidspunkt 0, $U_0 = V_0^*$.

Med denne definition er det muligt at bevise følgende resultat

Proposition 14 *(Duffie 2001, p. 184) Der eksisterer en super-replikerende porteføljestrategi h^*, hvis værdiproces W^{h^*} har initialværdi $W_0^{h^*} = V_0^*$. En rational indfrielsesstrategi er givet ved $\tau^* = \inf\{t \mid W_t^{h^*} = Z_t\}$.*

Super-replikerende betyder blot, at hvis optionskøberen ikke indfrier optionen optimalt, vil optionssælgeren, der samtidigt replikerer fordringen, få et overskud. Resultatet er således analogt til arbitrageresultatet for en europæisk option fra afsnit 2.5.

[1] ess sup betyder essentielt supremum og er et teknisk raffinement af maksimum, så værdier af funktionen på en mængde med mål 0 ikke har indflydelse på supremummet (Weisstein 2003, EssentialSupremum).

6.2 Indfrielsesgrænsen

Et centralt begreb i behandlingen af amerikanske optioner er indfrielsesgrænsen, der for hvert tidspunkt angiver ved hvilken pris, det er optimalt at indfri optionen.

Der ses nu på en option givet ved $Z_t = g(s,t)$ (typisk $g(s,t) = e^{-rt}\mathcal{P}(s)$ som i sidste afsnit), og der defineres $h(s,t) = \sup_{\tau\in[t,T]} \mathbb{E}_{\mathbb{Q}}(g(S_\tau,\tau))$, hvor processen S starter i $S_t = s$. g er altså optionens umiddelbare indfrielsesværdi, og h er optionens værdi givet, at den indfries optimalt i fremtiden. Det ses let, at $h(s,t) \geq g(s,t)$. Idet der fra proposition 14 følger, at en optimal indfrielsesstrategi er givet ved $\tau^* = \inf\{t \mid h(S_t,t) = g(S_t,t)\}$, kan indfrielsesregionen E *(exercise region)* defineres som

$$E = \{(s,t) \in \mathbb{R}\times[0,T] \mid h(s,t) = g(s,t))\}$$

Vi kan nu skrive $\tau^* = \inf\{t \mid (S_t,t) \in E\}$, og det vil være optimalt at indfri optionen, når (S_t,t) er i E, og ellers beholde den.

Resultatet omkring indfrielsesregionen ovenfor er generelt, mens vi ved at betragte en amerikansk salgsoption kan give resultatet en mere ligefrem formulering (Duffie 2001, p. 188). I dette tilfælde kan den optimale indfrielsesgrænse *(optimal exercise boundary)* beskrives med en kontinuert og differentiabel funktion $\overline{S}$ af tiden $t \in [0,T[$. Med $\overline{S}_T = K$, kan det optimale stoptidspunkt beskrives med $\tau^0 = \inf\{t \mid S_t \leq \overline{S}_t\}$. Således er den optimale indfrielsesstrategi for køberen at indfri optionen, når den rammer indfrielsesgrænsen.

Der er ikke nogen eksplicit formel for indfrielsesgrænsen, og den skal derfor findes med f.eks. en endelig differens metode. Vi ved dog, at $\overline{S}_T = K$, og at $\lim_{t\to T} S_t = K$, når $r^f \leq r^d$, og $\lim_{t\to T} S_t = K\frac{r^d}{r^f}$, når $r^f > r^d$ (Basso, Nardon & Pianca 2002). Intuitionen i dette er, at hvis den forventede stigning på valutakursen $r^d - r^f$ er negativ, vil der være tilfælde, hvor det bedre kan betale sig at indfri optionen straks og indkassere den indenlandske rente. Mere matematisk kan det formuleres, at det er optimalt at indfri optionen på tidspunkt t, hvis $K - S_t \geq e^{-r^d}(K - S_t e^{(r^d-r^f)(T-t)})$. Dette giver $S_t \leq K\frac{(1-e^{-r^d(T-t)})}{(1-e^{-r^f(T-t)})}$, der går mod $S_t \leq K\frac{r^d}{r^f}$, når $T-t$ går mod 0.

6.3 Begrænsning ved standard Monte Carlo til amerikanske optioner

I mange år var det betragtet som en praktisk umulighed at benytte Monte Carlo simulation til at prisfastsætte amerikanske optioner. I princippet kan

man godt finde en tilnærmelse til $V_0^* = \sup_{\tau \in [0;T]} \mathbb{E}(Z_\tau \mid \mathcal{F}_0)$ med standard Monte Carlo. Problemet er imidlertid, at man for at kunne vide, om det er optimalt at indfri optionen på et tidspunkt t, skal sammenligne Z_t med den forventede værdi af at beholde optionen. Hvis man til dette formål bruger en ny Monte Carlo simulation, vil antallet af simulationer vokse eksponentielt og gøre metoden ubrugelig i praksis.

6.4 Alternative metoder til prisfastsættelse af amerikanske optioner

Der findes naturligvis andre muligheder til prisfastsættelse af amerikanske optioner. De fleste af disse metoder er mere velegnede, sålænge optionen kun afhænger af én tilstandsvariabel. Da endemålet med denne opgave imidlertid er at prisfastsætte amerikanske optioner i SVJD-modellen, vil disse metoder skulle gøres mere komplekse i større eller mindre grad. Her vil kort blive nævnt de mest udbredte metoder.

Sammen med Monte Carlo metoden var brugen af binomialtræer en af de først brugte numeriske metoder til prisfastsættelse af optioner (Cox, Ross & Rubinstein 1979). Denne metode er ganske enkel at modificere til at tage højde for førtidig indfrielse (Hull 2000b, p. 210). Tilføjelsen af volatiliteten som ekstra tilstandsvariabel komplicerer imidlertid billedet en del, og muligheden for spring af vilkårlig størrelse gør, at der skal fundamentale ændringer i metoden til, for at den kan benyttes.

En anden mulighed er endelig differens metoden, der normalt er den mest oplagte metode til amerikanske optioner under Black-Scholes-modellen. Imidlertid giver udvidelserne med spring problemer på samme måde som for binomialtræerne. Der kan dog findes en implementation af Hestons SV-model, der jo er et specialtilfælde af SVJD-modellen, i Kluge (2002). Bates (1996, p. 78, 80) omtaler, men beskriver ikke, en endelig differens metode, der tager højde for spring.

Bates (1996, p. 77-78) bruger en analytisk tilnærmelse til at tage højde for førtidig indfrielse i SVJD-modellen, og ved en sammenligning med hans endelig differens metode finder han, at afvigelserne er ganske små. Denne formel er relativt let at implementere og hurtig at beregne, så den er brugbar, hvis høj præcision ikke er nødvendig. Da den som beskrevet i Bates (1991, p. 1026) er en udvidelse af formlen fra Barone-Adesi & Whaley (1987), har den imidlertid det handicap, at priserne ikke bliver særligt nøjagtige, og at der ikke findes en parameter at skrue på, der kan øge nøjagtigheden (Ju 1998, p. 628).

Endnu en mulighed er at finde en approksimation til indfrielsesgrænsen (*exercise boundary*). Dette er en metode, der bruges i f.eks. Ju (1998). Her udnyttes det, at såfremt man kender indfrielsesgrænsen, kan optionsprisen findes som prisen for den europæiske option plus et integrale, der afhænger af funktionen for indfrielsesgrænsen. Denne funktion tilnærmes som en stykvis eksponentiel funktion og prisen kan herefter findes med hurtige numeriske metoder. Tzavalis & Want (2003) bruger en lignende teknik, der bruges til at prisfastsætte optioner i Hestons model, hvor indfrielsesgrænsen er todimensionel. Om denne teknik med held kan udvides til modeller, hvor der indgår spring, er endnu et åbent spørgsmål.

Der findes også alternative måder, hvorpå man kan bruge Monte Carlo simulation til at prisfastsætte amerikanske optioner. Heraf er metoden forslået i Broadie & Glasserman (1997) den mest omtalte i litteraturen. Denne metode baserer sig på at prisudviklingen for det underliggende aktiv simuleres i en træstruktur. Herefter bruges denne information til at finde et estimat for optionens værdi, der altid vil ligge over den sande værdi, og et andet estimat, der altid vil ligge under den sande værdi. Begge disse estimater vil konvergere mod den sande værdi, og således kan metoden finde et interval, hvori optionens værdi ligger. På grund af træstrukturen er metoden ikke så nem at kombinere med et sædvanligt Monte Carlo setup, hvor én sti simuleres ad gangen. Bl.a. derfor er metoden noget besværligere at implementere end Longstaff-Schwartz metoden. Desuden er der ikke resultater, der tyder på, at den giver mere præcise resultater. Den har dog den fordel, at hukommelsesforbruget er mindre end for LSM, idet alle stier ikke behøves at blive lagret i hukommelsen på én gang. En implementation af denne metode kan findes i Eskildsen (2002).

6.5 Least-squares Monte Carlo

I de følgende afsnit vil least-squares Monte Carlo (LSM) metoden introduceret i Longstaff & Schwartz (2001) blive beskrevet. Fremstillingen følger Clément, Lamberton & Protter (2002).

Det første skridt er at omskrive det optimale stopproblem fra kontinuert tid til diskret tid. Herved bliver det muligt at bruge de simulerede Monte Carlo stier, der jo i deres natur er diskrete, til at udregne den initiale optionsværdi U_0.

S og Z er nu defineret som diskrete processer $(S_j)_{j=0,\ldots,n}$ og $(Z_j)_{j=0,\ldots,n}$ tilpasset den diskrete filtrering $(\mathcal{F}_j)_{j=0,\ldots,n}$. Z_j er stadig givet som funktion af processen for det underliggende aktiv $Z_j = e^{-rj\Delta t}\mathcal{P}(S_j)$. S_j og derfor Z_j antages at være en Markov-processer, og der gælder derfor $\mathbb{E}(Z_\tau \mid \mathcal{F}_j) =$

$\mathbb{E}(Z_\tau \mid S_j)$.

Med denne diskretisering bliver (6.1) til

$$V_0^* = \sup_{\tau \in \{0,\ldots,n\}} \mathbb{E}(Z_\tau \mid \mathcal{F}_0)$$

Med en ligefrem omskrivning af (6.2) bliver Snell envelopen i diskret tid $(U_j)_{j=0,\ldots,n}$

$$\begin{aligned} U_n &= Z_n \\ U_j &= \operatorname{ess\,sup}_{\tau \in \{j,\ldots,n\}} \mathbb{E}(Z_\tau \mid \mathcal{F}_j) \end{aligned}$$

Da Snell envelopen ikke umiddelbart lader sig implementere, omskrives denne til det dynamiske programmerings princip som følger. På hvert tidspunkt j sammenlignes afkastet ved øjeblikkelig indfrielse Z_j med det forventede afkast ved at fortsætte $\mathbb{E}(U_{j+1} \mid \mathcal{F}_j)$. Herved bliver det muligt at definere U_j rekursivt, og således finde frem til U_0.

$$\begin{aligned} U_n &= Z_n \\ U_j &= \max(Z_j, \mathbb{E}(U_{j+1} \mid \mathcal{F}_j)), \text{ for } 0 \leq j \leq n-1 \end{aligned}$$

Ved at definere τ_j i stil med definitionen af den rationelle indfrielsestaktik fra proposition 14

$$\tau_j = \min(k \geq j \mid U_k = Z_k)$$

kan U_j også skrives som

$$U_j = \mathbb{E}(Z_{\tau_j} \mid \mathcal{F}_j)$$

Specielt har vi, at $\mathbb{E}(U_0 \mid \mathcal{F}_0) = \mathbb{E}(Z_{\tau_0} \mid \mathcal{F}_0) = \sup_{\tau \in \{0,\ldots,n\}} \mathbb{E}(Z_\tau \mid \mathcal{F}_0)$, der jo netop er den ønskede optionsværdi.

Det dynamiske programmerings princip kan også formuleres ved hjælp af optimale stoptidspunkter τ_j

$$\begin{aligned} \tau_n &= n \\ \tau_j &= j\mathbf{1}_{A_j} + \tau_{j+1}\mathbf{1}_{A_j^c}, \text{ for } 0 \leq j \leq n-1 \end{aligned}$$

hvor $\mathbf{1}_{A_j}$ er indikatorfunktionen, der er lig 1, når $Z_j \in A_j$, og 0 ellers, A_j^c er A_j' komplement og

$$A_j = \{Z_j \geq \mathbb{E}(Z_{\tau_{j+1}} \mid \mathcal{F}_j)\}$$

A_j kan også skrives

$$A_j = \{Z_j \geq U_{j+1}\}$$

Og den rekursive definition kan formuleres for U

$$\begin{aligned} U_n &= Z_n \\ U_j &= Z_j \mathbf{1}_{A_j} + U_{j+1}\mathbf{1}_{A_j^c}, \text{ for } 0 \leq j \leq n-1 \end{aligned}$$

Tiden er nu kommet til at introducere det fundamentale skridt i Longstaff-Schwartz metoden, nemlig at den betingede forventning på tidspunkt j $\mathbb{E}(Z_{\tau_{j+1}} \mid \mathcal{F}_j)$ kan tilnærmes ved hjælp af regression med mindste kvadraters metode. Til dette bruger vi m basisfunktioner, der for formelt at kunne garantere konvergensen af metoden, skal være de m første funktioner i en række, der udgør en basis i et Hilbert rum. Det kan f.eks. være Legendre polynomierne som defineret i afsnit 4.4.1. I praksis viser det sig imidlertid, at standardmonomierne $1, x, x^2, ..., x^{m-1}$ giver tilstrækkelig konvergens.

Basisfunktionerne betegnes med $(e_k(x))_{k\geq 1}$. Med $P_j^m(z)$ betegnes projektionen af værdien af z ned på rummet udspændt af $\{e_1(S_j), ..., e_m(S_j)\}$. For $m < \infty$ vil projektionen kun være en tilnærmelse, og med $\tau_j^{[m]}$ betegnes den stoptid, der fremkommer, når der bruges m funktioner til tilnærmelsen. Med denne notation fås

$$\begin{aligned} \tau_n^{[m]} &= n \\ \tau_j^{[m]} &= j\mathbf{1}_{A_j} + \tau_{j+1}^{[m]}\mathbf{1}_{A_j^c} \end{aligned}$$

hvor $A_j = \{Z_j \geq P_j^{[m]}(Z_{\tau_{j+1}}^{[m]}) \wedge Z_j > 0\}$. Betingelsen $Z_j > 0$ gør, at kun de stier, hvor optionen er *in-the-money* bliver betragtet. Dette gør, at metoden bliver numerisk mere effektiv.

Idet der antages, at kursen S_0 er kendt på tidspunkt 0 og Z_0 således er deterministisk, haves følgende tilnærmelse til værdien på tidspunkt 0:

$$U_0^m = \max(Z_0, \mathbb{E}(Z_{\tau_1^{[m]}} \mid \mathcal{F}_1))$$

Der er imidlertid endnu ikke fundet en numerisk værdi for U_0^m. Derfor udføres en yderligere tilnærmelse, der bruger Monte Carlo simulation til at tilnærme $\mathbb{E}(Z_{\tau_1^{[m]}} \mid \mathcal{F}_1)$. Til dette formål simuleres der ligesom ved standard Monte Carlo M stier med n tidsinddelinger $(S_j^{(1)}, ..., S_j^{(M)})$. I dette tilfælde er det imidlertid nødvendigt at gemme de resulterende simulerede stier, så de alle er til rådighed samtidigt, idet dette gør det muligt at udnytte informationen på tværs af stierne.

For hver sti $i \in \{1, ..., N\}$ kan det optimale stoptidspunkt nu findes ved at arbejde baglæns:

$$\begin{aligned} \tau_n^{i,m,M} &= n \\ \tau_j^{i,m,M} &= j\mathbf{1}_{A_j} + \tau_{j+1}\mathbf{1}_{A_j^c} \end{aligned}$$

hvor vi nu bruger

$$A_j = \{Z_j \geq \alpha_j^{(m,M)} \cdot e^m(S_j^{(i)}) \wedge Z_j > 0\} \tag{6.3}$$

Her betegner $x \cdot y$ det sædvanlige prikprodukt mellem to vektorer, og α_j er parametrene i regressionen, der findes med

$$\alpha_j^{(m,M)} = \arg\min_{a \in \mathbb{R}^m} \sum_{i=1}^{M} \left(\mathbf{1}_{\{Z_j^{(i)} > 0\}} (Z^{(i)}_{\tau_{j+1}^{i,m,M}} - a \cdot e^m(S_j^{(i)}) \right)^2 \tag{6.4}$$

Igen er $\mathbf{1}_{\{Z_j^{(i)}>0\}}$ taget med for at sørge for at kun *in-the-money* stier tages med i regressionen.

Ud fra de fundne variable $\tau_j^{i,m,M}$ bliver tilnærmelsen til U_0^m således

$$U_0^{m,M} = \max\left(Z_0, \frac{1}{M} \sum_{i=1}^{M} Z^{(i)}_{\tau_1^{i,m,M}} \right) \tag{6.5}$$

Dette er den endelige optionsværdi, og der kan skrives

$$\widehat{F}(0, S_0) = U_0^{m,M}$$

I formlerne (6.4) og (6.5) kan $Z^{(i)}_{\tau_{j+1}^{i,m,M}}$ erstattes med $U_{j+1}^{(i)}$, idet de netop er lig hinanden. Dette er en fordel i implementeringen af algoritmen, idet det bliver unødvendigt at holde styr på Z-værdierne.

Kildekoden til algoritmen kan findes i afsnit D.4.2 og brugen er demonstreret i afsnit C.6.

6.6 Konvergens-resultater

Læg mærke til, at der i løbet af proceduren er udført 3 tilnærmelser. Først og fremmest behandles problemet i diskret tid og der bruges en Euler-tilnærmelse el.lign. til at simulere Monte Carlo stierne. Denne tilnærmelse kan forbedres ved at skrue på parameteren n, der er antallet af tidsintervaller. Dernæst tilnærmer vi U_0 ved at projicere Z_j ned på et underrum bestående af m basisfunktioner. Denne tilnærmelse U_0^m kan forbedres ved at forøge m. Til

sidst udføres regressionen ved at bruge informationen på tværs af de M stier, hvorved vi får $U_0^{m,M}$. Igen er tilnærmelsen bedre, jo flere stier M der bruges.

På trods af alle disse tilnærmelser kan det bevises, at den tilnærmede værdi $\widehat{F}(0, S_0)$ fundet med Longstaff Schwartz metoden konvergerer mod den sande værdi $F(0, S_0)$, når n, m og M går mod uendelig. I Longstaff & Schwartz (2001, p. 125) sandsynliggøres det, at dette er rigtigt, mens grundige beviser findes i Clément et al. (2002, p. 455) og Egloff & Min-Oo (2002). Disse inkluderer dog ikke bevis for en konvergens til den sande optionsværdi, men kun et bevis for konvergensen af værdien efter at stien er diskretiseret. Beviset for konvergensen af den diskrete tilnærmelse til det kontinuerte tilfælde er mere kompliceret (Egloff & Min-Oo 2002, p. 4).

Et andet nyttigt resultat er følgende, der siger, at værdien fundet med simulation vil være opad begrænset af den sande værdi. Dette virker intuitivt, idet en hvilken som helst anden stopregel end den optimale, og altså hermed også stopreglen fundet med LSM, vil resultere i en værdi for den amerikanske option, der er lig med eller mindre end værdien ved optimal indfrielse. Lidt løst formuleret:

Proposition 15 *(Longstaff & Schwartz 2001, p. 124) For alle endelige værdier af m og n gælder*

$$F(0, S_0) \geq \lim_{M \to \infty} U_0^{m,M}$$

Med hensyn til valget af antallet og typen af basisfunktionerne, viser flere undersøgelser, at det kun er nødvendigt med et lille antal funktioner, og at monomierne S, S^2, S^3 fremfor de mere komplicerede polynomiefamilier som f.eks. Legendre-polynomierne, giver tilstrækkelig god konvergens (Stentoft 2001, p. 15-16), (Longstaff & Schwartz 2001, p. 142-143). Moreno & Navas (2001) undersøger 10 forskellige typer polynomier og kommer til samme konklusion for en standard salgsoption, mens resultaterne for mere komplicerede derivater ikke er entydig. Morera (2003) undersøger *wavelet transforms* som alternativ til forskellige typer basisfunktioner og finder, at der med *wavelet transforms* kan opnås samme præcision med kortere beregningstid.

Clément et al. (2002, p. 458) giver resultater omkring hastigheden af konvergensen, og Stentoft (2001, p. 16) beskriver en implementeringsmetode, der giver mulighed for at udregne samme.

6.7 Mindste kvadraters metode

I implementationen af algoritmen er det kun nødvendigt at gemme værdien af vektoren U og rent programmeringsmæssigt bliver metoden noget simp-

lere. En vigtig del af implementationen er imidlertid algoritmen til mindste kvadraters metode, og denne vil derfor blive beskrevet her.

Problemet fra (6.4) er et problem med mindste kvadraters metode. Vektoren b defineres nu som de $U_j = Z^{(i)}_{\tau^{i,m,M}_{j+1}}$, hvor $Z^{(i)}_j > 0$, og tilsvarende vektoren x som de $S^{(i)}_j$, hvor $Z^{(i)}_j > 0$. Ydermere sættes $e_1(x) = x^0 = 1$, $e_2(x) = x^1 = x$ og $e_i(x) = x^{i-1}$, så vektoren $e^m(x) = (1, x, ..., x^{m-1})$. Designmatricen A defineres som en $M \times m$ matrix, hvor $A_{ij} = e_j(x_i)$. I dette tilfælde bliver designmatricen den såkaldte Vandermonde-matrix

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^{m-1} \end{pmatrix}$$

Med disse definitioner kan (6.4) formuleres som (Press et al. 1992, p. 665, 671)

$$\alpha = \arg\min_a |A \cdot a - b|^2$$

Dette problem løses effektivt ved hjælp af singulær værdi dekomposition (SVD), der kort fortalt går ud på, at en $M \times m$ matrix A, kan skrives som matrixproduktet $A = USV^T$, hvor S er en matrix med singulærværdierne i diagonalen (Press et al. 1992, p. 53). Den pseudoinverse af A kan skrives $A^{-1} = VS^{-1}U^T$. Ved hjælp af denne kan α findes som

$$\alpha = VS^{-1}U^T b$$

Idet S^{-1} er en diagonalmatrix findes den inverse meget enkelt ved at tage den reciprokke værdi $\frac{1}{S_{ii}}$ af alle diagonalelementerne. Dog skal man tage højde for, at nogle af disse kan være nul (dette sker når A er singulær). Ligeledes kan man risikere, at S_{ii} er meget tæt på nul, hvilket kan gøre metoden numerisk ustabil. Derfor bør alle værdier, hvis forhold med den største singulære værdi er mindre end m gange maskinens epsilon (det mindste tal processoren kan repræsentere) (Press et al. 1992, p. 672).

Efter at have fundet parametervektoren α, kan approksimationen til værdien med de givne funktioner findes som

$$\widehat{b} = A\alpha$$

Denne vektor indeholder præcis de værdier, der er den forventede fortsættelsesværdi for de enkelte stier, der bruges i (6.3).

$$\widehat{b}^{(i)} = \alpha_j^{(m,M)} \cdot e^m(S_j^{(i)})$$

Således kan disse værdier bruges til at finde den næste $U_j^{(i)}$

$$U_j^{(i)} = \max(Z_j^{(i)}, \widehat{b}^{(i)})$$

Kildekoden til mindste kvadraters metode kan findes i afsnit D.4.4.

6.8 Benyttelse på SVJD-sti

For at kunne benytte den ovenfor beskrevne algoritme på en sti genereret i SVJD-modellen er det nødvendigt at lade begge tilstandsvariabler indgå i regressionen. Springkomponenten derimod er ikke en tilstandsvariabel og skal ikke indgå. Designmatricen for et givet par af tilstandsvariable (s, v) på tidspunkt t, når der bruges monomier op til 2. grad, får udseendet

$$A = \begin{pmatrix} 1 & s_1 & s_1^2 & v_1 & v_1^2 \\ 1 & s_2 & s_2^2 & v_2 & v_2^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s_M & s_M^2 & v_M & v_M^2 \end{pmatrix}$$

Kildekoden til LSM algoritmen i SVJD-tilfældet, kan findes i afsnit D.5.2, og brugen demonstreres i C.7.

6.9 Kontrolvariabel-teknik

Ved brug af LSM-algoritmen har vi en oplagt kontrolvariabel i form af den europæiske pris for de samme model-parametre. Dette gælder både for BS-modellen og SVJD-modellen. Med notationen fra afsnit 5.6 sættes F_S lig den europæiske pris fundet med den analytiske formel, $\widehat{F}_S$ sættes lig den pris som den simulerede sti giver for en europæisk option og $\widehat{F}_K$ sættes lig den pris som LSM giver for den simulerede sti. Det forbedrede estimat for stien findes så med $(\widehat{F}_K - \widehat{F}_S) + F_S$ som tidligere forklaret. Hvor stor en variansreduktion dette giver undersøges nærmere i afsnit 7.5.

Kontrolvariabel-teknikken ville kunne forbedres yderligere ved at finde den optimale værdi θ^*, der derefter bruges i udtrykket $\widehat{F}_K + \theta^*(F_S - \widehat{F}_S)$ (Rasmussen 2002, p. 10), (Boyle et al. 1997, p. 1275). Andre forslag til at forbedre kontrolvariabel-teknikken ved brug af LSM-metoden kan findes i Rasmussen (2002) og Tian & Burrage (2003).

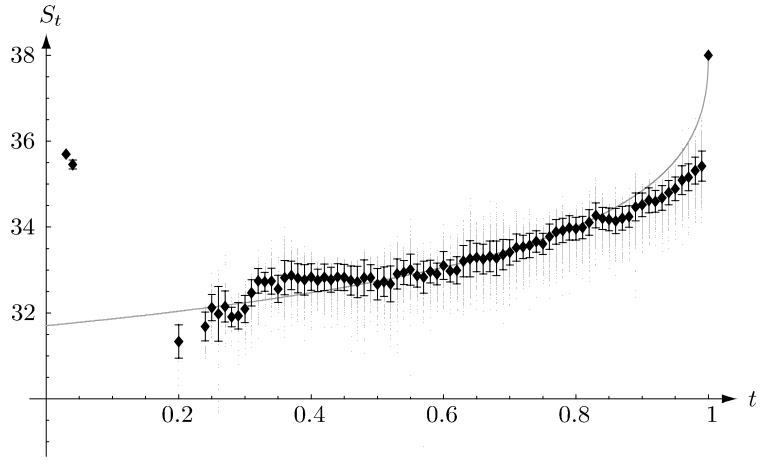
6.10 Reduktion af hukommelsesforbrug

En ulempe ved LSM er, at den har et ret stort hukommelsesforbrug, idet samtlige stier skal lagres i hukommelsen for at være klar til at bruge i regressionen. Dette problem opstår primært, fordi stierne simuleres fremad på trods af, at algoritmen arbejder baglæns. Således bliver det nødvendigt at lagre $N * m$ tal under hele proceduren. (Hvis også volatiliten indgår som tilstandsvariabel skal dette tal yderligere ganges med 2.) Dette problem kan delvist imødegås ved at simulere stierne baglæns, så det kun er nødvendigt at lagre N tal. For tilfældet, hvor aktivet følger en geometrisk brownsk bevægelse kan stierne simuleres baglæns ved hjælp af *brownian bridges*, som beskrevet i Stentoft (2002, p. 29). Tilsvarende teknikker ville kunne findes for SV-modellen, og det samme er muligvis tilfældet for SVJD-modellen, der inkluderer spring. Denne teknik er dog ikke implementeret i det medfølgende program.

En anden metode, der kan bruges til at reducere hukommelsesforbruget er at simulere stien med samme antal punkter men benytte et mindre antal punkter i Longstaff-Schwartz algoritmen. Dette betyder i praksis, at antallet af tidspunkter, hvor førtidig indfrielses tillades, reduceres. Den amerikanske option, der i princippet kan indfries på alle kontinuerte tidspunkter, bliver således i højere grad end tidligere tilnærmet med en bermuda option, der kun kan indfries på et antal diskrete tidspunkter. Udover en forbedring af hukommelsesforbruget reducerer denne metode også antallet af udregninger og algoritmen bliver således væsentligt hurtigere mod en reduktion af nøjagtigheden. Denne metode er implementeret i det medfølgende program for SVJD-tilfældet, da det her er nødvendigt med en finere tidsinddeling, idet simulationen af stien kun er en tilnærmelse. Derimod er det ikke nødvendigt i BS-tilfældet, da simulationen alligevel er nøjagtig som vist i formel (5.5).

6.11 Indfrielsesgrænsen for LSM

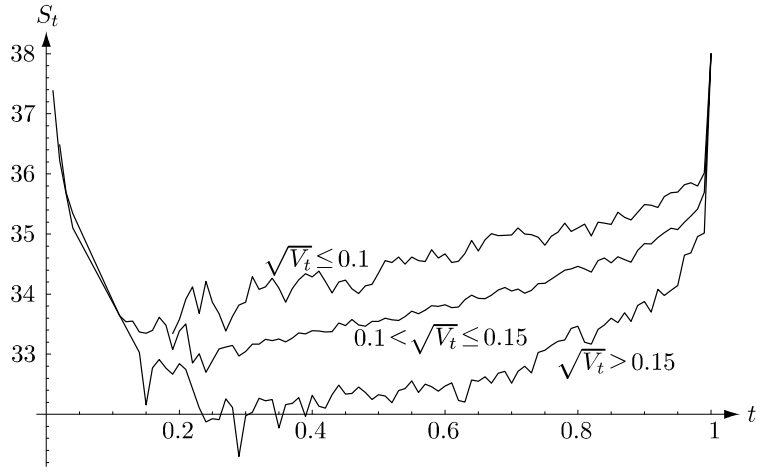
Hvis man ønsker en tilnærmelse til den indfrielsesgrænse, der ligger implicit i Longstaff-Schwartz metoden, kan man finde den ved at plotte punkterne $(\tau_j^{i,m,M}\Delta t, S^{(i)}_{\tau_j^{i,m,M}})$. Hvert af disse M punkter (t, S_t) vil repræsentere det sted på hver enkelt sti, hvor det ifølge LSM-algoritmen har været optimalt at indfri optionen. Denne beslutning er, som det er forklaret i afsnit 6.5 baseret på den information, der ligger i de øvrige simulerede stier. Da metoden er iboende tilnærmelsesvis, vil de fundne indfrielsespunkter ikke være optimale, men kun tilnærmelsesvist optimale. I figur 6.1 er vist, hvor indfrielsespunkterne ligger i forhold til den sande indfrielsesgrænse fundet med endelig differens metoden.



Figur 6.1: Indfrielsespunkter (lysegrå) fundet med LSM sammenlignet med indfrielsesgrænsen fundet med endelig differens (mørkegrå). De sorte fejlbarer angiver gennemsnittet med en standardafvigelse på hver side for indfrielsespunkterne for hver tidstrin. De brugte parametre til BS-modellen er $K = 38$, $T - t = 1$, $S_0 = 40$, $r^f = 0.08$, $r^d = 0.06$, $\sigma = 0.15$. Der er brugt 100000 stier.

I SVJD-modellen er både S_t og V_t tilstandsvariable og indfrielsesgrænsen er således 3-dimensionel og bliver til en flade. For SV-tilfældet er dette illustreret i Tzavalis & Want (2003, p. 42). Hvis der indgår flere tilstandsvariable i form af flere underliggende aktiver, bliver indfrielsesgrænsen 3- eller fler-dimensionel.

Ved brug af LSM-metoden er det muligt at komme frem til en tilnærmelse til den 3-dimensionelle indfrielsesgrænse ved at gruppere de fundne indfrielsespunkter (t, S_t, V_t) efter volatiliteten og derved lave en slags niveaukurver. Dette er illustreret i figur 6.2. Det ses, at jo lavere volatiliten er, jo højere skal kursen være, for at det kan betale sig at indfri optionen. På samme måde som for den to-dimensionelle indfrielsesgrænse ses det, at der er indfrielsespunkter tæt ved $t = 0$, der ligger urealistisk højt.



Figur 6.2: Niveaukurver for indfrielsesgrænsen konstrueret ved at gruppere volatiliteten $\sqrt{V_t}$ i intervallerne $[0; 0.1]$, $]0.1; 0.15]$ og $]0.15; \infty[$. Kurverne er baseret på indfrielsespunkter fra simulation af 2*10000 stier.

7 Numeriske resultater

I dette kapitel vil de resultater, som det udarbejdede program udregner, blive præsenteret. Først vil det blive afprøvet om programmet regner rigtigt ved at sammenligne de givne Monte Carlo resultater med de analytiske løsninger. Samtidigt vil det blive præsenteret, hvor store forbedringer i resultaterne, som brugen af antitetiske stier og kontrolvariabelteknikken giver.

De Mathematica-dokumenter, der er benyttet til udregninger, kan findes på den medfølgende CD-ROM (se bilag B). Dette gør det simpelt at gentage testene eller lave flere testkørsler med andre parameterværdier. Da der ved en ny kørsel af testene imidlertid vil blive valgt nye *seeds* til Monte Carlo algoritmen vil resultaterne ikke være de samme som de viste resultater (men forhåbentligt konsistente).

7.1 Monte Carlo Black-Scholes

For at kunne vurdere de beregnede værdier, der findes med de metoder, der er udviklet i denne afhandling, er det formålstjenligt at kunne sammenligne med en kendt, velafprøvet metode. Derfor sammenlignes Monte Carlo metoden i dette underafsnit derfor med den analytiske løsning for Black-Scholes modellen.

Som parametre er valgt de Black-Scholes parametre, der svarer til parametrene for mbas i 7.1. Volatiliteten er valgt til $\sigma = 0.15$, selvom den Black-Scholes volatilitet, der svarer til SVJD-modellen strengt taget er 0.150001. Den analytiske pris udregnet med (2.14) bliver 0.37636. Denne værdi antages

Model	S_t	r^d	r^f	$\sqrt{V_t}$	σ_v	$\sqrt{\frac{\alpha}{\beta^*}}$	β^*	$\frac{\ln 2}{\beta^*}$	ρ	λ	$\overline{k}^*$	δ
mbas	40	0.08	0.06	0.15	0.15	0.15	4	0.1733	0	0	0	0
mVt	40	0.08	0.06	**0.2**	0.15	0.15	4	0.1733	0	0	0	0
msigmav	40	0.08	0.06	0.15	**0.3**	0.15	4	0.1733	0	0	0	0
mrho	40	0.08	0.06	0.15	0.15	0.15	4	0.1733	**0.1**	0	0	0
mjump	40	0.08	0.06	**0.1118**	**0.2**	**0.1118**	4	0.1733	0	**2**	0	**0.07**

Tabel 7.1: Modelparametre brugt i beregningstestene. Modellerne er opstillet som variationer af en basismodel. Værdierne er de samme som i Bates (1996, p. 80).

benchmarkpris	den pris, der betragtes som den sande pris
MC pris	en enkelt pris udregnet med Monte Carlo simulation
std.fejl	den estimerede standardfejl for MC prisen
bias	den gennemsnitlige afvigelse fra benchmarkprisen, baseret på 100 simulationer
std.afv.	standardafvigelsen på de 100 simulationer, samtidigt et estimat for standardfejlen
beregn.tid	beregningstid i sekunder

Tabel 7.2: Beskrivelse af betegnelser brugt i tabellerne med beregningstest for Monte Carlo simulationer.

(uden de store skrupler) som værende korrekt og kaldes benchmarkprisen.

De priser, der udregnes med Monte Carlo metoden, sammenlignes nu med den analytiske pris på følgende måde. Først udregnes en pris med tilhørende standardfejl vha. Monte Carlo for at give en indikation af hvilket resultat man får ved én almindelig udregning med det givne antal stier. Dernæst foretages der 100 nye beregninger og den gennemsnitlige afvigelse fra benchmarkprisen beregnes $\frac{1}{100}\sum_{i=1}^{100}(\widehat{F}_i - F)$. Herved er det muligt at se om Monte Carlo metoden regner systematisk forkert. Samtidigt udregnes standardafvigelsen for de beregnede priser, hvilket giver et estimat for standardfejlen (Boyle et al. 1997, p. 1278). Desuden angives beregningstiden i sekunder for en enkelt pris på en Pentium III 1 GHz PC med 522 MB RAM. En oversigt over de udregnede tal ses i tabel 7.2.

I tabel 7.3 vises udregnede priser for forskellige antal stier med og uden brug af antitetiske stier. Da det ikke forbedrer præcisionen at bruge flere tidstrin, er der brugt ét tidstrin. Som forventet viser bias-tallene, at der ikke er en systematisk afvigelse fra den analytiske pris.

Det ses at brugen af antitetiske stier ikke giver en væsentlig anderledes bias, men at variansen og dermed standardfejlen som forventet reduceres. Da brugen af antitetiske stier kun forøger beregningstiden minimalt er det således oplagt at bruge disse. På trods af, at der brugt antetiske stier, er standardfejlen imidlertid stadig estimeret til 0.00064 ved brug af 1000000 stier. Det kan derfor konstateres, at der for det valgte eksempel skal bruges mere end en million stier for at opnå præcision ud på 4. decimal. Da standardfejlen afhænger af antallet af stier med faktoren $\frac{1}{\sqrt{M}}$ vil det ikke hjælpe væsentligt at tilføje flere stier. Dette illustrerer den ulempe ved Monte Carlo metoden, som må accepteres for at få metodens store fleksibilitet, nemlig at beregningstiden firdobles, hvis præcisionen ønskes fordoblet.

	stier	1000	10000	50000	100000	1000000
	MC pris	0.41265	0.38145	0.38011	0.37708	0.37668
	std.fejl	0.03090	0.00917	0.00412	0.00290	0.00092
rå	bias	0.00301	-0.00011	0.00034	-0.00009	0.00009
	std.afv.	0.02780	0.01014	0.00398	0.00329	0.00096
	beregn.tid	0.00	0.03	0.18	0.36	3.55
	MC pris	0.41913	0.37581	0.37590	0.37877	0.37563
	std.fejl	0.01958	0.00588	0.00265	0.00187	0.00059
antitetisk	bias	0.00150	-0.00032	-0.00033	-0.00011	0.00005
	std.afv.	0.01885	0.00630	0.00251	0.00187	0.00064
	beregn.tid	0.00	0.04	0.19	0.39	3.81

Tabel 7.3: Sammenligning af Monte Carlo optionspriser med den analytiske pris i BS-modellen. Benchmarkprisen er 0.37636. Priserne er for en salgsoption med K=38 og T-t=0.25. σ=0.15 og øvrige parametre som mbas i tabel 7.1. Der er brugt 1 tidstrin, og det indikerede antal stier.

7.2 Monte Carlo SVJD

I lighed med prisen for en europæisk option under Black-Scholes modellen kan de udregnede priser i SVJD-modellen sammenlignes med en analytisk formel, idet denne er implementeret i det medfølgende program. Til beregningstesten bruges parametrene for mbas i tabel 7.1. Benchmarkværdien findes til 0.37443 med den semi-analytiske formel (4.5). I tabel 7.4 er effekten af antallet af stier på simulationens præcision vist. Idet nøjagtigheden af simulationen af stien afhænger af antallet af tidsinddelinger, er der brugt 100 tidstrin. Ved sammenligning med tabel 7.3 ses det, at der ikke er væsentligt højere bias, og at standardfejlen ikke er højere for SVJD-modellen end for BS-modellen. Dette påviser, at simulationen af SVJD-stien er korrekt for de valgte parameterværdier.

Forøgelsen i beregningstid ved brug af antitetiske stier er lidt større i SVJD-tilfældet (16% mod 7%). Dette skyldes primært, at der bruges flere tidstrin. Dog er fordelen ved brug af den variansreducerende teknik stadig åbenlys.

Da det naturligvis ikke er tilstrækkeligt at afprøve korrektheden af udregningerne for ét valg af parameterværdier, udregnes nu tal for de forskellige sæt af parameterværdier, der er listet i tabel 7.1 og aftalekursen K varieres også. Til udregningen bruges der 10000 stier, da det giver en fornuftig afvejning af præcision og beregningstid. Som benchmark bruges igen den semi-analytiske formel for SVJD-modellen. Idet de brugte parameter er de samme som i Bates (1996, p. 80), er det samtidigt muligt at sikre, at implementationen af den analytiske formel er korrekt. Det viser sig, at priserne er præcise op til de 3 decimaler, der er vist i Bates-artiklen.

Resultaterne er vist i tabel 7.5. For parametervalgene mbas og msigmav er der ingen systematisk bias og standardfejlene er af samme størrelseorden

		1000	10000	50000	100000
	MC pris	0.34824	0.36501	0.37272	0.37613
	std.fejl	0.02772	0.00909	0.00412	0.00292
rå	bias	-0.00321	-0.00107	0.00007	-0.00059
	std.afv.	0.02319	0.00948	0.00405	0.00245
	beregn.tid	0.15	1.55	7.73	15.27
	MC pris	0.38000	0.36618	0.37611	0.37503
	std.fejl	0.01941	0.00587	0.00269	0.00189
antitetisk	bias	-0.00353	-0.00102	0.00051	-0.00030
	std.afv.	0.01844	0.00577	0.00247	0.00191
	beregn.tid	0.17	1.87	8.89	17.69

Tabel 7.4: Sammenligning af Monte Carlo optionspriser med den analytiske pris i SVJD-modellen. Benchmarkprisen er 0.37443. Priserne er for en salgsoption med K=38 og T-t=0.25 og øvrige parametre som mbas i tabel 7.1. Der er brugt 100 tidstrin, og det indikerede antal stier.

som det observeredes for BS-modellen. Her skal det bemærkes, at jo højere optionspris, jo større standardfejl og bias kan der tillades, idet det primært er den relative afvigelse fra den sande pris, der er relevant. For mVt er alle bias-tallene negative. Dette kan være tilfældigt, men kan også skyldes en systematisk afvigelse.

For mrho er bias-tallene høje og varierer samtidigt med aftalekursen K. Standardfejlene er imidlertid ikke højere end for andre parametervalg. Dette tyder på en systematisk fejlagtigt prisfastsættelse, når $\rho \neq 0$. Dette kan forklares med den tidligere observation fra afsnit 5.7.2 om, at fordelingen for de simulerede Monte Carlo stier bliver mere skæv, end den skal være ifølge den analytiske fordeling, når der indgår korrelation i modellen.

For mjump er standardafvigelsen for de enkelte udregnede priser meget høj, og samtidigt er bias meget høj. Det ses også, at enkeltpriserne (MC pris) er meget unøjagtige. Noget af upræcisionen skyldes, at der kun er brugt 100 tidsinddelinger, da det nøjagtige tidspunkt for springets indtræffen er vigtigt. Simulationen ville derfor med fordel forbedres med metoden beskrevet i afsnit 5.7.2, p. 5.7.2.

For Monte Carlo simulationen i SVJD-modellen afhænger præcisionen ikke kun af antallet af stier, men også af antallet af tidsinddelinger. For at studere effekten af forskellige kombinationer af disse, er der i tabel 7.6 listet resultater af dette. Da hverken bias eller standardfejlen bliver væsentligt mindre ved en forøgelse af antallet af tidsintervaller, konkluderes det, at 100 tidsintervaller og 10000 stier er en passende afvejning mellem præcision og beregningstid. Denne konklusion er dog kun gældende for de valgte parametre, og vil muligvis være forskellig for andre parametervalg.

Model		$K = 38$	$K = 39$	$K = 40$	$K = 41$	$K = 42$
mbas	benchmark	0.37443	0.66167	1.07404	1.61749	2.28323
	MC pris	0.38151	0.66104	1.08552	1.63303	2.27907
	std.fejl	0.00605	0.00741	0.00829	0.00786	0.00658
	bias	-0.00100	-0.00138	0.00085	0.00069	-0.00013
	std.afv.	0.00576	0.00835	0.00842	0.00710	0.00622
	beregn.tid	1.76	1.77	1.78	1.74	1.73
mVt	benchmark	0.57502	0.90202	1.33437	1.87396	2.51471
	MC pris	0.55915	0.89456	1.32645	1.86195	2.51379
	std.fejl	0.00781	0.00926	0.00988	0.00942	0.00825
	bias	-0.00085	-0.00021	-0.00390	-0.00139	-0.00192
	std.afv.	0.00789	0.00838	0.00973	0.00860	0.00789
	beregn.tid	1.74	1.74	1.78	1.77	1.77
msigmav	benchmark	0.36930	0.64847	1.05640	1.60147	2.27365
	MC pris	0.37445	0.64152	1.05024	1.58733	2.26292
	std.fejl	0.00621	0.00765	0.00850	0.00830	0.00761
	bias	-0.00075	0.00033	-0.00028	-0.00143	-0.00100
	std.afv.	0.00603	0.00672	0.00914	0.00761	0.00884
	beregn.tid	1.77	1.77	1.77	1.75	1.73
mrho	benchmark	0.36876	0.65803	1.07372	1.62065	2.28897
	MC pris	0.36278	0.66940	1.06756	1.61578	2.28534
	std.fejl	0.00593	0.00753	0.00823	0.00775	0.00673
	bias	0.00447	0.00428	-0.00027	-0.00217	-0.00588
	std.afv.	0.00590	0.00791	0.00848	0.00658	0.00560
	beregn.tid	1.88	1.86	1.87	1.83	1.83
mjump	benchmark	0.35647	0.61937	1.01807	1.56650	2.25182
	MC pris	0.10246	1.03762	0.64812	1.48987	3.56046
	std.fejl	0.00297	0.00880	0.00493	0.00454	0.00731
	bias	0.05801	0.02065	0.08120	0.04034	-0.02672
	std.afv.	0.47325	0.57148	0.61139	0.57233	0.43976
	beregn.tid	1.86	1.83	1.81	1.82	1.83

Tabel 7.5: Sammenligning af analytisk pris med Monte Carlo i SVJD-modellen. K er varieret som vist. T-t=0.25. De øvrige parametre er som vist i tabel 7.1. Der er brugt 10000 stier samt de antitetiske og 100 tidsinddelinger.

stier	tidstrin		rå	antitetisk
10000	50	MC	0.38358	0.38161
		std.fejl	0.00934	0.00602
		bias	-0.00041	0.00018
		std.afv.	0.00868	0.00678
		beregn.tid	0.80	0.94
10000	100	MC	0.36225	0.37182
		std.fejl	0.00890	0.00592
		bias	0.00060	-0.00092
		std.afv.	0.00837	0.00561
		beregn.tid	1.53	1.77
10000	200	MC	0.39019	0.38630
		std.fejl	0.00952	0.00613
		bias	0.00125	-0.00022
		std.afv.	0.00978	0.00704
		beregn.tid	3.07	3.65
10000	300	MC	0.37512	0.37874
		std.fejl	0.00933	0.00597
		bias	0.00090	-0.00156
		std.afv.	0.00902	0.00557
		beregn.tid	4.60	5.47
50000	50	MC	0.37089	0.37287
		std.fejl	0.00413	0.00266
		bias	0.00017	-0.00025
		std.afv.	0.00396	0.00252
		beregn.tid	3.98	4.55
50000	100	MC	0.36826	0.37487
		std.fejl	0.00411	0.00267
		bias	-0.00063	-0.00030
		std.afv.	0.00406	0.00270
		beregn.tid	7.70	8.75
50000	200	MC	0.37606	0.37646
		std.fejl	0.00416	0.00270
		bias	0.00058	-0.00011
		std.afv.	0.00444	0.00269
		beregn.tid	15.45	17.41

Tabel 7.6: Effekt af antallet af stier og tidstrin for Monte Carlo i SVJD-modellen.

kursinddelinger	tidsinddelinger	pris	beregningstid
100	1000	0.381023	0.03
200	4000	0.381118	0.20
400	16000	0.381132	1.58
800	64000	0.381141	12.58
1000	100000	0.381143	24.69

Tabel 7.7: Konvergens for endelig differensmetoden.

7.3 Endelig differens Black-Scholes

For at teste om implementationen af least-squares Monte Carlo regner rigtigt, kan vi teste resultaterne op mod resultaterne fra en endelig differensmetode. For først at sikre, at den pris, vi finder med endelig differens metoden er præcis nok, undersøger vi, hvor mange tidsinddelinger, der er nødvendige før metoden konvergerer tilstrækkeligt. Den brugte implementation er taget fra QuantLib, der både skifter variabel ved at tage logaritmen af kursen og bruger prisen på den tilsvarende europæiske option som kontrolvariabel. Til udregningerne, der ses i tabel 7.7, er antal tidsinddelinger og kursinddelinger valgt, så de stemmer overens med anbefalingen i Hull (2000b, p. 422) om at $\Delta Z = \sigma\sqrt{3\Delta t}$, hvor $Z = \ln S$. Med udgangspunkt i tabellen konkluderes det, at værdien konvergerer mod en værdi, der ikke kan ligge langt fra 0.381143.

7.4 LSM Black-Scholes

Ved at tage udgangspunkt i prisen udregnet med endelig differens metoden, kan det kontrolleres om least-squares Monte Carlo metoden regner rigtigt. Det er samtidigt interessant at undersøge, hvilken kombination af antallet af stier og antallet af tidstrin, der er det bedste valg for at få en god præcision. For LSM-algoritmen er det udover antitetiske stier muligt at bruge en kontrolvariabel til variansreduktion. Effekten af disse er også interessant at studere. I tabel 7.8 er derfor lavet udregninger, der illustrerer disse effekter.

Idet brugen af antitetiske stier fordobler antallet af stier, der lagres i hukommelsen, og ligeledes antallet, der bruges i regressionen, er brugen af disse langt dyrere i computerkraft, end det var tilfældet ved standard Monte Carlo. Dette fremgår også tydeligt af de rapporterede beregningstid – tilfældet 10000 stier og 100 tidsintervaller giver eksempelvis en forøgelse i tidsforbrug på 75%. Dette bør naturligvis indgå i overvejelserne, når man beslutter, om man vil benytte denne teknik eller ej. Idet benyttelsen af antitetiske stier giver en mindre standardfejl end benyttelsen af dobbelt så mange rå stier, er det imidlertid rimeligt at sammenligne f.eks. 5000 stier plus antitetiske stier med 10000 rå stier. Disse to kombinationer vil have det samme hukommelsesforbrug og samme antal beregninger i regressionen, men benyttelsen

af antitetiske stier vil have en variansreducerende effekt og således lavere standardfejl. Desuden kan det ses i tabellen, at beregningstiden er en anelse lavere (3.02 mod 3.51 sek.). Det kan således konkluderes, at det stadig er en fordel at benytte de antitetiske stier.

Ligeledes ses det, at brugen af kontrolvariablen kun giver en minimal forøgelse i udførelsestiden, men en væsentligt forbedring af standardfejlen.

7.5 LSM SVJD

De netop præsenterede beregningstests for henholdsvis Monte Carlo metoden i SVJD-modellen og least-squares Monte Carlo metoden i BS-modellen, giver en god forventning om, at kombinationen af disse, nemlig least-squares Monte Carlo i SVJD-modellen, også giver korrekte resultater. Alligevel er det altid et godt princip at have to væsenforskellige implementationer af den samme teoretiske prisfastsættelsesmodel, så det er muligt at teste disse op mod hinanden. Dette er ikke gjort i denne afhandling, men da der i Bates (1996) er givet resultater for en implementering af SVJD-modellen med endelig differens metoden, kan der sammenlignes med disse. Dette er gjort i tabel 7.9. Det kan konkluderes, at der ikke er væsentlige afvigelser for nogen af de brugte parameterværdier og aftalekursen. Interessant nok observeres der ikke højere biastal for mrho og mjump end for de andre parameter, som det ellers var tilfældet for standard Monte Carlo.

Beregningstiderne for LSM SVJD er væsentligt længere end for LSM BS. Dette ses ved at sammenligne tiden 17.83 sek. for mbas, $K = 40$ fra tabel 7.9 med tiden 6.16 sek. ved 10000 stier og 100 tidsintervaller fra tabel 7.8. Det større tidsforbrug skyldes, at både S_t og V_t indgår i regressionen. Det observeres desuden, at beregningstiden bliver længere, når K bliver højere. Da der regnes på en salgsoption, vil en højere aftalekurs medføre, at der er flere stier, der er *in-the-money* og således indgår i regression. Dette medfører et større regnearbejde og dermed længere udførelsestid.

stier	tidstrin		rå	antitetisk	kontrolvar.	anti.+kont.
		MC	0.37595	0.37638	0.38363	0.38256
		std.fejl	0.01161	0.00713	0.00638	0.00445
5000	50	bias	0.00375	0.00219	0.00487	0.00224
		std.afv.	0.01203	0.00814	0.00597	0.00408
		beregn.tid	0.92	1.59	0.95	1.67
		MC	0.38071	0.38325	0.37497	0.38646
		std.fejl	0.01158	0.00720	0.00707	0.00476
5000	100	bias	0.00360	0.00242	0.00491	0.00141
		std.afv.	0.01385	0.00717	0.00661	0.00439
		beregn.tid	1.86	3.02	1.73	3.09
		MC	0.37088	0.40024	0.38747	0.37755
		std.fejl	0.01135	0.00733	0.00684	0.00462
5000	200	bias	0.00491	0.00086	0.00582	0.00134
		std.afv.	0.01190	0.00791	0.00660	0.00433
		beregn.tid	3.48	6.20	3.40	6.27
		MC	0.38067	0.37571	0.38964	0.38100
		std.fejl	0.01195	0.00668	0.00684	0.00480
5000	300	bias	0.00558	0.00183	0.00472	0.00248
		std.afv.	0.01307	0.00774	0.00693	0.00472
		beregn.tid	5.32	9.70	5.14	9.30
		MC	0.37441	0.38390	0.39096	0.37849
		std.fejl	0.00818	0.00515	0.00427	0.00329
10000	50	bias	0.00085	0.00034	0.00244	0.00045
		std.afv.	0.00886	0.00545	0.00461	0.00313
		beregn.tid	1.87	3.08	1.77	3.25
		MC	0.37562	0.38930	0.38567	0.37694
		std.fejl	0.00761	0.00541	0.00435	0.00326
10000	100	bias	0.00112	0.00035	0.00188	0.00030
		std.afv.	0.00804	0.00581	0.00443	0.00293
		beregn.tid	3.51	6.13	3.41	6.16
		MC	0.38280	0.38149	0.38237	0.38008
		std.fejl	0.00830	0.00511	0.00435	0.00329
10000	200	bias	0.00215	0.00044	0.00239	0.00015
		std.afv.	0.00895	0.00576	0.00429	0.00300
		beregn.tid	7.06	14.03	6.81	14.42
		MC	0.38151	0.38176	0.38034	0.37843
100000	50	std.fejl	0.00257	0.00161	0.00139	0.00098
		beregn.tid	18.77	35.34	19.08	34.39
		MC	0.37471	0.37888	0.38126	0.38126
100000	100	std.fejl	0.00249	0.00160	0.00141	0.00100
		beregn.tid	36.86	68.88	36.77	69.05

Tabel 7.8: Effekt af antallet af stier og tidstrin for least-squares Monte Carlo i BS-modellen. Samme parametre som i tabel 7.3.

Model		$K = 38$	$K = 39$	$K = 40$	$K = 41$	$K = 42$
mbas	benchmark	0.379	0.672	1.095	1.653	2.343
	MC pris	0.38076	0.66852	1.08837	1.64770	2.32595
	std.fejl	0.00315	0.00435	0.00644	0.00810	0.01042
	bias	0.00136	-0.00077	-0.00285	-0.00330	-0.00833
	std.afv.	0.00283	0.00452	0.00633	0.00840	0.01055
	beregn.tid	4.78	8.13	17.83	25.86	30.48
mVt	benchmark	0.582	0.916	1.358	1.911	2.571
	MC pris	0.59177	0.91631	1.35577	1.91142	2.56422
	std.fejl	0.00447	0.00596	0.00766	0.00979	0.01169
	bias	0.00116	-0.00200	-0.00350	-0.00670	-0.00694
	std.afv.	0.00433	0.00634	0.00772	0.00927	0.01062
	beregn.tid	5.89	10.80	18.33	25.53	29.50
msigmav	benchmark	0.374	0.658	1.077	1.638	2.335
	MC pris	0.37674	0.65835	1.06791	1.63149	2.31426
	std.fejl	0.00261	0.00426	0.00601	0.00749	0.00991
	bias	0.00113	-0.00046	-0.00468	-0.00955	-0.01181
	std.afv.	0.00321	0.00435	0.00628	0.00809	0.00917
	beregn.tid	4.98	8.03	18.25	27.66	30.89
mrho	benchmark	0.373	0.668	1.094	1.655	2.347
	MC pris	0.37903	0.67357	1.09307	1.65650	2.33654
	std.fejl	0.00336	0.00442	0.00625	0.00822	0.00986
	bias	0.00231	0.00034	-0.00249	-0.00398	-0.00478
	std.afv.	0.00293	0.00370	0.00565	0.00859	0.01107
	beregn.tid	5.08	8.30	18.02	25.94	30.78
mjump	benchmark	0.360	0.626	1.030	1.588	2.292
	MC pris	0.35972	0.62200	1.03883	1.58841	2.31650
	std.fejl	0.00277	0.00408	0.00480	0.00739	0.00950
	bias	0.00253	0.00157	0.00014	-0.00161	-0.00515
	std.afv.	0.00391	0.00505	0.00666	0.00926	0.01225
	beregn.tid	5.20	8.22	17.88	27.27	32.06

Tabel 7.9: Sammenligning af least-squares Monte Carlo i SVJD-modellen med pris udregnet med endelig differens i Bates (1996). K er varieret som vist. T-t=0.25. De øvrige parametre er som vist i tabel 7.1. Der er brugt 10000 stier samt de antitetiske og 100 tidsinddelinger. Den europæiske pris er brugt som kontrolvariabel.

8 Konklusion

Formålet med denne afhandling har været at behandle teorien bag SVJD-modellen og least-squares Monte Carlo metoden og udvikle et program, der kombinerer modellen med beregningsmetoden. Herved kan der prisfastsættes amerikanske optioner i SVJD-modellen.

Kapitel 2 beskrev det fremherskende Black-Scholes paradigme for prisfastsættelse af optioner og grundlæggende begreber såsom det risikoneutrale sandsynlighedsmål blev forklaret. Det analytiske formel for europæiske optioner i BS-modellen blev præsenteret, og det blev godtgjort, hvordan denne kan bruges både for valuta og for en aktie, der udbetaler en kontinuert dividende.

I kapitel 3 blev der argumenteret for, at SVJD-modellen er en velegnet model i forhold til mange andre konkurerende modeller. Det skyldes ikke mindst, at der er godt empirisk belæg for, at en beskrivende model skal indeholde både elementer af stokastisk volatilitet og spring. Det blev blev desuden vist, at mange af de mest benyttede og beskrevne modeller er specialtilfælde af Bates' SVJD-model, og at den også derfor er interessant at studere. Det blev desuden konstateret, at flere forsøg på yderligere udvidelser af modellen ikke har været særligt frugtbare. Der blev dog også påpeget svagheder ved modellen, primært sværheden ved at estimere de nødvendige parametre.

I kapitel 4 blev SVJD-modellen præsenteret. Da SVJD-modellen implicerer et ikke-komplet marked var det nødvendigt at indføre visse antagelser om investorernes nyttefunktioner for at nå frem til en model under det risikoneutrale sandsynlighedsmål. Med udgangspunkt i modellen under det risikoneutrale sandsynlighedsmål blev der fundet frem til en semi-lukket formel for optionsprisen. Denne krævede kun numerisk beregning af et enkelt integrale, og Gauss-Kronrod algoritmen blev derfor introduceret som en velegnet metode til denne type integration. Herefter blev de mest gængse metoder til parameterestimation omtalt.

Der blev udviklet en teknik til at finde et kvantitativt udtryk for varians, skævhed og kurtosis via den karakteristiske funktion, og dette gjorde det muligt at finde den BS-model, der lå tættest på en givet SVJD-model. Tæthedsfunktionen gav mulighed for at illustrere effekterne af forskellige

parametervalg i SVJD-modellen grafisk. De vigtigste effekter var følgende: Højerere volatilitet af volatiliteten gav højere kurtosis. Positiv korrelation mellem kursændring og volatilitetsændring gav positiv skævhed. Tilsvarende for negativ korrelation. Spring gav skævhed i samme retning som springenes middelværdi og desuden en højere kurtosis.

I forlængelse heraf blev prisningseffekterne af parametervalgene behandlet og de tilhørende volatilitetssmil illustreret. Kurtosis gav et symmetrisk volatilitetssmil, mens skævhed gav et skævt grin.

I kapitel 5 blev Monte Carlo metoden for europæiske optioner beskrevet. Metodens store fleksibilitet og lette mulighed for anvendelse på forskellige modeller, herunder SVJD-modellen, samt den enkle implementering blev fremhævet som begrundelse for valget af denne metode fremfor andre metoder. Det blev desuden vist, hvordan den samtidige simulering af kurs og volatilitet i SVJD-modellen kan udføres, herunder hvordan den antitetiske sti simuleres.

I kapitel 6 præsenteredes teorien for prisfastsættelse af amerikanske optioner og de centrale begreber, herunder indfrielsesgrænsen blev defineret. Det blev samtidigt vist, at indfrielsesgrænsen ikke nødvendigvis konvergerer mod aftalekursen, når tiden nærmer sig udløbstidspunktet. Det blev konkluderet, at standard Monte Carlo ikke er brugbar til amerikanske optioner, og at andre kendte metoder kun vanskeligt lader sig anvende på SVJD-modellen.

Least-squares Monte Carlo metoden blev herefter beskrevet. Det blev vist, hvordan informationen på tværs af de simulerede stier kan bruges til at estimere fortsættelsesværdien for en amerikansk option. Det skete vha. af mindste kvadraters metode. Ved at arbejde baglæns fra udløbstidspunktet kunne værdien på tidspunkt 0 findes. Det blev vist, hvordan problemet med mindste kvadraters metoden kan formuleres på matrixform, og hvordan problemet kan løses med singulær værdi dekomposition.

Det blev herefter forklaret, at regression i SVJD-tilfældet skal afhænge af både kursen og volatiliteten. Det blev desuden vist, hvordan den europæiske pris effektivt kan bruges som kontrolvariabel i LSM-algoritmen og hermed reducere variansen.

Det blev desuden forklaret, hvordan hukommelsesforbruget i BS-tilfældet kan reduceres ved brug af Brownian bridges. For SVJD-tilfældet kunne hukommelsesforbruget og beregningstiden nedsættes ved kun at bruge et mindre antal punkter på den simulerede sti som mulige indfrielsespunkter.

Herefter blev den indfrielsesgrænse, der er implicit givet ved LSM algoritmen, præsenteret og illustreret grafisk.

I kapitel 7 blev der præsenteret en række beregningstest, der påviste, at de implementerede algoritmer regner korrekt i de fleste tilfælde. Dog vist der sig små upræcisioner, når korrelationen i SV-modellen ikke var 0. Det viste

sig også, at der i en model med spring skal bruges mange tidstrin og dermed en lang beregningstid, hvis der skal opnås præcise resultater.

Alle de præsenterede algoritmer er implementeret i et program, der er vist i bilag D og leveret på den til afhandlingen hørende CD-ROM (se bilag B). Beregningsrutiner er stillet til rådighed gennem Mathematica, hvilket er beskrevet i detaljer i brugervejledningen bilag C. Herved er der således produceret et program, der kan anvendes af andre studerende eller forskere til yderligere at belyse modellerne og metodernes egenskab. De implementerede rutiner indgår i QuantLib biblioteket, der distribueres som open source, og giver således en meget stor tilgængelighed. Dette kombineret med Mathematica brugergrænsefladen skulle give gode muligheder for anvendelser af andre.

A Notationsoversigt

A.1 Små bogstaver

e_k	basisfunktion til brug i regressionen i LSM-algoritmen
$f(\omega)$	pris for en Monte Carlo sti
$\overline{k}$	gennemsnitlig springstørrelse i springmodeller
$\overline{k}^*$	risikojusteret gennemsnitlig springstørrelse
t	tidspunkt
r	risikofri rente
r^d	indenlandsk risikofri rente
r^f	udenlandsk risikofri rente
s	observeret værdi af kursen S_t

A.2 Store bogstaver

B_t	proces for risikofrit aktiv
B_t^d	proces for indenlandsk risikofrit aktiv
B_t^f	proces for udenlandsk risikofrit aktiv
$\widetilde{B}_t^f$	proces for det udenlandske risikofri aktiv udtrykt i indenlandsk valuta
D_t	dividendeproces for aktie
$\mathbb{E}$	forventet værdi
$\mathbb{E}_{\mathbb{Q}}$	forventet værdi under sandsynlighedsmålet $\mathbb{Q}$
$\widehat{F}_K$	pris på kompleks instrument tilnærmet med Monte Carlo simulation
$\widehat{F}_S$	pris på simpelt instrument tilnærmet med Monte Carlo simulation
F	prisen på en *forward*
F	pris udregnet med Monte Carlo simulation; funktion for pris på instrument
F_1, F_2	moment-genererende funktioner for sandsynlighederne P_1 og P_2

F_m	observeret markedsværdi
$\mathcal{F}_t$	filtrering
G_n	integralet fundet med gaussisk kvadratur ved brug af n punkter
G_t	gain proces for aktie
K	aftalekurs (exercise price)
K_{2n+1}	integralet fundet med Gauss-Kronrod integration ved brug af $2n+1$ punkter
$L_n(x)$	det n'te Legendre-polynomium
M_t	martingale proces
$M(u)$	momentgenererende funktion
M	antal stier i Monte Carlo simulationen
$N(\mu, \sigma)$	normalfordelingen med middelværdi μ og standardafvigelse σ
P_1, P_2	sandsynligheder, der indgår i prisfastsættelsen af optioner i SVJD-modellen
$\mathbb{P}$	det objektive sandsynlighedsmål
$P_j^m(z)$	projektion af z på rummet udspændt af basisvektorerne, som brugt til regressionen i LSM-algoritmen
$\mathbb{Q}$	det risikoneutrale sandsynlighedsmål
$\mathbb{Q}(z)$	tæthedsfunktionen i punktet z for sandsynlighedsmålet $\mathbb{Q}$
$\mathbb{Q}(\omega)$	sandsynligheden for at stien ω følges
$\mathbb{Q}(A)$	sandsynligheden for at udtrykket A er sandt
$R(u)$	kumulant-genererende funktion
S_t	kursen på det underliggende aktiv på tidspunkt t
$\overline{S}_t$	indfrielsesgrænsen for en amerikansk option
$\widehat{S}_k$	kursen S i tidsinterval k simuleret med Monte Carlo simulation
T	udløbstid for option
$\mathcal{X}$	proces for fordring
X_t	vilkårlig stokastisk proces
V_t	proces for variansen i modeller med stokastisk volatilitet
V^h	værdiproces for porteføljе
$W_{v,t}$	Wiener-proces brugt i processen for variansen, V_t
W_t	Wiener-proces
$\widetilde{W}_t$	Wiener-proces under sandsynlighedsmålet $\mathbb{Q}$
Z_t	$\ln S_t$; payoff for S_t tilbagediskonteret

A.3 Græske bogstaver

α	konstant, der er bestemmende for niveauet på *steady state volatility* i Hestons model
α^*	risikojusteret α

β	konstant, der bestemmer hastigheden på *mean reversion* i Hestons model
β^*	risikojusteret β
γ_1	skævhed
γ_2	overskydende kurtosis
δ	standardafvigelse for spring i springmodeller
ϵ_k	et tilfældigt tal udtrukket fra standardnormalfordelingen
$\epsilon_{U,k}$	et tilfældigt tal mellem 0 og 1 udtrukket fra ligefordelingen
ϵ	den begåede fejl i Gauss-Kronrod integrationen
ε	tolerancen for fejl i Gauss-Kronrod integrationen
κ_i	i'te kumulant
λ	springintensiteten i spring-modeller
λ^*	risikojusteret λ
μ	drift i stokastisk proces; middelværdi
μ_n	n'te centrale moment
ρ	korrelationen mellem ændringer i kursen og volatiliteten
$\Pi(t, \mathcal{X})$	prisprocessen for en fordring med proces $\mathcal{X}$
σ	volatilitet i Black-Scholes modellen
σ_{imp}	implicit Black-Scholes volatilitet
σ_v	volatilitetens volatilitet i modeller med stokastisk volatilitet
ω	en sti i Monte Carlo simulationen
Ω	udfaldsrummet for stier i Monte Carlo simulation

B CD-ROM indhold

I dette bilag er indholdet af den medfølgende CD-ROM beskrevet. De samme filer kan også hentes ned fra afhandlingens hjemmeside http://www.nielses.dk/stud/cand/download

\Dokument
Afhandlingen i PDF-format nes-lsm-svjd.pdf
\Dokument\LaTeX
Afhandlingen i LaTeX format (skrevet i Scientific WorkPlace)

\NesQuant
Beregningsrutinerne, der er udviklet i denne afhandling, lavet som udvidelse til QuantLib
\NesQuant\nq
Kildekoden til NesQuant C++ beregningsrutinerne

\QuantLibMma
QuantLib for Mathematica: Grænsefladen mellem QuantLib og Mathematica, der er udviklet i forbindelse med denne afhandling.
\QuantLibMma*.cpp
Kildekoden til C++ delen af QuantLib for Mathematica, f.eks. mmaoptions.cpp.
\QuantLibMma\QuantLib
Mathematica-kildekoden til QuantLib for Mathematica, f.eks. Options.m.
\QuantLibMma\QuantLib\QuantLibMma.exe
Det færdige program, der startes fra Mathematica og stiller beregningsrutinerne til rådighed.

\QuantLib
QuantLib version 0.3.3
\QuantLib\Docs
Dokumentation af objekter og funktioner i QuantLib
\QuantLib\ql
Kildekoden til QuantLib

\MathReader

Et program fra Wolfram Research, der gør det muligt at læse de medfølgende Mathematica *notebooks*.

\Notebooks

Brugervejledningen til programmet: QuantLibMma-brugvejl.nb

\Notebooks\Figurer

De Mathematica-*notebooks*, der er brugt til at fremstille figurerne i afhandlingen.

\Notebooks\Tabeller

De Mathematica-*notebooks*, der er brugt til at fremstille tabellerne i afhandlingen.

C Brugervejledning

I dette bilag præsenteres en kortfattet brugervejledning til det medfølgende program, så det er muligt hurtigt at komme i gang med at bruge det. Vejledningen er udarbejdet på engelsk for at udenlandske brugere af programmet også kan få glæde af den. Vejledningen ligger på CD-ROM'en i Mathematica-format (\Notebooks\QuantLibMma-brugvejl.nb) og kan afprøves direkte.

Interfacet kaldes QuantLib for Mathematica, da det er tænkt som en generel udvidelse, der eksporterer beregningsfunktionaliteten fra QuantLib til Mathematica. Indtil videre stiller størstedelen af funktionerne dog beregningsrutinerne fra NesQuant (udviklet i denne afhandling) til rådighed.

C.1 Installation

In order to use QuantLib for Mathematica, you need to have a working copy of Mathematica installed on your computer. A 30-day trial version (with saving disabled) can be downloaded from http://www.wolfram.com/products/mathematica/trial.cgi

To install QuantLib for Mathematica, copy the folder \QuantLibMma\QuantLib from the CD-ROM into your Mathematica Applications directory. In version 5 on a Microsoft Windows system this is most often c:\Program Files\Wolfram Research\Mathematica\5.0\AddOns\Applications.

After having done this, you can start QuantLib for Mathematica from within Mathematica with the following command.

```
In[1]:=  << QuantLib`
```

To see a description you can use the following command:

```
In[2]:=  ?QuantLib
         QuantLib gives access to various financial functions from the QuantLib C++
         library. More info: http://www.nielses.dk/quantlib/mma
```

C.2 Valuing a European option

The basic idea of the QuantLib for Mathematica interface is that you can value a specific instrument in a specific model using a specific method. Depending on the method, it may be possible to specify additional options and parameters to control how the method calculates the value. If the method is not specified, a default method is used.

To demonstrate the concept, we start by valuing a european option using the Black-Scholes model. First we can use "?" to get a description of a `EuropeanOption`.

```
In[3]:=  ?EuropeanOption
         EuropeanOption[strike, residualtime, type] represents a European
         option of the given type (CallOption, PutOption or Straddle),
         exercise price (strike) and time to maturity (residualtime).
```

Then we use the variable `eur` to define a European call option.

```
In[4]:=  eur = EuropeanOption[625, 0.25, CallOption];
```

Then we define the model under which it should be priced:

```
In[5]:=  ?BlackScholesModel
         BlackScholesModel[stockprice, riskfreerate, dividend, vol] represents a
         model where the underlying asset is worth stockprice at time zero and the
         risk free rate, dividend yield and volatility are as given.
In[6]:=  bsmodel = BlackScholesModel[625, 0.0345, 0, 0.1823]
Out[6]=  BlackScholesModel[625, 0.0345, 0, 0.1823]
```

Now we use can calculate the value for this option. Automatically the default `Method -> Analytic` is used.

```
In[7]:=  Value[eur, bsmodel]
Out[7]=  25.4067
```

However, we can also use the Monte Carlo method. Here we use the `Keyfigures` function as we would also like to have the standard error returned.

```
In[8]:=  Keyfigures[eur, bsmodel, Method -> MonteCarlo]
Out[8]=  {Value -> 25.4827, StandardError -> 0.116867}
```

If we want better precision, we can use more sample paths.

```
In[9]:=  ?Samples
         Samples is an option used in conjunction with Method -> MonteCarlo that
         specifies how many sample paths should be simulated.
In[10]:= Keyfigures[eur, bsmodel, Method -> MonteCarlo, Samples -> 500000]
Out[10]= {Value -> 25.3578, StandardError -> 0.0521309}
```

Now we can use the standard Mathematica features to e.g. show how the standard error depends on the number of samples used.

```
In[11]:= Table[StandardError /. Keyfigures[eur, bsmodel, Method -> MonteCarlo,
               Samples -> s], {s, 10000, 100000, 10000}]
Out[11]= {0.373922, 0.259566, 0.211876, 0.184762, 0.164114, 0.149644, 0.139189, \
         0.130548, 0.122554, 0.115903}
```

C.3 European option in the SVJD-model

We can evaluate the same option in a different model, namely the SVJD-model.

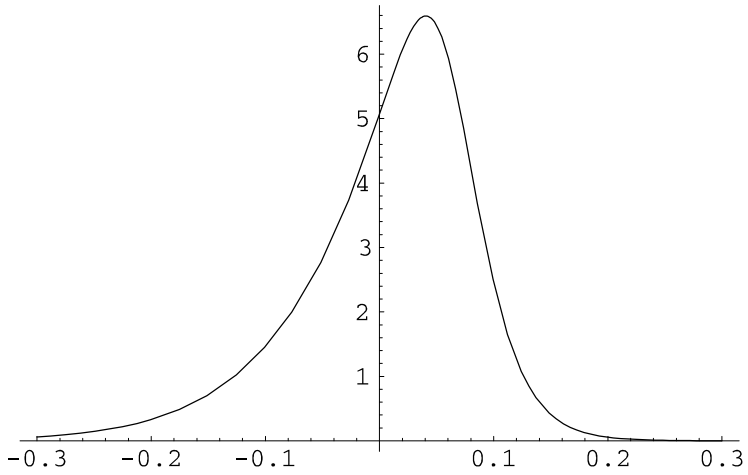
```
In[12]:= ?SVJDModel
         SVJDModel[underlying, riskFreeRate, dividendYield, volatility,
         volatilityOfVolatility, steadyStateVolatility,
         meanReversionRate, correlationUnderlyingVolatility,
         jumpIntensity, jumpMean, jumpStandardDeviation]
         represents a SVJD (stochastic volatility / jump diffusion)
         model with the given parameters.
```

The following parameters are estimated parameters for the S&P500 stock-index taken from Bakshi, Cao and Chen (1997, p. 2018).

```
In[13]:= svjdmodel = SVJDModel[625, 0.0345, 0, 0.1403, 0.38, 0.1403, 2.03, -0.57,
         0.59, -0.05, 0.07];
In[14]:= Keyfigures[eur, svjdmodel]
Out[14]= {Value -> 21.4055}
```

We also have the possibility to draw the probability density function for the return on the stock.

```
In[15]:= Plot[PDF[eur, svjdmodel, z], {z, -0.3, 0.3}]
```



We can also calculate the mean, variance, skewness and kurtosis for the return.

```
In[16]:= {Skewness[eur, svjdmodel], KurtosisExcess[eur, svjdmodel]}
Out[16]= {-1.02932, 2.12459}
```

If we want to know what the implied Black-Scholes volatility is for a given price calculated in the SVJD-model, we can use the ImpliedVolatility function. Obviously the same function can be used to find the implied volatility for a market observed price.

```
In[17]:= ImpliedVolatility[Value[eur, svjdmodel], eur, bsmodel]
Out[17]= 0.149863
```

C.4 Monte Carlo for a European option

Earlier we saw how to value a European option in the BS-model using Monte Carlo simulation. The same approach is used for the SVJD-model. In this example we also use the antithetic paths to reduce variance.

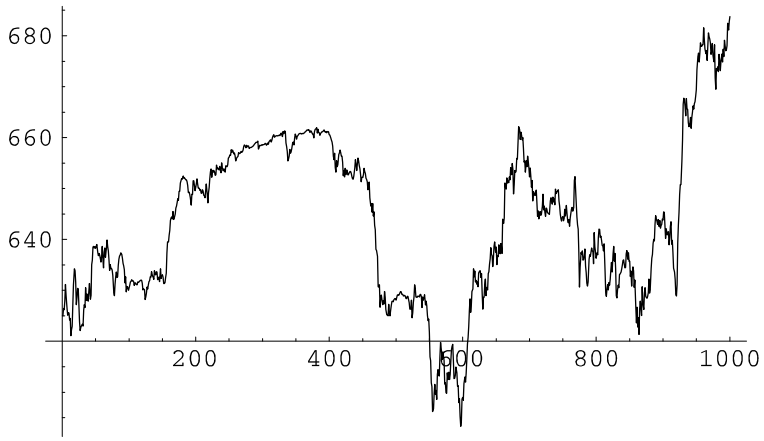
```
In[18]:= Keyfigures[eur, svjdmodel, Method -> MonteCarlo, AntitheticPaths -> True]
Out[18]= {Value -> 20.0498, StandardError -> 0.059956}
```

It is also possible to change the number of time steps. The default is 100.

```
In[19]:= Options[Keyfigures]
Out[19]= {Method -> Automatic, Samples -> Automatic, AntitheticPaths -> False,
          ControlVariate -> False, GridPoints -> 100, TimeSteps -> 100,
          ExerciseTimes -> Automatic, PolynomialDegree -> 2}
```

There is also a function to show the development of the underlier along a path.

```
In[20]:= ListPlot[MonteCarloPath[svjdmodel], PlotJoined -> True]
```



C.5 Finite differences for an American option

It is possible to use the method of finite differences to value an American option in the BS-model.

```
In[21]:= ?FiniteDifferences
         FiniteDifferences is an option for Value and Keyfigures
         specifying that the finite differences method should be used.
         GridPoints and TimeSteps can be used to adjust the accuracy.
In[22]:= am = AmericanOption[625, 0.25, CallOption];
In[23]:= Value[am, bsmodel, GridPoints -> 200, TimeSteps -> 4000]
Out[23]= 25.4067
```

C.6 Least-squares Monte Carlo for an American option

When the method MonteCarlo is specified when valuing an American option the least-squares Monte Carlo algorithm is used. The number of samples, time steps and number of basis functions (PolynomialDegree) can be specified.

```
In[24]:= Keyfigures[am, bsmodel, Method -> MonteCarlo, TimeSteps -> 50,
             PolynomialDegree -> 4]
Out[24]= {Value -> 25.2305, StandardError -> 0.360366}
```

In this case it is possible to use the European option price as a control variate.

```
In[25]:= Keyfigures[am, bsmodel, Method -> MonteCarlo, ControlVariate -> True]
Out[25]= {Value -> 25.1417, StandardError -> 0.194501}
```

C.7 American option in the SVJD-model

For an American option in the SVJD-model there is only one supported way to find the value, namely using the least-squares Monte Carlo method. This method is automatically chosen by default.

```
In[26]:= Keyfigures[am, svjdmodel, AntitheticPaths -> True, ControlVariate -> True]
Out[26]= {Value -> 21.4631, StandardError -> 0.045207}
```

For this method there is one further option. As the American option is effectively evaluated as a Bermudan option with a limited number of exercise points, the number of these can be modified. As default, ExercisePoints is chosen to be the same number as the number of time steps used in the path generation (TimeSteps).

```
In[27]:= Keyfigures[am, svjdmodel, AntitheticPaths -> True, ControlVariate -> True,
             ExercisePoints -> 50]
Out[27]= {Value -> 21.3721, StandardError -> 0.0513546}
```

D NesQuant C++ kildekode

I dette bilag vises kildekoden til de udvidelser til QuantLib, der er udviklet i denne afhandling. De rutiner, der er udviklet af afhandlingens forfatter, er defineret i et *namespace* kaldet NesQuant for at kunne adskille dem fra den kildekode, der er udviklet af QuantLib gruppen.

Den viste kode har mange afhængigheder til resten af QuantLib-biblioteket, der af pladshensyn ikke kan vises her. Den komplette kildekode til QuantLib kan findes på den medfølgende CD-ROM.

Til udviklingen af C++ programmerne er brugt Microsoft Visual C++ 6.0, og de brugte projektfiler er inkluderet sammen med kildekoden. Koden kan dog uændret bruges med mange andre compilere.

D.1 Brugerlicens

```
/*
 Copyright (C) 2003 Niels Elken Sønderby

 This file is part of QuantLib, a free-software/open-source library
 for financial quantitative analysts and developers - http://quantlib.org/

 QuantLib is free software: you can redistribute it and/or modify it under the
 terms of the QuantLib license.  You should have received a copy of the
 license along with this program; if not, please email ferdinando@ametrano.net
 The license is also available online at http://quantlib.org/html/license.html

 This program is distributed in the hope that it will be useful, but WITHOUT
 ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 FOR A PARTICULAR PURPOSE.  See the license for more details.
*/

/*
 NesQuant is an extension to QuantLib
 http://www.nielses.dk/quantlib/nesquant
*/
```

D.2 Analytisk SVJD beregning

D.2.1 svjdengine.hpp

```
#ifndef nesquant_svjdengine_h
#define nesquant_svjdengine_h

#include <ql/option.hpp>
#include <ql/PricingEngines/GenericEngine.hpp>

#include <complex>
using std::complex;

// really belongs in quantlib.h
#define QL_IMAG std::imag
#define QL_REAL std::real

// this won't work!
// #define QL_EXP std::exp
// #define QL_LOG std::log
// so we treat complex<double> seperately
#define QL_COMPLEX_EXP std::exp
#define QL_COMPLEX_LOG std::log

// this works, but issues a warning-message (macro redefinition)
// #define QL_SQRT std::sqrt
// #define QL_POW std::pow
// so we make a complex version as well
#define QL_COMPLEX_SQRT std::sqrt
#define QL_COMPLEX_POW std::pow

using namespace QuantLib;
using namespace QuantLib::PricingEngines;

namespace NesQuant {

    //! parameters for plain option calculation
    class PlainOptionParameters : public virtual Arguments {
      public:
        PlainOptionParameters() : type(Option::Type(-1)),
                                  underlying(Null<double>()),
                                  strike(Null<double>()),
                                  dividendYield(Null<double>()),
                                  riskFreeRate(Null<double>()),
                                  maturity(Null<double>()),
                                  volatility(Null<double>()) {}

        Option::Type type;
        double underlying, strike;
        Spread dividendYield;
        Rate riskFreeRate;
        Time maturity;
        double volatility;
    };

    //! parameters for stochastic volatility model (SV) (Heston 1993)
    class StochasticVolatilityArguments : public virtual Arguments {
      public:
        StochasticVolatilityArguments() :
                 volatilityOfVolatility(Null<double>()),
                 steadyStateVolatility(Null<double>()),
```

```
            meanReversionRate(Null<double>()),
            correlationUnderlyingVolatility(Null<double>()) {}
    double volatilityOfVolatility, steadyStateVolatility,
           meanReversionRate, correlationUnderlyingVolatility;
};

//! parameters for jump diffusion model (JD) (Bates 1996)
class JumpArguments : public virtual Arguments {
  public:
    JumpArguments() : jumpIntensity(Null<double>()),
                      jumpMean(Null<double>()),
                      jumpStandardDeviation(Null<double>()) {}
    double jumpIntensity, jumpMean, jumpStandardDeviation;
};

//! parameters for stochastic volatility model and jumps (SVJD) (Bates 1996)
class SVJDArguments : public PlainOptionParameters,
                      public StochasticVolatilityArguments,
                      public JumpArguments {
    void validate() const {
        //! \TODO Validation of arguments
    };
};

//! \brief SVJD option pricing engine for European options
/*! Pricing engine for european options in the stochastic volatility and
    jumps model (SVJD) introduced by Bates (1996).
    Contains the Heston (1993) SV square root model as a special case. \n

    Reference: \n
    Bates, David S. (1996): \n
    Jumps and stochastic volatility:
    exchange rate processes implicit in deutsche mark options \n
    Review of Financial Studies 9: p. 69-107 \n
    http://rfs.oupjournals.org/cgi/content/abstract/9/1/69
*/

class SVJDEngine;

class SVJDIntegrand
{
    public:
        SVJDIntegrand(int j, double S, double K, const SVJDEngine* engine)
            : j_(j), S_(S), K_(K), engine_(engine) {}

        double operator()(double u) const;
    private:
        int j_;
        double S_, K_;
        const SVJDEngine* engine_;
};

class SVJDPdfIntegrand
{
    public:
        SVJDPdfIntegrand(double z, const SVJDEngine* engine)
            : z_(z), engine_(engine) {}

        double operator()(double u) const;
    private:
        double z_;
        const SVJDEngine* engine_;
```

```
    };

    class SVJDEngine : public GenericEngine<SVJDArguments, OptionValue> {

      public:
        void calculate() const;

        //! gives the value of the probability density function for the
        //! stock return z = log(ST / S0)
        double pdf(double z) const;

      // these two need access to F
      friend double SVJDIntegrand::operator()(double u) const;
      friend double SVJDPdfIntegrand::operator()(double u) const;

      protected:
        // helper function
        void translateArguments() const;

        // mathematical symbols to make code more readable
        mutable double S_, K_;
        mutable Rate r_;
        mutable Spread q_;
        mutable Time T_;
        mutable double V_, sigmav_, betastar_, alpha_, rho_;
        mutable double lambdastar_, kmeanstar_, delta_;

        // auxiliary variables / functions
        inline double my(int j) const
            { return (3.0 - 2.0 * j) / 2.0; }

        inline double beta(int j) const
            { return betastar_ + rho_ * sigmav_ * (j - 2); }

        complex<double> gamma(int j, complex<double> u) const;

        complex<double> C(int j, complex<double> u) const;

        complex<double> D(int j, complex<double> u) const;

        complex<double> E(int j, complex<double> u) const;

        // moment generating functions (F1 and F2)
        complex<double> F(int j, complex<double> u) const;

        // probabilities (P1 and P2)
        double P(int j) const;
    };

}

#endif
```

D.2.2 svjdengine.cpp

```
#include <nq/Math/KronrodIntegral.hpp>
#include <nq/PricingEngines/svjdengine.hpp>

namespace NesQuant {
```

```
void SVJDEngine::translateArguments() const {
    // translate argument names to mathematical names
    S_ = arguments_.underlying;
    K_ = arguments_.strike;
    r_ = arguments_.riskFreeRate;
    q_ = arguments_.dividendYield;
    T_ = arguments_.maturity;
    V_ = arguments_.volatility * arguments_.volatility;
    sigmav_ = arguments_.volatilityOfVolatility;
    betastar_ = arguments_.meanReversionRate;
    alpha_ = (arguments_.steadyStateVolatility
                * arguments_.steadyStateVolatility)
             * betastar_;
    rho_ = arguments_.correlationUnderlyingVolatility;
    lambdastar_ = arguments_.jumpIntensity;
    kmeanstar_ = arguments_.jumpMean;
    delta_ = arguments_.jumpStandardDeviation;
}

void SVJDEngine::calculate() const {
    translateArguments();

    // calculate discount factors
    DiscountFactor dividendDiscount = QL_EXP(-q_ * T_);
    DiscountFactor riskFreeDiscount = QL_EXP(-r_ * T_);

    // calculate probabilities
    double MP1, MP2; // modified P1, P2

    switch (arguments_.type) {
      case Option::Call:
        MP1 = P(1);
        MP2 = P(2);
        break;
      case Option::Put:
        MP1 = P(1) - 1.0;
        MP2 = P(2) - 1.0;
        break;
      case Option::Straddle:
        MP1 = 2.0 * P(1) - 1.0;
        MP2 = 2.0 * P(2) - 1.0;
        break;
      default:
        throw IllegalArgumentError("SVJDEngine: invalid option type");
    }

    // calculate option price
    results_.value = S_ * dividendDiscount * MP1
                     - K_ * riskFreeDiscount * MP2;
}

complex<double> SVJDEngine::gamma(int j, complex<double> u) const {
    return QL_COMPLEX_SQRT( (rho_ * sigmav_ * u - beta(j))
                              * (rho_ * sigmav_ * u - beta(j))
                            - 2.0 * sigmav_ * sigmav_
                              * (my(j) * u + (u * u) / 2.0));
}

complex<double> SVJDEngine::C(int j, complex<double> u) const {
    complex<double> expgammajT = QL_COMPLEX_EXP(gamma(j, u) * T_);

    return (r_ - q_ - lambdastar_ * kmeanstar_) * u * T_
```

```
          - ((alpha_ * T_) / (sigmav_ * sigmav_))
            * (rho_ * sigmav_ * u - beta(j) - gamma(j, u))
          - ((2.0 * alpha_) / (sigmav_ * sigmav_))
   * QL_COMPLEX_LOG(1.0 + (rho_ * sigmav_ * u - beta(j) - gamma(j, u))
          / 2.0
            * ((1.0 - expgammajT) / gamma(j, u)));
}

complex<double> SVJDEngine::D(int j, complex<double> u) const {
    complex<double> expgammajT = QL_COMPLEX_EXP(gamma(j, u) * T_);

    return -2.0 * (my(j) * u + (u * u) / 2.0)
              / (rho_ * sigmav_ * u - beta(j)
                 + gamma(j, u) * (1.0 + expgammajT)
                   / (1.0 - expgammajT));
}

complex<double> SVJDEngine::E(int j, complex<double> u) const {
    double temp1;
    complex<double> temp2;

    temp1 = pow(1.0 + kmeanstar_, my(j) + 0.5);
    temp2 = QL_COMPLEX_POW(1.0 + kmeanstar_, u);

    return lambdastar_ * T_ * temp1
           * (temp2 * QL_COMPLEX_EXP(
                delta_ * delta_ * (my(j) * u + (u * u) / 2.0))
             - 1.0);
}

complex<double> SVJDEngine::F(int j, complex<double> u) const
{
    return QL_COMPLEX_EXP(C(j, u) + D(j, u) * V_ + E(j, u));
}

double SVJDIntegrand::operator()(double u) const {
    const complex<double> I(0,1);
    double temp = QL_LOG(K_/S_);
    return QL_IMAG(engine_->F(j_, I * u)
           * QL_COMPLEX_EXP(-I * u * temp))/u;
}

double SVJDEngine::P(int j) const {
    using NesQuant::Math::KronrodIntegral;
    KronrodIntegral integrator(0.00001);

    SVJDIntegrand integrand(j, S_, K_, this);

    double integral = integrator(integrand, 0, 300);

    return 0.5 + M_1_PI * integral;
}

double SVJDPdfIntegrand::operator()(double u) const {
    const complex<double> I(0,1);
    return QL_REAL(engine_->F(2, I * u)
           *QL_COMPLEX_EXP(-I * u * z_));
}

double SVJDEngine::pdf(double z) const {
    translateArguments();
```

```
            using NesQuant::Math::KronrodIntegral;
            KronrodIntegral integrator(0.00001);

            SVJDPdfIntegrand integrand(z, this);

            double integral = integrator(integrand, 0, 300);

            return M_1_PI * integral;
        }

}
```

D.2.3 kronrodintegral.hpp

```
#ifndef nesquant_kronrod_integral_h
#define nesquant_kronrod_integral_h

#include <ql/types.hpp>
#include <ql/errors.hpp>
#include <ql/DataFormatters.hpp>

namespace NesQuant {

namespace Math {

    //! Integral of a 1-dimensional function using the Gauss-Kronrod method
    /*! References: \n
        Gauss-Kronrod Integration
        <http://mathcssun1.emporia.edu/~oneilcat/ExperimentApplet3/ExperimentApplet3.html> \n
        NMS - Numerical Analysis Library
        <http://www.math.iastate.edu/burkardt/f_src/nms/nms.html>
    */

    class KronrodIntegral {
      public:
        KronrodIntegral(double tolerance_, long maxFunctionEvaluations);

        template <class F>
        double operator()(const F& f, double a, double b) {
            QL_REQUIRE(a < b,
                       "to compute an integral on [a,b] it must be a<b; "
                       "a="+DoubleFormatter::toString(a)+", "
                       "b="+DoubleFormatter::toString(b));
            functionEvaluations_ = 0;

            return GaussKronrod(f, a, b, tolerance_);
        }

        long functionEvaluations() { return functionEvaluations_; }

      private:

        template <class F>
        double GaussKronrod(const F& f,
                            const double a, const double b,
                            const double tolerance) {
            // weights for 7-point Gauss-Legendre integration
            // (only 4 values out of 7 are given as they are symmetric)
            static const double g7w[]= { 0.417959183673469,
                0.381830050505119, 0.279705391489277, 0.129484966168870 };
```

```
    // weights for 15-point Gauss-Kronrod integration
    static const double k15w[]={ 0.209482141084728, 0.204432940075298,
                          0.190350578064785, 0.169004726639267,
                          0.140653259715525, 0.104790010322250,
                          0.063092092629979, 0.022935322010529 };
    // abscissae (evaluation points)
    // for 15-point Gauss-Kronrod integration
    static const double k15t[]={ 0.000000000000000, 0.207784955007898,
                          0.405845151377397, 0.586087235467691,
                          0.741531185599394, 0.864864423359769,
                          0.949107912342758, 0.991455371120813 };

    const double halflength = (b - a) / 2;
    const double center = (a + b) / 2;

    double g7; // will be result of G7 integral
    double k15; // will be result of K15 integral

    double t, fsum; // t (abscissa) and f(t)
    double fc = f(center);
    g7 = fc * g7w[0];
    k15 = fc * k15w[0];

    // calculate g7 and half of k15
    int j, j2;
    for (j = 1, j2 = 2; j < 4; j++, j2 += 2) {
        t = halflength * k15t[j2];
        fsum = f(center - t) + f(center + t);
        g7  += fsum * g7w[j];
        k15 += fsum * k15w[j2];
    }

    // calculate other half of k15
    for (j2 = 1; j2 < 8; j2 += 2) {
        t = halflength * k15t[j2];
        fsum = f(center - t) + f(center + t);
        k15 += fsum * k15w[j2];
    }

    // multiply by (a - b) / 2
    g7 = halflength * g7;
    k15 = halflength * k15;

    // 15 more function evaluations have been used
    functionEvaluations_ += 15;

    // error is <= k15 - g7
    // if error is larger than tolerance then split the interval
    // in two and integrate recursively
    if (QL_FABS(k15 - g7) < tolerance)
        return k15;
    else {
        QL_REQUIRE(functionEvaluations_+30 <= maxFunctionEvaluations_,
            "maxFunctionEvaluations of "
            + IntegerFormatter::toString(maxFunctionEvaluations_)
            + " exceeded");
        return GaussKronrod(f, a, center, tolerance/2)
               + GaussKronrod(f, center, b, tolerance/2);
    }
}

double tolerance_;
```

```
        long functionEvaluations_;
        long maxFunctionEvaluations_;
    };

    inline KronrodIntegral::KronrodIntegral
        (double tolerance, long maxFunctionEvaluations = 500)
    : tolerance_(tolerance), maxFunctionEvaluations_(maxFunctionEvaluations) {
        QL_REQUIRE(tolerance > QL_EPSILON, "tolerance must be > 0");
        QL_REQUIRE(maxFunctionEvaluations >= 15,
                   "maxFunctionEvaluations must be >= 15");
    }

}

}

#endif
```

D.3 Monte Carlo for SVJD-model

D.3.1 svjdpathgenerator.hpp

```
#ifndef nesquant_montecarlo_svjdpath_generator_h
#define nesquant_montecarlo_svjdpath_generator_h

#include <ql/diffusionprocess.hpp>
#include <ql/MonteCarlo/path.hpp>
#include <ql/RandomNumbers/randomarraygenerator.hpp>

using namespace QuantLib::MonteCarlo;

namespace QuantLib {

    namespace MonteCarlo {

        template <class SG>
        class SVJDPathGenerator : public PathGenerator<SG> {
          public:
            SVJDPathGenerator(double riskFreeRate,
                  double dividendYield,
                  double volatility, double volatilityOfVolatility,
                  double steadyStateVolatility,
                  double meanReversionRate,
                  double correlationUnderlyingVolatility,
                  double jumpIntensity, double jumpMean,
                  double jumpStandardDeviation,
                  Time length, Size timeSteps,
                  const SG& generator);
            const sample_type& next() const;
            const sample_type& volatility() const;
            const sample_type& antithetic() const;
          private:
            double r_, q_;
            double v0_;
            double sigmav_;
            double alphabeta_; // alpha / betastar
            double betastar_;
            double rho_;
```

```
    double lambda_;
    double kmeanstar_;
    double delta_;

    mutable sample_type volatility_;

    RandomNumbers::MersenneTwisterUniformRng urng_;
    RandomNumbers::ICGaussianRng<
                      RandomNumbers::MersenneTwisterUniformRng,
                      Math::InverseCumulativeNormal> grng_;

    mutable std::vector<double> random1_;
    mutable std::vector<double> random2_;
    mutable std::vector<bool> isJump_;
    mutable std::vector<double> randomj_;
};

template <class SG>
SVJDPathGenerator<SG>::SVJDPathGenerator(
        double riskFreeRate, double dividendYield,
        double volatility, double volatilityOfVolatility,
        double steadyStateVolatility, double meanReversionRate,
        double correlationUnderlyingVolatility,
        double jumpIntensity, double jumpMean,
        double jumpStandardDeviation,
        Time length, Size timeSteps,
        const SG& generator)
: PathGenerator<SG>
    (Handle<DiffusionProcess>
         (new SquareRootProcess(1, 1, 1)), // dummy
     length, timeSteps, generator),
    volatility_(Path(timeGrid_),1.0),
    r_(riskFreeRate), q_(dividendYield),
    v0_(volatility * volatility),
    sigmav_(volatilityOfVolatility),
    alphabeta_(steadyStateVolatility * steadyStateVolatility),
    betastar_(meanReversionRate),
    rho_(correlationUnderlyingVolatility),
    lambda_(jumpIntensity),
    kmeanstar_(jumpMean), delta_(jumpStandardDeviation),
    random1_(next_.value.size()), random2_(next_.value.size()),
    isJump_(next_.value.size()), randomj_(next_.value.size())
    { }

template <class SG>
inline const typename SVJDPathGenerator<SG>::sample_type&
SVJDPathGenerator<SG>::next() const {
    typedef typename SG::sample_type sequence_type;

    const sequence_type& sequence1 = generator_.nextSequence();
    std::copy(sequence1.value.begin(), sequence1.value.end(),
              random1_.begin());
    const sequence_type& sequence2 = generator_.nextSequence();
    std::copy(sequence2.value.begin(), sequence2.value.end(),
              random2_.begin());

    double V = v0_;
    double dW, dWv, dJ = 0;
    double Zdrift, Zdiff;
    double Vdrift, Vdiff;

    double dt;
```

```
    Time t;

    for (Size i = 0; i < next_.value.size(); i++) {
        t = timeGrid_[i+1];
        dt = timeGrid_.dt(i);

        dW = random1_[i];
        dWv = rho_ * dW + QL_SQRT(1 - rho_ * rho_) * random2_[i];

        Vdrift = (betastar_ * (alphabeta_ - V)) * dt;
        Vdiff = sigmav_ * QL_SQRT(V * dt) * dWv;
        V += Vdrift + Vdiff;
        if (V < 0.00001) {
            Vdiff += -V + 0.00001;
            V = 0.00001;
        }

        if (lambda_ > 0) {
            isJump_[i] = (urng_.next().value < lambda_ * dt);
            if (isJump_[i]) {
                randomj_[i] = grng_.next().value;
                dJ = QL_LOG(1 + kmeanstar_)
                     - delta_ * delta_ / 2
                     + delta_ * randomj_[i];
            } else {
                dJ = 0;
            }
        }

        Zdrift = (r_ - q_ - lambda_ * kmeanstar_ - 0.5 * V) * dt;
        Zdiff = QL_SQRT(V * dt) * dW + dJ;

        next_.value.drift()[i] = Zdrift;
        next_.value.diffusion()[i] = Zdiff;
        volatility_.value.drift()[i] = Vdrift;
        volatility_.value.diffusion()[i] = Vdiff;
    }

    return next_;
}

template <class SG>
inline const typename SVJDPathGenerator<SG>::sample_type&
SVJDPathGenerator<SG>::antithetic() const {
    double V = v0_;
    double dW, dWv, dJ = 0;
    double Zdrift, Zdiff;
    double Vdrift, Vdiff;

    double dt;
    Time t;

    for (Size i = 0; i < next_.value.size(); i++) {
        t = timeGrid_[i+1];
        dt = timeGrid_.dt(i);

        dW = - random1_[i];
        dWv = - (rho_ * dW + QL_SQRT(1 - rho_ * rho_) * random2_[i]);

        Vdrift = (betastar_ * (alphabeta_ - V)) * dt;
        Vdiff = sigmav_ * QL_SQRT(V * dt) * dWv;
        V += Vdrift + Vdiff;
```

```
                if (V < 0.00001) {
                    Vdiff += -V + 0.00001;
                    V = 0.00001;
                }

                if (lambda_ > 0) {
                    if (isJump_[i]) {
                        dJ = QL_LOG(1 + kmeanstar_)
                             - delta_ * delta_ / 2
                             + delta_ * (- randomj_[i]);
                    } else {
                        dJ = 0;
                    }
                }

                Zdrift = (r_ - q_ - lambda_ * kmeanstar_ - 0.5 * V) * dt;
                Zdiff = QL_SQRT(V * dt) * dW + dJ;

                next_.value.drift()[i] = Zdrift;
                next_.value.diffusion()[i] = Zdiff;
                volatility_.value.drift()[i] = Vdrift;
                volatility_.value.diffusion()[i] = Vdiff;
            }

            return next_;
        }

        template <class SG>
        inline const typename SVJDPathGenerator<SG>::sample_type&
        SVJDPathGenerator<SG>::volatility() const {
            return volatility_;
        }

        typedef SVJDPathGenerator
                <RandomNumbers::GaussianRandomSequenceGenerator>
                     GaussianSVJDPathGenerator;

    }

}

#endif
```

D.3.2 mcsvjdengine.hpp

```
#ifndef nesquant_mcsvjd_engine_h
#define nesquant_mcsvjd_engine_h

#include <ql/grid.hpp>
#include <ql/MonteCarlo/montecarlomodel.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
#include <ql/TermStructures/flatforward.hpp>
#include <ql/DayCounters/Actual365.hpp>
#include <ql/Calendars/Target.hpp>

#include <nq/MonteCarlo/svjdpathgenerator.hpp>
#include <nq/PricingEngines/svjdengine.hpp>

using namespace QuantLib;
using namespace QuantLib::PricingEngines;
```

```
using namespace QuantLib::MonteCarlo;
using namespace QuantLib::TermStructures;
using namespace QuantLib::Calendars;
using namespace QuantLib::DayCounters;
using QuantLib::Math::Statistics;
using NesQuant::SVJDArguments;

namespace NesQuant {

    template<class RNG = MonteCarlo::PseudoRandom,
             class S = Statistics>
    class MCSVJDEngine
    : public GenericEngine<SVJDArguments, OptionValue>,
      public McSimulation<SingleAsset<RNG>, S> {
      public:
        void calculate() const;
        typedef typename
        McSimulation<SingleAsset<RNG>,S>::path_generator_type
            path_generator_type;
        typedef typename
        McSimulation<SingleAsset<RNG>,S>::path_pricer_type
            path_pricer_type;
        typedef typename
        McSimulation<SingleAsset<RNG>,S>::stats_type
            stats_type;
        // constructor
        MCSVJDEngine(Size maxTimeStepsPerYear,
                     bool antitheticVariate = false,
                     bool controlVariate = false,
                     Size requiredSamples = Null<int>(),
                     double requiredTolerance = Null<double>(),
                     Size maxSamples = Null<int>(),
                     long seed = 0);
        // McSimulation implementation
        Handle<path_generator_type> pathGenerator() const;
        Handle<path_pricer_type> pathPricer() const;
        TimeGrid timeGrid() const;
        // data members
        Size maxTimeStepsPerYear_;
        Size requiredSamples_, maxSamples_;
        double requiredTolerance_;
        long seed_;
    };


    // inline definitions

    template<class RNG, class S>
    inline
    MCSVJDEngine<RNG,S>::MCSVJDEngine(Size maxTimeStepsPerYear,
                                      bool antitheticVariate,
                                      bool controlVariate,
                                      Size requiredSamples,
                                      double requiredTolerance,
                                      Size maxSamples,
                                      long seed)
    : McSimulation<SingleAsset<RNG>,S>(antitheticVariate, false),
      maxTimeStepsPerYear_(maxTimeStepsPerYear),
      requiredSamples_(requiredSamples),
      maxSamples_(maxSamples),
      requiredTolerance_(requiredTolerance),
      seed_(seed) {
```

```
        QL_REQUIRE(!controlVariate,
                   "MCSVJDEngine: controlVariate not supported");
  }

// template definitions

template<class RNG, class S>
inline Handle<QL_TYPENAME MCSVJDEngine<RNG,S>::path_generator_type>
MCSVJDEngine<RNG,S>::pathGenerator() const
{
    TimeGrid grid = timeGrid();

    typename RNG::rsg_type gen =
        RNG::make_sequence_generator(grid.size(), seed_);

    Time length = arguments_.maturity;
    Size timeSteps = grid.size();

    return Handle<path_generator_type>
               (new GaussianSVJDPathGenerator(
                   arguments_.riskFreeRate,
                   arguments_.dividendYield,
                   arguments_.volatility,
                   arguments_.volatilityOfVolatility,
                   arguments_.steadyStateVolatility,
                   arguments_.meanReversionRate,
                   arguments_.correlationUnderlyingVolatility,
                   arguments_.jumpIntensity,
                   arguments_.jumpMean,
                   arguments_.jumpStandardDeviation,
                   length, timeSteps,
                   gen));
}


template<class RNG, class S>
inline void MCSVJDEngine<RNG,S>::calculate() const {

    QL_REQUIRE(requiredTolerance_ != Null<double>() ||
               int(requiredSamples_) != Null<int>(),
               "MCSVJDEngine::calculate: "
               "neither tolerance nor number of samples set");

    //! Initialize the one-factor Monte Carlo
    mcModel_ =
        Handle<MonteCarloModel<
                    SingleAsset<RNG>, S> >(
           new MonteCarloModel<
                    SingleAsset<RNG>, S>(
               pathGenerator(), pathPricer(), S(),
               antitheticVariate_));

    if (requiredTolerance_ != Null<double>()) {
        if (int(maxSamples_) != Null<int>())
            value(requiredTolerance_, maxSamples_);
        else
            value(requiredTolerance_);
    } else {
        valueWithSamples(requiredSamples_);
    }

    results_.value = mcModel_->sampleAccumulator().mean();
```

```
        if (RNG::allowsErrorEstimate)
            results_.errorEstimate =
                mcModel_->sampleAccumulator().errorEstimate();
    }


    template <class RNG, class S>
    inline
    Handle<QL_TYPENAME MCSVJDEngine<RNG,S>::path_pricer_type>
    MCSVJDEngine<RNG,S>::pathPricer() const {
        RelinkableHandle<TermStructure> ts(Handle<TermStructure>
            (new FlatForward(Date::todaysDate(),
                             TARGET().advance(Date::todaysDate(), 2, Days),
                             arguments_.riskFreeRate, Actual365())));

        return Handle<MCSVJDEngine<RNG,S>::path_pricer_type>(
            new EuropeanPathPricer(arguments_.type,
                                   arguments_.underlying,
                                   arguments_.strike,
                                   ts));
    }

    template <class RNG, class S>
    inline TimeGrid MCSVJDEngine<RNG,S>::timeGrid() const {
        return TimeGrid(arguments_.maturity,
                        Size(arguments_.maturity *
                             maxTimeStepsPerYear_));
    }

}


#endif
```

D.4 Least-squares Monte Carlo

D.4.1 lsmvanillaengine.hpp

```
#ifndef nesquant_lsmvanillaengine_h
#define nesquant_lsmvanillaengine_h

#include <ql/MonteCarlo/europeanpathpricer.hpp>
#include <ql/diffusionprocess.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/rngtypedefs.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
#include <ql/payoff.hpp>
#include <ql/MonteCarlo/path.hpp>
#include <ql/Math/statistics.hpp>

using namespace QuantLib;
using namespace QuantLib::PricingEngines;
using namespace QuantLib::MonteCarlo;
using namespace QuantLib::RandomNumbers;
using namespace QuantLib::Math;

namespace NesQuant {
```

```
    class LSMVanillaEngine : public VanillaEngine {

    public:
        LSMVanillaEngine::LSMVanillaEngine(Size samples = 100,
                                           Size timeSteps = 50,
                                           Size degree = 2,
                                           bool antithetic = false,
                                           bool controlVariate = false,
                                           long seed = 0) :
                                           timeSteps_(timeSteps),
                                           degree_(degree),
                                           samples_(samples),
                                           seed_(seed),
                                           antitheticVariate_(antithetic),
                                           controlVariate_(controlVariate) {
          };

        void calculate() const;

    private:
        Size timeSteps_;
        Size degree_; // highest degree for polynomial
        Size samples_;
        bool antitheticVariate_, controlVariate_;
        long seed_;
        mutable Statistics sampleAccumulator_;
    };

}

#endif
```

D.4.2 lsmvanillaengine.cpp

```
#include <nq/PricingEngines/lsmvanillaengine.hpp>
#include <nq/Math/polynomialfit.hpp>
#include <iostream>

using namespace std;

namespace NesQuant {

    void LSMVanillaEngine::calculate() const {

        double s0 = arguments_.underlying;
        double maturity = arguments_.maturity;
        Handle<Payoff> &payoff = arguments_.payoff;

        Handle<DiffusionProcess> bs(new BlackScholesProcess(
                                arguments_.riskFreeTS,
                                arguments_.dividendTS,
                                arguments_.volTS,
                                s0));

        GaussianRandomSequenceGenerator gen =
            PseudoRandom::make_sequence_generator(timeSteps_, seed_);

        Handle<GaussianPathGenerator> pathGenerator =
            Handle<GaussianPathGenerator> (new GaussianPathGenerator(
```

```
          bs, maturity, timeSteps_, gen));

// if antithetic paths are used, twice as many paths are used
Size modSamples; // modified samples
if (antitheticVariate_)
    modSamples = 2 * samples_;
else
    modSamples = samples_;

// Initialize control variate
double controlVariateValue;
std::vector<double> controlVariatePathValue(modSamples);
arguments_.exerciseType = Exercise::European;
Handle<PlainVanillaPayoff> payoff2 = arguments_.payoff;
EuropeanPathPricer controlPP(payoff2->optionType(), s0,
                             payoff2->strike(), arguments_.riskFreeTS);
if (controlVariate_) {
    AnalyticEuropeanEngine controlPE;

    VanillaOptionArguments* controlArguments =
        dynamic_cast<VanillaOptionArguments*>(
            controlPE.arguments());
    *controlArguments = arguments_;
    controlPE.calculate();

    const VanillaOptionResults* controlResults =
        dynamic_cast<const VanillaOptionResults*>(controlPE.results());
    controlVariateValue = controlResults->value;
}

// generate paths
std::vector<std::vector<double> >
    spaths(modSamples, std::vector<double>(timeSteps_));

GaussianPathGenerator::sample_type pathSample = pathGenerator->next();

for (Size i = 0; i < modSamples; ++i) {
    if (antitheticVariate_ && i%2 == 1)
        pathSample = pathGenerator->antithetic();
    else
        pathSample = pathGenerator->next();

    double s = s0;
    for (Size j = 0; j < timeSteps_; ++j) {
        s *= QL_EXP(pathSample.value[j]);
        spaths[i][j] = s;
    }

    if (controlVariate_) {
        controlVariatePathValue[i] = controlPP(pathSample.value);
    }
}

// set values at maturity
std::vector<double> U(modSamples);
std::vector<Size> stoppingTime(modSamples, -1); // never exercised = -1

Size k = timeSteps_;
DiscountFactor discount = arguments_.riskFreeTS->discount(maturity);

for (i = 0; i < modSamples; ++i) {
    U[i] = discount * (*payoff)(spaths[i][k-1]);
```

```
}

// move backwards
double dt = maturity / timeSteps_;

for (k = timeSteps_ - 1; k > 0; --k) {
    cout << "k : " << k << endl;

    // find in-the-money paths and (x, y) regression data
    std::vector<Size> itmpaths;
    std::vector<double> xdata;
    std::vector<double> ydata;

    for (i = 0; i < modSamples; ++i) {
        double s = spaths[i][k-1];
        if ((*payoff)(s) > 0) {
            itmpaths.push_back(i);
            xdata.push_back(s);
            ydata.push_back(U[i]);
        }
    }

    if (itmpaths.size() > 0) {
        // do regression
        PolynomialFit pf(xdata, ydata, degree_);
        const Array contval = pf.value();

        // compare continuation value with exercise value
        discount = arguments_.riskFreeTS->discount(k * dt);
        for (Size l = 0; l < itmpaths.size(); ++l) {
            i = itmpaths[l];
            double exerval = discount * (*payoff)(xdata[l]);

            if (exerval > contval[l]) {
                U[i] = exerval;
                stoppingTime[i] = k;
            }
        }
    }
}

// adjust for control variate
if (controlVariate_)
    for (i = 0; i < modSamples; i++) {
        U[i] += controlVariateValue - controlVariatePathValue[i];
    }

// calculate average
if (antitheticVariate_)
    for (i = 0; i < modSamples; i+=2) {
        sampleAccumulator_.add((U[i] + U[i+1]) / 2);
    }
else
    for (i = 0; i < samples_; i++) {
        sampleAccumulator_.add(U[i]);
    }
double value = sampleAccumulator_.mean();

// compare with exercise at time 0
if ((*payoff)(s0) > value) {
    value = (*payoff)(s0);
    sampleAccumulator_.reset();
```

```
            for (i = 0; i < modSamples; i++) {
                if ((!antitheticVariate_) || i%2==0)
                    sampleAccumulator_.add(value);
                stoppingTime[i] = 0;
            }
        }

        // return value
        results_.value = value;
        results_.errorEstimate = sampleAccumulator_.errorEstimate();
    }

}
```

D.4.3 polynomialfit.hpp

```
#ifndef nesquant_polynomialfit_h
#define nesquant_polynomialfit_h

#include <vector>
#include <ql/Math/svd.hpp>

using namespace QuantLib;
using namespace QuantLib::Math;

namespace NesQuant {

    class PolynomialFit {

    public:
        PolynomialFit::PolynomialFit(Matrix data, Size degree = 2)
                  : data_(data), degree_(degree),
                    fitFunctions_(degree_ * (data.columns() - 1) + 1),
                    parameters_(Array(fitFunctions_)),
                    value_(Array(data_.rows())),
                    isPerformed_(false) {
            QL_REQUIRE(data.rows() > 0, "no data");
            QL_REQUIRE(degree >= 1, "degree of polynomium must be >= 1");
        };

        PolynomialFit::PolynomialFit(std::vector<double> x,
                                     std::vector<double> y,
                                     Size degree = 2)
                  : degree_(degree),
                    fitFunctions_(degree_ + 1),
                    parameters_(Array(fitFunctions_)),
                    value_(Array(x.size())),
                    isPerformed_(false) {
            QL_REQUIRE(x.size() == y.size(), "vectors must have same size");
            QL_REQUIRE(x.size() > 0, "empty vectors not allowed");
            QL_REQUIRE(degree >= 1, "degree of polynomium must be >= 1");

            data_ = Matrix(x.size(), 2);

            for (Size i = 0; i < x.size(); ++i) {
                data_[i][0] = x[i];
                data_[i][1] = y[i];
            }
        };
```

```
        PolynomialFit::PolynomialFit(std::vector<double> x,
                                     std::vector<double> y,
                                     std::vector<double> z,
                                     Size degree = 2)
                  : degree_(degree),
                    fitFunctions_(2 * degree_ + 1),
                    parameters_(Array(fitFunctions_)),
                    value_(Array(x.size())),
                    isPerformed_(false) {
            QL_REQUIRE(x.size() == y.size() && y.size() == z.size(),
                       "vectors must have same size");
            QL_REQUIRE(x.size() > 0, "empty vectors not allowed");
            QL_REQUIRE(degree >= 1, "degree of polynomium must be >= 1");

            data_ = Matrix(x.size(), 3);

            for (Size i = 0; i < x.size(); ++i) {
                data_[i][0] = x[i];
                data_[i][1] = y[i];
                data_[i][2] = z[i];
            }
        };

        const Array value();
        const Array parameters();

    private:
        void perform();

        Matrix data_;
        Size degree_;
        Size fitFunctions_;
        mutable bool isPerformed_;
        mutable Array value_;
        mutable Array parameters_;
    };

}

#endif
```

D.4.4 polynomialfit.cpp

```
#include <nq/Math/polynomialfit.hpp>

namespace NesQuant {

    const Array PolynomialFit::value() {
        perform();
        return value_;
    }

    const Array PolynomialFit::parameters() {
        perform();
        return parameters_;
    }

    void PolynomialFit::perform() {
        if (!isPerformed_) {
            try {
```

```
// SVD can only handle mxn matrices where m>=n
bool useTranspose = false;
if (data_.rows() < fitFunctions_)
    useTranspose = true;

// perform the fit using singular value decomposition

// make design matrix M
Matrix designMat(data_.rows(), fitFunctions_);

for (Size i = 0; i < data_.rows(); ++i) {
    designMat[i][0] = 1;
    Size j = 1;
    for (Size k = 0; k < data_.columns() - 1; ++k) {
        double powx = data_[i][k];
        designMat[i][j] = powx;
        j++;
        for (Size l = 2; l <= degree_; ++l) {
            powx *= data_[i][k];
            designMat[i][j] = powx;
            j++;
        }
    }
}

if (useTranspose)
    designMat = transpose(designMat);

// do singular value decomposition
SVD svd(designMat);
Matrix U, UT, V;
Array s;
svd.getU(U);
UT = transpose(U);
svd.getV(V);
svd.getSingularValues(s);

// create diagonalmatrix S = diag(1/s)
Matrix S(s.size(), s.size(), 0.0);
for (i = 0; i < s.size(); ++i) {
    if (QL_FABS(s[i]) > QL_EPSILON)
        S[i][i] = 1 / s[i];
    else
        S[i][i] = 0.0;
}

Array f_(data_.rows());
for (i = 0; i < f_.size(); ++i) {
    f_[i] = data_[i][data_.columns() - 1];
}

if (!useTranspose) {
    // calculate parameters a = V diag(1/s) UT y
    parameters_ = V * S * UT * f_;
    // calculate value v = M a
    value_ = designMat * parameters_;
} else {
    parameters_ = U * S * transpose(V) * f_;
    value_ = transpose(designMat) * parameters_;
}

isPerformed_ = true;
```

```
            } catch (...) {
                // if an error occurs unset flag and rethrow error
                isPerformed_ = false;
                throw;
            }
        }
    }

}
```

D.5 LSM for SVJD-modellen

D.5.1 lsmsvjdengine.hpp

```
#ifndef nesquant_lsmsvjdengine_h
#define nesquant_lsmsvjdengine_h

#include <ql/diffusionprocess.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/rngtypedefs.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
#include <ql/payoff.hpp>
#include <ql/MonteCarlo/path.hpp>
#include <ql/Math/matrix.hpp>
#include <ql/MonteCarlo/europeanpathpricer.hpp>

#include <nq/PricingEngines/svjdengine.hpp>
#include <nq/MonteCarlo/svjdpathgenerator.hpp>
#include <nq/Math/polynomialfit.hpp>

using namespace QuantLib;
using namespace QuantLib::PricingEngines;
using namespace QuantLib::MonteCarlo;
using namespace QuantLib::RandomNumbers;
using namespace QuantLib::Math;

namespace NesQuant {

    class LSMSVJDEngine : public GenericEngine<SVJDArguments, OptionValue> {

    public:
        LSMSVJDEngine::LSMSVJDEngine(Size samples = 100,
                                     Size timeSteps = 50,
                                     Size exerciseTimes = 50,
                                     Size degree = 2,
                                     bool antitheticVariate = false,
                                     bool controlVariate = false,
                                     long seed = 0) :
                                         timeSteps_(timeSteps),
                                         exerciseTimes_(exerciseTimes),
                                         degree_(degree),
                                         samples_(samples),
                                         seed_(seed),
                                         antitheticVariate_(antitheticVariate),
                                         controlVariate_(controlVariate) {
                    QL_REQUIRE(timeSteps % exerciseTimes == 0,
                               "timeSteps must be divisible with exerciseTimes");
```

```
                                        };

        void calculate() const;
        const Matrix exercisePoints() const {
            return exercisePoints_;
        }
        const Statistics& sampleAccumulator() const {
            return sampleAccumulator_;
        }

    private:
        bool antitheticVariate_, controlVariate_;
        Size timeSteps_, exerciseTimes_;
        Size degree_; // highest degree for polynomial
        Size samples_;
        long seed_;
        mutable Matrix exercisePoints_;
        mutable Statistics sampleAccumulator_;
    };

}

#endif
```

D.5.2 lsmsvjdengine.cpp

```
#include <nq/PricingEngines/lsmsvjdengine.hpp>
#include <iostream>
using namespace std;

namespace NesQuant {

    void LSMSVJDEngine::calculate() const {

        double s0 = arguments_.underlying,
               v0 = arguments_.volatility * arguments_.volatility;
        double maturity = arguments_.maturity;
        PlainVanillaPayoff payoff(arguments_.type, arguments_.strike);

        GaussianRandomSequenceGenerator gen =
            PseudoRandom::make_sequence_generator(timeSteps_, seed_);

        Handle<GaussianSVJDPathGenerator> pathGenerator =
            Handle<GaussianSVJDPathGenerator>
                       (new GaussianSVJDPathGenerator(
                           arguments_.riskFreeRate,
                           arguments_.dividendYield,
                           arguments_.volatility,
                           arguments_.volatilityOfVolatility,
                           arguments_.steadyStateVolatility,
                           arguments_.meanReversionRate,
                           arguments_.correlationUnderlyingVolatility,
                           arguments_.jumpIntensity,
                           arguments_.jumpMean,
                           arguments_.jumpStandardDeviation,
                           maturity, timeSteps_,
                           gen));

        // number of time steps between values used as exercise times
        Size timeDistance = timeSteps_ / exerciseTimes_;
```

```
// if antithetic paths are used, twice as many paths are used
Size modSamples; // modified samples
if (antitheticVariate_)
    modSamples = 2 * samples_;
else
    modSamples = samples_;

// Initialize control variate
double controlVariateValue;
std::vector<double> controlVariatePathValue(modSamples);
EuropeanPathPricer_old controlPP(arguments_.type, s0, arguments_.strike,
                      QL_EXP(-arguments_.riskFreeRate * maturity), false);
if (controlVariate_) {
    SVJDEngine controlPE;

    SVJDArguments* controlArguments =
        dynamic_cast<SVJDArguments*>(
            controlPE.arguments());
    *controlArguments = arguments_;
    controlPE.calculate();

    const OptionValue* controlResults =
        dynamic_cast<const OptionValue*>(controlPE.results());
    controlVariateValue = controlResults->value;
}

// generate paths
std::vector<std::vector<double> >
    spaths(modSamples, std::vector<double>(exerciseTimes_));
std::vector<std::vector<double> >
    vpaths(modSamples, std::vector<double>(exerciseTimes_));

GaussianSVJDPathGenerator::sample_type
    pathSample(pathGenerator->timeGrid(), 1.0);
GaussianSVJDPathGenerator::sample_type
    volPathSample(pathGenerator->timeGrid(), 1.0);

for (Size i = 0; i < modSamples; i++) {
    if (antitheticVariate_ && i%2 == 1)
        pathSample = pathGenerator->antithetic();
    else
        pathSample = pathGenerator->next();
    volPathSample = pathGenerator->volatility();

    double s = s0, v = v0, k = 0;
    for (Size j = 0; j < timeSteps_; j++) {
        s *= QL_EXP(pathSample.value[j]);
        v += volPathSample.value[j];
        if ((j + 1) % timeDistance == 0) {
            spaths[i][k] = s;
            vpaths[i][k] = v;
            k++;
        }
    }

    if (controlVariate_) {
        controlVariatePathValue[i] = controlPP(pathSample.value);
    }
}

// set values at maturity
```

```
std::vector<double> U(modSamples);
exercisePoints_ = Matrix(modSamples, 3, -1.0); // (t, St, Vt)

Size k = exerciseTimes_;
DiscountFactor discount = QL_EXP(-arguments_.riskFreeRate * maturity);

for (i = 0; i < modSamples; i++) {
    U[i] = discount * payoff(spaths[i][k-1]);
}

// move backwards
double dt = maturity / exerciseTimes_;

for (k = exerciseTimes_ - 1; k > 0; k--) {
    cout << "k : " << k << endl;

    // find in-the-money paths and (x, y) regression data
    std::vector<Size> itmpaths;
    std::vector<double> xdata, vdata, fdata;

    for (i = 0; i < modSamples; i++) {
        double s = spaths[i][k-1];
        if (payoff(s) > 0) {
            itmpaths.push_back(i);
            xdata.push_back(s);
            vdata.push_back(vpaths[i][k-1]);
            fdata.push_back(U[i]);
        }
    }

    if (itmpaths.size() > 0) {
        // do regression
        PolynomialFit pf(xdata, vdata, fdata, degree_);
        const Array contval = pf.value();

        // compare continuation value with exercise value
        discount = QL_EXP(-arguments_.riskFreeRate * k * dt);
        for (Size l = 0; l < itmpaths.size(); l++) {
            i = itmpaths[l];
            double exerval = discount * payoff(xdata[l]);

            if (exerval > contval[l]) {
                U[i] = exerval;
                exercisePoints_[i][0] = k * dt;
                exercisePoints_[i][1] = xdata[l];
                exercisePoints_[i][2] = QL_SQRT(vdata[l]);
            }
        }
    }
}

// adjust for control variate
if (controlVariate_)
    for (i = 0; i < modSamples; i++) {
        U[i] = U[i] + 1.0 * (controlVariateValue - controlVariatePathValue[i]);
    }

// calculate average
if (antitheticVariate_)
    for (i = 0; i < modSamples; i+=2) {
        sampleAccumulator_.add((U[i] + U[i+1]) / 2);
    }
```

```
        else
            for (i = 0; i < samples_; i++) {
                sampleAccumulator_.add(U[i]);
            }

        double value = sampleAccumulator_.mean();

        // compare with exercise at time 0
        if (payoff(s0) > value) {
            value = payoff(s0);
            sampleAccumulator_.reset();
            for (i = 0; i < modSamples; i++) {
                if ((!antitheticVariate_) || i%2==0)
                    sampleAccumulator_.add(value);
                exercisePoints_[i][0] = 0;
                exercisePoints_[i][1] = s0;
                exercisePoints_[i][2] = arguments_.volatility;
            }
        }

        // return value
        results_.value = value;
        results_.errorEstimate = sampleAccumulator_.errorEstimate();
    }

}
```

E Mathematica-C++ kildekode

Til at forbinde den udviklede C++ kode med Mathematica bruges Mathematicas dataudvekslingsprotokol MathLink[1]. Denne integration kræver en del programkode, både i Mathematica-sproget og C++. Af pladshensyn er kun en lille del af den nødvendige kode medtaget i dette bilag. Den resterende kildekode kan findes på CD-ROM'en.

Som eksempel for at forstå principperne er valgt LSM metoden for SVJD-modellen. Koden i `options.m` sørger for at oversætte `Keyfigures[AmericanOption[...],SVJDModel[...]]` til `nqAmericanOptionSVJDLSM`. Herefter sørger filen `QuantLibMma.tm` for at binde Mathematica og C++ sammen, og til sidst kalder `nqAmericanOptionSVJDLSM` funktionen i `mmaoptions.cpp` de korrekte rutiner i NesQuant-biblioteket.

E.1 options.m

```
(* :Title: Options *)

(* :Context: QuantLib`Options` *)

(* :Author: Niels Elken S\UNICODE{0xf8}nderby *)

(* :Mathematica Version: 4.0 *)

(* :Copyright: Copyright (C) 2003 Niels Elken S\UNICODE{0xf8}nderby *)

(* :License:

   This file is part of QuantLib for Mathematica, a Mathematica extension for
   QuantLib, a free-software/open-source financial C++ library
   - http://www.nielses.dk/quantlib/mma/
   - http://quantlib.org/

   QuantLib for Mathematica is free software: you can redistribute it and/or
   modify it under the terms of the QuantLib license.  You should have
   received a copy of the license along with this program; if not, please
   email ferdinando@ametrano.net The license is also available online at
   http://quantlib.org/html/license.html
```

[1] http://www.wolfram.com/solutions/mathlink/

```
   This program is distributed in the hope that it will be useful, but
   WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
   or FITNESS FOR A PARTICULAR PURPOSE. See the license for more details.
*)

BeginPackage["QuantLib`Options`",
"QuantLib`Common`QuantLibCommon`" (* Needs SymbolToNumber functionality *)
]

(* Usage messages *)

Keyfigures::usage =
"Keyfigures[instrument, model] gives all keyfigures (including Value)
for the given instrument under the given model."

Value::usage =
"Value[instrument, model] is a short form
for Value /. Keyfigures[instrument, model]
and gives the value of the instrument under the given model."

CallOption::usage =
"CallOption is an option type where the buyer has the right to buy the
underlying at maturity."

PutOption::usage =
"PutOption is an option type where the buyer has the right to sell the
underlying at maturity."

Straddle::usage =
"Straddle is the sum of a call option and a put option (see CallOption,
PutOption)."

EuropeanOption::usage =
"EuropeanOption[strike, residualtime, type] represents a European
option of the given type (CallOption, PutOption or Straddle), exercise
price (strike) and time to maturity (residualtime)."

AmericanOption::usage =
"AmericanOption[strike, residualtime, type] represents an American
option of the given type (CallOption, PutOption or Straddle), exercise
price (strike) and time to maturity (residualtime)."

BlackScholesModel::usage =
"BlackScholesModel[stockprice, riskfreerate, dividend, vol] represents a
model where the underlying asset is worth stockprice at time zero and the
risk free rate, dividend yield and volatility are as given."

SVJDModel::usage =
"SVJDModel[underlying, riskFreeRate, dividendYield, volatility,
volatilityOfVolatility, steadyStateVolatility,
meanReversionRate, correlationUnderlyingVolatility,
jumpIntensity, jumpMean, jumpStandardDeviation] represents a SVJD
(stochastic volatility / jump diffusion) model with the given parameters."

AntitheticPaths::usage =
"AntitheticPaths is an option for Value and Keyfigures (when
Method -> MonteCarlo is chosen) and MonteCarloPath to specifiy whether
antithetic paths should be used in the simulation."

ControlVariate::usage =
"ControlVariate is an option used in conjunction with
Method -> MonteCarlo that specifies whether a control variate should be used."
```

```
Samples::usage =
"Samples is an option used in conjunction with Method -> MonteCarlo that
specifies how many sample paths should be simulated."

ImpliedVolatility::usage =
"ImpliedVolatility[price, instr, bsmodel] gives the volatility which makes
the price equal the price calculated in the given Black-Scholes model for
the given instrument ."

PDF::usage = "PDF[instr, model, z] gives the probability density function
for the return of the underlying instrument in the given model at expiry
evaluated at z."

Mean::usage = "Mean of PDF"

Variance::usage = "Variance of PDF"

Skewness::usage = "Skewness of PDF"

KurtosisExcess::usage = "KurtosisExcess of PDF"

GridPoints::usage =
"GridPoints is an option used in connection with Method -> FiniteDifferences
that specifies the number of grid points for the price of the underlier."

TimeSteps::usage =
"TimeSteps is an option used in connection with Method -> FiniteDifferences
and Method -> MonteCarlo that specifies how many time intervals are used
in the calculation."

ExerciseTimes::usage =
"ExerciseTimes is an option used in connection with Method -> MonteCarlo
for an AmericanOption that specifies the number of times the option
can be exercised. This number must divisible with TimeSteps."

FiniteDifferences::usage = "FiniteDifferences is an option for
Value and Keyfigures specifying that the finite differences method
should be used. GridPoints and TimeSteps can be used to adjust the
accuracy."

PolynomialDegree::usage = "PolynomialDegree is an option used when
valuing an AmericanOption with Method -> MonteCarlo that specifies
the number of polynomials used in the regression in the
least-squares Monte Carlo method."

StandardError::usage = "StandardError is the standard error
returned from Keyfigures when using Method -> MonteCarlo. It is calculated
as the sample standard deviation divided with the square root of the number
of paths."

Begin["`Private`"]

(* Give access to QuantLib`Private` context as LinkPatterns for external
   program are there. Does a more elegant solution exist? *)
AppendTo[$ContextPath, "QuantLib`Private`"]

(* Translation table for passing "enum"-values to C++ *)
OptionTypeNumbers = {
{ CallOption, 0 },
{ PutOption, 1 },
{ Straddle, 2 }
```

```
}

Keyfigures[EuropeanOption[strike_, maturity_, type_],
    BlackScholesModel[underlying_, riskFreeRate_,
                      dividendYield_, volatility_],
    opts___?OptionQ] ^:=
    Module[{meth = Method /. {opts} /. Options[Keyfigures],
            samples = Samples /. {opts} /. Options[Keyfigures],
            antith = AntitheticPaths /. {opts} /. Options[Keyfigures]},
    meth = If[meth === Automatic, Analytic, meth];

    Switch[meth,
    Analytic,
    qlEuropeanOption[SymbolToNumber[type, OptionTypeNumbers],
                         underlying, strike,
                         dividendYield, riskFreeRate,
                                maturity, volatility],
           MonteCarlo,
    If[samples == Automatic, samples = 100000];
    qlEuropeanOptionMC[SymbolToNumber[type, OptionTypeNumbers],
                         underlying, strike,
                         dividendYield, riskFreeRate,
                                maturity, volatility, samples, If[antith,1,0]]
           ]

           ]

Keyfigures[AmericanOption[strike_, maturity_, type_],
    BlackScholesModel[underlying_, riskFreeRate_,
                      dividendYield_, volatility_],
    opts___?OptionQ] ^:=
    Module[{meth = Method /. {opts} /. Options[Keyfigures],
            timeSteps = TimeSteps /. {opts} /. Options[Keyfigures],
            gridPoints = GridPoints /. {opts} /. Options[Keyfigures],
            samples = Samples /. {opts} /. Options[Keyfigures],
            degree = PolynomialDegree /. {opts} /. Options[Keyfigures],
            antithetic = AntitheticPaths /. {opts} /. Options[Keyfigures],
            controlVariate = ControlVariate /. {opts} /. Options[Keyfigures]
  },
    meth = If[meth === Automatic, FiniteDifferences, meth];

    Switch[meth,
           FiniteDifferences,
    qlAmericanOptionFD[SymbolToNumber[type, OptionTypeNumbers],
                       underlying, strike,
                       dividendYield, riskFreeRate,
                              maturity, volatility, timeSteps, gridPoints],
           MonteCarlo,
    If[samples == Automatic, samples = 10000];
    nqAmericanOptionLSM[SymbolToNumber[type, OptionTypeNumbers],
                        underlying, strike,
                        dividendYield, riskFreeRate,
                               maturity, volatility, timeSteps,
                               samples, degree,
                               If[antithetic,1,0], If[controlVariate,1,0]]
           ]

           ]

Keyfigures[AmericanOption[strike_, maturity_, type_],
     SVJDModel[underlying_, riskFreeRate_,
        dividendYield_, volatility_,
```

```
        volatilityOfVolatility_, steadyStateVolatility_,
        meanReversionRate_, correlationUnderlyingVolatility_,
        jumpIntensity_, jumpMean_, jumpStandardDeviation_],
    opts___?OptionQ] ^:=
    Module[{meth = Method /. {opts} /. Options[Keyfigures],
            timeSteps = TimeSteps /. {opts} /. Options[Keyfigures],
            exerciseTimes = ExerciseTimes /. {opts} /. Options[Keyfigures],
            samples = Samples /. {opts} /. Options[Keyfigures],
            degree = PolynomialDegree /. {opts} /. Options[Keyfigures],
            antithetic = AntitheticPaths /. {opts} /. Options[Keyfigures],
            controlVariate = ControlVariate /. {opts} /. Options[Keyfigures]
  },
    meth = If[meth === Automatic, MonteCarlo, meth];

    Switch[meth,
           MonteCarlo,
    If[samples == Automatic, samples = 10000];
    If[exerciseTimes == Automatic, exerciseTimes = timeSteps];
    nqAmericanOptionSVJDLSM[SymbolToNumber[type, OptionTypeNumbers],
                         underlying, strike,
                         dividendYield, riskFreeRate,
                                maturity, volatility, volatilityOfVolatility,
                                steadyStateVolatility, meanReversionRate,
                                correlationUnderlyingVolatility,
                                jumpIntensity, jumpMean,
                                jumpStandardDeviation,
                                timeSteps, exerciseTimes, samples, degree,
                                If[antithetic,1,0], If[controlVariate,1,0]]
                       ]
           ]

ImpliedVolatility[value_, EuropeanOption[strike_, maturity_, type_],
    BlackScholesModel[underlying_, riskFreeRate_,
                      dividendYield_, volatility_],
    opts___?OptionQ] ^:=
    qlImpliedVolatility[value, SymbolToNumber[type, OptionTypeNumbers],
    underlying, strike, dividendYield, riskFreeRate, maturity]

Keyfigures[EuropeanOption[strike_, maturity_, type_],
    SVJDModel[underlying_, riskFreeRate_,
                      dividendYield_, volatility_,
                      volatilityOfVolatility_, steadyStateVolatility_,
                      meanReversionRate_, correlationUnderlyingVolatility_,
                      jumpIntensity_, jumpMean_, jumpStandardDeviation_],
    opts___?OptionQ] ^:=
    Module[{meth = Method /. {opts} /. Options[Keyfigures],
            timeSteps = TimeSteps /. {opts} /. Options[Keyfigures],
            samples = Samples /. {opts} /. Options[Keyfigures],
            antithetic = AntitheticPaths /. {opts} /. Options[Keyfigures],
            controlVariate = ControlVariate /. {opts} /. Options[Keyfigures]
           },
    meth = If[meth === Automatic, Analytic, meth];

    Switch[meth,
    Analytic,
    nqSVJDOption[SymbolToNumber[type, OptionTypeNumbers],
                         underlying, strike,
                         dividendYield, riskFreeRate,
                                maturity, volatility, volatilityOfVolatility,
                                steadyStateVolatility, meanReversionRate,
                                correlationUnderlyingVolatility,
                                jumpIntensity, jumpMean,
```

```
                                jumpStandardDeviation],
    MonteCarlo,
    If[samples == Automatic, samples = 100000];
    nqSVJDOptionMC[SymbolToNumber[type, OptionTypeNumbers],
                          underlying, strike,
                          dividendYield, riskFreeRate,
                                 maturity, volatility, volatilityOfVolatility,
                                 steadyStateVolatility, meanReversionRate,
                                 correlationUnderlyingVolatility,
                                 jumpIntensity, jumpMean,
                                 jumpStandardDeviation,
                                 timeSteps, samples,
                                 If[antithetic,1,0], If[controlVariate,1,0]]
          ]
          ]

PDF[EuropeanOption[strike_, maturity_, type_],
    SVJDModel[underlying_, riskFreeRate_,
              dividendYield_, volatility_,
              volatilityOfVolatility_, steadyStateVolatility_,
              meanReversionRate_, correlationUnderlyingVolatility_,
              jumpIntensity_, jumpMean_, jumpStandardDeviation_],
    x_,
    opts___?OptionQ] ^:=

    nqSVJDPDF[x, SymbolToNumber[type, OptionTypeNumbers],
      underlying, strike,
      dividendYield, riskFreeRate,
      maturity, volatility, volatilityOfVolatility,
      steadyStateVolatility, meanReversionRate,
      correlationUnderlyingVolatility,
      jumpIntensity, jumpMean,
      jumpStandardDeviation
    ]

CDFN[x_] := (1 + Erf[x/Sqrt[2]])/2

PDFN[x_] := 1 / Sqrt[2 Pi] * Exp[-x^2 / 2]

PDFN[m_,s_,x_] := 1/(E^((-m + x)^2/(2*s^2))*Sqrt[2*Pi]*s)

PDF[EuropeanOption[strike_, maturity_, type_],
    BlackScholesModel[underlying_, riskFreeRate_, dividendYield_,
      volatility_], x_, opts___?OptionQ] ^:=
  PDFN[(riskFreeRate -
             dividendYield - (volatility^2) / 2)maturity,
      volatility Sqrt[maturity], x]

Options[Keyfigures] := { Method -> Automatic, Samples -> Automatic,
                         AntitheticPaths -> False, ControlVariate -> False,
                         GridPoints -> 100,
                         TimeSteps -> 100, ExerciseTimes -> Automatic,
                         PolynomialDegree -> 2 }

Value[x___] := Value /. Keyfigures[x]

(* Formel *)

TranslateSVJDModel[model_] :=
model /.
  SVJDModel[underlying_, riskFreeRate_, dividendYield_, volatility_,
      volatilityOfVolatility_, steadyStateVolatility_, meanReversionRate_,
```

```
      correlationUnderlyingVolatility_,  jumpIntensity_, jumpMean_,
      jumpStandardDeviation_] :>
    {S = underlying,
     r = riskFreeRate, rf = dividendYield,
     V = volatility^2,
     sigmav = volatilityOfVolatility,
     alpha = (steadyStateVolatility^2)*meanReversionRate,
     betastar = meanReversionRate,
     rho = correlationUnderlyingVolatility,
     lambdastar = jumpIntensity,
     kmeanstar = jumpMean,
     delta = jumpStandardDeviation}

TranslateInstrument[instr_] :=
    instr /.
    EuropeanOption[strike_, residualTime_, typ_] :>
    {K = strike, T = residualTime, type = typ}

(*
Keyfigures[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    Switch[type,
    CallOption,
        {Value ->
         S Exp[-rf T] P[1, Log[K/S], V, T, K]
         - Exp[-r T] K P[2, Log[K/S], V, T, K]},
    PutOption,
        {Value ->
         S Exp[-rf T] (P[1, Log[K/S], V, T, K] - 1)
         - Exp[-r T] K (P[2, Log[K/S], V, T, K] - 1)}
           ]
           ]
*)

P[j_, x_, V_, T_, K_] := 0.5 + (1/Pi) NIntegrate[integr[j, u, x], {u, 0, 300}]

integr[j_, u_, x_] := Im[F[j, I u] Exp[-I u x]] / u

F[j_, u_] := Exp[CC[j, u] + DD[j, u] V + EE[j, u]]

CC[j_, u_] := ((r - rf - lambdastar kmeanstar) u T
                - ((alpha T) / sigmav^2) (rho sigmav u - beta[j] - gamma[j, u])
                - ((2 alpha) / sigmav^2) *
                  Log[1 + 0.5 (rho sigmav u - beta[j] - gamma[j, u]) *
                          ((1 - Exp[gamma[j, u] T]) / gamma[j, u])])

DD[j_, u_] :=  -2 (my[j] u + 0.5 u^2) /
    (rho sigmav u - beta[j] + gamma[j, u] *
        (1 + Exp[gamma[j, u] T]) / (1 - Exp[gamma[j, u] T]))

EE[j_, u_] := lambdastar T (1 + kmeanstar)^(my[j] + 0.5) *
                ((((1 + kmeanstar)^u) Exp[delta^2 (my[j] u + u^2 / 2)]) - 1)

gamma[j_, u_] :=
             Sqrt[(rho sigmav u - beta[j])^2 - 2 sigmav^2 (my[j] u + 0.5 u^2)]

my[1] = 0.5; my[2] = -0.5;
```

```
beta[1] := betastar - rho sigmav; beta[2] := betastar;

(* Nielsen (1999) p. 49 *)
R[u_] := CC[2, u] + DD[2, u] V + EE[2, u]

Mean[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    D[R[u], u] /. u -> 0
    ]

Variance[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    D[R[u], u, u] /. u -> 0
    ]

Skewness[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    (D[R[u], u, u, u] / (D[R[u], u, u]^(3/2)) ) /. u -> 0
    ]

KurtosisExcess[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    (D[R[u], u, u, u, u] / (D[R[u], u, u]^2)) /. u -> 0
    ]

BlackScholesVolatility[EuropeanOption[instrpar__],
    SVJDModel[modelpar__],
    opts___?OptionQ] ^:=
    Module[{},
    TranslateSVJDModel[SVJDModel[modelpar]];
    TranslateInstrument[EuropeanOption[instrpar]];
    Sqrt[(D[R[u], u, u] /. u -> 0) / T]
    ]

BlackScholesModel[EuropeanOption[instrpar__],
    model_SVJDModel,
    opts___?OptionQ] ^:= BlackScholesModel[model[[1]], model[[2]],
                  model[[3]],
                  BlackScholesVolatility[EuropeanOption[instrpar],
                                         model]]

End[]

EndPackage[]
```

E.2 QuantLibMma.tm

(Uddrag)

```
:Begin:
:Function:       nqAmericanOptionSVJDLSM
:Pattern:        nqAmericanOptionSVJDLSM[type_Integer, underlying_?NumberQ,
    strike_?NumberQ, dividendYield_?NumberQ, riskFreeRate_?NumberQ,
    maturity_?NumberQ, volatility_?NumberQ, volatilityOfVolatility_?NumberQ,
    steadyStateVolatility_?NumberQ, meanReversionRate_?NumberQ,
    correlationUnderlyingVolatility_?NumberQ, jumpIntensity_?NumberQ,
    jumpMean_?NumberQ, jumpStandardDeviation_?NumberQ, timeSteps_?NumberQ,
    exerciseTimes_?NumberQ, samples_?NumberQ, degree_Integer,
    antithetic_Integer, controlVariate_Integer]
:Arguments:      { type, underlying, strike, dividendYield, riskFreeRate,
    maturity, volatility, volatilityOfVolatility, steadyStateVolatility,
    meanReversionRate, correlationUnderlyingVolatility, jumpIntensity,
    jumpMean, jumpStandardDeviation, timeSteps, exerciseTimes, samples,
    degree, antithetic, controlVariate }
:ArgumentTypes:  { Integer, Real, Real, Real, Real, Real, Real, Real, Real,
    Real, Real, Real, Real, Real, Real, Real, Real, Integer, Integer, Integer }
:ReturnType:     Manual
:End:
```

E.3 mmaoptions.cpp

(Uddrag)

```
void nqAmericanOptionSVJDLSM(int type, double underlying, double strike,
                         double dividendYield, double riskFreeRate,
                         double maturity, double volatility,
                         double volatilityOfVolatility,
                         double steadyStateVolatility, double meanReversionRate,
                         double correlationUnderlyingVolatility,
                         double jumpIntensity, double jumpMean,
                         double jumpStandardDeviation,
                         double timeSteps, double exerciseTimes,
                         double samples, int degree,
                         int antithetic, int controlVariate)
{
    try {
        LSMSVJDEngine engine(samples, timeSteps, exerciseTimes, degree,
                             antithetic, controlVariate);

        // downcast the Arguments struct to the appropriate type...
        SVJDArguments* underArgs =
            dynamic_cast<SVJDArguments*>(engine.arguments());
        QL_ENSURE(underArgs != 0, "dynamic_cast failed");
        // ... and set the values
        underArgs->type = (Option::Type)type;
        underArgs->underlying = underlying;
        underArgs->strike = strike;
        underArgs->dividendYield = dividendYield;
        underArgs->riskFreeRate = riskFreeRate;
        underArgs->maturity = maturity;
        underArgs->volatility = volatility;

        underArgs->volatilityOfVolatility = volatilityOfVolatility;
```

```
        underArgs->steadyStateVolatility = steadyStateVolatility;
        underArgs->meanReversionRate = meanReversionRate;
        underArgs->correlationUnderlyingVolatility
            = correlationUnderlyingVolatility;

        underArgs->jumpIntensity = jumpIntensity;
        underArgs->jumpMean = jumpMean;
        underArgs->jumpStandardDeviation = jumpStandardDeviation;

        engine.calculate();

        const OptionValue* results =
            dynamic_cast<const OptionValue*>(engine.results());
        QL_ENSURE(results != 0, "dynamic_cast failed");

        MLPutFunction(stdlink, "List", 2);
        QLMLPutRule(stdlink, "Value", results->value);
        QLMLPutRule(stdlink, "StandardError", results->errorEstimate);
    } QLML_HANDLE_EXCEPTIONS
}
```

Litteratur

Ametrano, F. & Ballabio, L. (2003). Quantlib - a free/open-source library for quantitative finance. http://quantlib.org/.

Andersen, T., Benzoni, L. & Lund, J. (2002). An empirical investigation of continuous-time equity return models, *Journal of Finance* **57**(3): 1239–1284.

Andreasen, J. (2003). Derivatives - the view from trenches, Foredrag på Institut for Matematiske Fag på Københavns Universitet. http://www.act.ku.dk/colloquium/2003/jandreasen.pdf.

Bakshi, G., Cao, C. & Chen, Z. (1997). Empirical performance of alternative option pricing models, *Journal of Finance* **52**(5): 2003–2049. http://links.jstor.org/sici?sici=0022-1082(199712)52:5O

Barone-Adesi, G. & Whaley, R. E. (1987). Effecient analytic approximation of American option values, *Journal of Finance* **42**: 301–320.

Basso, A., Nardon, M. & Pianca, P. (2002). Optimal exercise of american options. http://amases2002.univr.it/amases/abstracts/Basso.pdf.

Bates, D. S. (1988). Pricing options on jump-diffusion processes. http://www.biz.uiowa.edu/faculty/dbates/papers/chapter3.pdf.

Bates, D. S. (1991). The crash of '87: Was it expected? The evidence from options markets, *Journal of Finance* **46**: 1009–1044.

Bates, D. S. (1996). Jumps and stochastic volatility: Exchange rate processes implicit in Deutsche Mark options, *Review of Financial Studies* **9**(1): 69–107. http://rfs.oupjournals.org/cgi/content/abstract/9/1/69.

Bates, D. S. (2003). Maximum likelihood estimates of latent affine processes. http://www.biz.uiowa.edu/faculty/dbates/papers/bates_03.pdf.

Baxter, M. & Rennie, A. (1996). *Financial calculus: An introduction to derivative pricing*, 1st edn, Cambridge University Press.

Björk, T. (1998). *Arbitrage Theory in Continuous Time*, 1st edn, Oxford University Press.

Black, F. & Scholes, M. (1973). The pricing of options and corporate liabilities, *Journal of Political Economy* **81**: 637–654.

Boyle, P., Broadie, M. & Glasserman, P. (1997). Monte Carlo methods for security pricing, *Journal of Economic Dynamics and Control* **21**: 1267–1321.

Boyle, P. P. (1977). Options: A Monte Carlo approach, *Journal of Financial Economics* **4**(3): 323–338. http://www.sciencedirect.com/science/article/B6VBX-458X2BW-Y/2/dfa6a2b4ceb42aad63a2df40bbd8a4e2.

Broadie, M. & Glasserman, P. (1997). Pricing American style securities using simulation, *Journal of Economic Dynamics and Control* **21**: 1323–1353.

Clément, E., Lamberton, D. & Protter, P. (2002). An analysis of a least squares regression method for American option pricing, *Finance and Stochastics* **6**(4): 449–471. http://econpapers.hhs.se/article/sprfinsto/v_3A6_3Ay_3A2002_3Ai_3A4_3Ap_3A449-471.htm.

Cox, J. C. & Ross, S. A. (1976). The valuation of options for alternative stochastic processes, *Journal of Financial Economics* **3**(1-2): 145–166. http://www.sciencedirect.com/science/article/B6VBX-45N4YVB-7/2/8d23a23f6cd0f4d9ef11b3fc7992515c.

Cox, J., Ross, S. & Rubinstein, M. (1979). Option pricing: A simplified approach, *Journal of Financial Economics* **7**: 229–263.

Cyganowski, S., Grüne, L. & Kloeden, P. E. (2002). Maple for jump-diffusion stochastic differential equations in finance, *in* S. S. Nielsen (ed.), *Programming Languages and Systems in Computational Economics and Finance*, Kluwer, Dordrecht, pp. 441–460. http://www.uni-bayreuth.de/departments/math/ lgruene/papers/jumpfin.html.

Duffie, D. (2001). *Dynamic Asset Pricing Theory*, 3rd edn, Princeton University Press.

Dupire, B. (1994). Pricing with a smile, *RISK Magazine* **7**(1): 18–20. http://www.reech.com/company/research.papers/downloads/smile.doc.

Egloff, D. & Min-Oo, M. (2002). Convergence of Monte Carlo algorithms for pricing American options, Department of Mathematics and Statistics, McMaster University, Ontario, Canada. http://www.math.mcmaster.ca/minoo/mypapers/em.pdf.

Eskildsen, B. (2002). *Monte Carlo simulation*, cand.merc.(mat.) hovedopgave, Handelshøjskolen i København, Institut for Finansiering. http://hermescat.lib.cbs.dk/is/www/full-sh.asp?fn=x656194366.

Frühwirth-Schnatter, S. & Sögner, L. (2003). Bayesian estimation of the heston stochastic volatility model, Department of Applied Statistics, Johannes Kepler University, Linz.

Gerald, C. F. & Wheatley, P. O. (1999). *Applied Numerical Analysis*, 6th edn, Addison-Wesley, Reading, Massachusetts.

Heston, S. L. (1993). A closed form solution for options with stochastic volatility with applications to bond and currency options, *Review of Financial Studies* **6**: 327–343. http://rfs.oupjournals.org/cgi/content/abstract/6/2/327.

Heston, S. L. & Nandi, S. (2000). A closed-form GARCH option valuation model, *Review of Financial Studies* **13**(3): 585–625. http://rfs.oupjournals.org/cgi/content/abstract/13/3/585.

Hull, J. C. (2000a). *Options, Futures, and Other Derivatives*, 4th edn, Prentice Hall, Inc.

Hull, J. C. (2000b). *Opzioni, Futures e altri derivati*, seconda edn, Il Sole 24 ORE, Milano. Italiensk oversættelse ved Emilio Barone af "Options, Futures, and Other Derivatives", 4th edition.

Hull, J. & White, A. (1987). The pricing of options on assets with stochastic volatilities, *Journal of Finance* **42**(2): 281–300. http://links.jstor.org/sici?sici=0022-10823E2.0.CO

Jorion, P. (1988). On jump processes in the foreign exchange and stock markets, *Review of Financial Studies* **1**(4): 427–445. http://rfs.oupjournals.org/cgi/content/abstract/1/4/427.

Ju, N. (1998). Pricing an American option by approximating its early exercise boundary as a multipiece exponential function, *Review of Financial Studies* **11**(3): 627–646. http://rfs.oupjournals.org/cgi/content/abstract/11/3/627.

Kluge, T. (2002). *Pricing derivatives in stochastic volatility models using the finite difference method*, Diploma thesis, Technische Universität Chemnitz, Fakultät für Mathematik. http://archiv.tu-chemnitz.de/pub/2003/0008/data/.

Koenig, A. & Moo, B. (2000). *Accelerated C++*, 1st edn, Addison-Wesley. http://www.acceleratedcpp.com/.

Lewis, A. (2000). *Option Valuation under Stochastic Volatility*, 1st edn, Finance Press.

Lipton, A. (2002). The vol smile problem, *RISK Magazine* **15**(2): 61–65.

Longstaff, F. A. & Schwartz, E. S. (2001). Valuing American options by simulation: A simple least-squares approach, *Review of Financial Studies* **14**(1): 113–147. http://rfs.oupjournals.org/cgi/content/abstract/14/1/113.

Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous, *Journal of Financial Economics* **3**(1-2): 125–144. http://www.sciencedirect.com/science/article/B6VBX-45N4YVB-6/2/0d8372fda5ec0938340ed1838965b45e.

Moreno, M. & Navas, J. F. (2001). On the robustness of least-squares Monte Carlo for pricing American derivatives, Department of Economics and Business, Universitat Pompeu Fabra. http://econpapers.hhs.se/paper/upfupfgen/543.htm.

Morera, F. (2003). *Opzioni Americane: l'algoritmo di Longstaff-Schwartz per la determinazione del prezzo*, Tesi di laurea, Università degli Studi Roma Tre. http://www.mat.uniroma3.it/didatticacds/studenti/laureati/morera/sintesi_morera.pdf.

Nielsen, J. H. (1999). *Lukkede udtryk for optionspriser, fordelinger og volatilitetssmil i modeller med stokastisk volatilitet og spring*, cand.merc.(mat.) hovedopgave, Handelshøjskolen i København, Institut for Finansiering. http://hermescat.lib.cbs.dk/is/www/full-sh.asp?fn=x648036153.

O'Neil, C. (2002). Gauss-Kronrod integration – theory and Java applet. http://mathcssun1.emporia.edu/~oneilcat/ExperimentApplet3/ExperimentApplet3.html.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge. http://www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf.html.

Rasmussen, N. S. (2002). Improving the least-squares monte-carlo approach. http://ideas.repec.org/p/hhb/aarfin/2002_018.html.

Sepp, A. (2003). Fourier transform for option pricing under affine jump-diffusions: An overview. http://hot.ee/seppar/papers/stochjumpvols.pdf.

Srikant, M. (1998). *Option pricing with stochastic volatility*, Master's thesis, National University of Singapore, Department of Computational Science. http://www.srikant.org/thesis/.

Stein, E. M. & Stein, J. C. (1991). Stock price distributions with stochastic volatility: An analytic approach, *Review of Financial Studies* **4**(4): 727–752. http://rfs.oupjournals.org/cgi/content/abstract/4/4/727.

Stentoft, L. (2001). Assessing the least squares Monte-Carlo approach to American option valuation, CAF Working Paper Series, No. 90, Økonomisk Institut, Århus Universitet. http://www.econ.au.dk/vip_htm/lstentoft/papers/wp-90.pdf.

Stentoft, L. (2002). Convergence of the least squares Monte-Carlo approach to American option valuation, CAF Working Paper Series, No. 113, Økonomisk Institut, Århus Universitet. http://www.econ.au.dk/vip_htm/lstentoft/papers/wp-113.pdf.

Tian, T. & Burrage, K. (2003). Accuracy issues of Monte-Carlo methods for valuing American options, *The Anziam Journal* **44**: C739–C758. http://anziamj.austms.org.aue/V44/CTAC2001/Tian/home.htm.

Tzavalis, E. & Want, S. (2003). Pricing American options under stochastic volatility: A new method using Chebyshev polynomials to approximate the early exercise boundary, Working Paper No. 488, Queen Mary, University of London. http://www.econ.qmw.ac.uk/papers/wp488.htm.

UnRisk (2003). UnRisk pricing engine – fast, accurate pricing of derivatives in Mathematica. http://www.unriskderivatives.com/.

Weisstein, E. (2003). Mathworld: Eric Weisstein's world of mathematics. http://mathworld.wolfram.com/.

Wilmott, P. (1998). *Derivatives: The Theory and Practice of Financial Engineering*, 1st edn, John Wiley & Sons Ltd.

Wolfram, S. (1999). *The Mathematica Book*, 4th edn, Cambridge University Press. http://documents.wolfram.com/indexbook.html.