STOCHASTIC VOLATILITY MODELS: CALIBRATION, PRICING AND HEDGING

by

Warrick Poklewski-Koziell

Programme in Advanced Mathematics of Finance

School of Computational and Applied Mathematics

University of the Witwatersrand,

Private Bag-3, Wits-2050, Johannesburg

South Africa

May 2012

A Dissertation Submitted for the Degree of Master of Science

ABSTRACT

Stochastic volatility models have long provided a popular alternative to the Black-Scholes-Merton framework. They provide, in a self-consistent way, an explanation for the presence of implied volatility smiles/skews seen in practice. Incorporating jumps into the stochastic volatility framework gives further freedom to financial mathematicians to fit both the short and long end of the implied volatility surface. We present three stochastic volatility models here - the Heston model, the Bates model and the SVJJ model. The latter two models incorporate jumps in the stock price process and, in the case of the SVJJ model, jumps in the volatility process. We analyse the effects that the different model parameters have on the implied volatility surface as well as the returns distribution. We also present pricing techniques for determining vanilla European option prices under the dynamics of the three models. These include the fast Fourier transform (FFT) framework of Carr and Madan as well as two Monte Carlo pricing methods. Making use of the FFT pricing framework, we present calibration techniques for fitting the models to option data. Specifically, we examine the use of the genetic algorithm, adaptive simulated annealing and a MATLAB optimisation routine for fitting the models to option data via a least-squares calibration routine. We favour the genetic algorithm and make use of it in fitting the three models to ALSI and S&P 500 option data. The last section of the dissertation provides hedging techniques for the models via the calculation of option price sensitivities. We find that a delta, vega and gamma hedging scheme provides the best results for the Heston model. The inclusion of jumps in the stock price and volatility processes, however, worsens the performance of this scheme. MATLAB code for some of the routines implemented is provided in the appendix.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Diane Wilcox, for suggesting a thoroughly interesting research topic. Her assistance and guidance with the subject matter of this dissertation helped me immeasurably. Furthermore, I am very grateful to the National Research Foundation[1] who provided me with a generous bursary to aid me in my studies. My family and Heather also provided me with continuous support and encouragement for which I am extremely grateful.

[1]The opinions expressed in this document do not necessarily represent those of the National Research Foundation.

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

[illegible]

Warrick Poklewski-Koziell

May 2012

Contents

List of Figures

Chapter 1

Introduction

Financial mathematicians continuously seek to find stock price models that best explain observed stock price dynamics. The most influential of these models has been the Black-Scholes-Merton model (Black and Scholes [7]; Merton [42]) that was formulated in the early 1970's by the three men after whom the model is named. Much of the popularity of the model came about as a result of its simplicity and the ease with which it provides pricing and hedging solutions for option contracts. This simplicity, however, has many drawbacks. Notably, the model enforces constant stock volatilities and permits only log-normally distributed asset returns. Such dynamics have been shown to be inconsistent with observations in actual financial markets. Market crashes have occurred far more frequently than anticipated by these dynamics. One of the most notable crashes was that of 1987, which led to the emergence of higher implied volatilities for in and out-of-the-money options than at-the-money options. This was due to an increased awareness that the model was incapable of describing the tail activities of stock price probability distributions. Such observations have lead some financial experts to investigate certain stylised facts in financial markets — that stock returns exhibit excess kurtosis and skewness, that volatility is non-constant and tends to cluster and, increasingly, that many markets show signs of jumps in stock prices (and even in the stock price volatility). This has lead to the exploration of stock price models that exhibit such characteristics.

In the past two decades, much research has centred around incorporating stochastic volatility as well as jump components into stock price models. Works by Bakshi et al. [3], Bates [5], Broadie et al. [10], Duffie et al. [22], Gatheral [25] and Heston [28] — to name but a few — have explored the merits and hindrances of using such models to explain stock price dynamics. These models are complex and do not always yield closed form solutions for option pricing. They are, however, very useful in allowing mathematicians to fit both the short and long end of the implied volatility surface. They give a realistic explanation for the presence of the implied volatility skew and are more robust in their descriptions of stock price and volatility movements than the Black-Scholes model is.

In this dissertation, we examine three stochastic volatility models, namely the Heston model, the Bates model and a stochastic volatility model with jumps in both the stock price and variance processes (SVJJ model). Each model is an extension of the previous one, starting with the Heston model, which comprises a stock price process similar to that of the Black-Scholes price process, where the constant volatility term has been replaced by a stochastic term evolving according to a mean-reverting diffusion process. The Bates model then allows for the inclusion of a jump term in the stock price process, while the SVJJ model also includes a jump term in the volatility process. Heston [28] saw the need to devise a stochastic volatility model capable of explaining the skewness in the distributions of stock price returns, as well as the empirically observed implied volatility skew. At the same time, he desired a model that exhibited a "closed-form" (i.e. an integral representation) method for pricing vanilla European options and appealed to Fourier transform techniques for this purpose. This led to the formulation of the Heston model, which today is still extremely popular due to its ability to replicate many observed market phenomena, as well as the ease with which vanilla option prices can be computed under the dynamics of the model. Bates [5] extended this model due to his observation that it was unable to fully explain the implied volatility smile resulting from excess kurtosis in returns distributions. He argued that adding jumps to the price process of the model made it more capable of this task and thus more empirically consistent. In his analysis, he tested his model on Deutsche Mark options data over the period 1984 to 1991 and found evidence supporting the need for jumps in the stock price process of the model.

The paper by Duffie et al. [22] provides a comprehensive treatment on affine jump-diffusion processes. A model that arises naturally from their analysis is the SVJJ model and they compare the performance of this model to the Bates and the Heston models. Calibrating the three models to option data, the authors find that the SVJJ model provides the best fit to the data. Other works of particular interest to us are those by Bakshi et al. [3], Broadie et al. [10] and Gatheral [25]. All three give insight into the addition of jump terms to the stock price and variance processes and find, to varying degrees, that the inclusion of jumps is necessary for stochastic volatility models to comply with market observed phenomena.

The rest of this dissertation is structured as follows. In Chapter 2, we review the three stochastic volatility models and show some of the effects that the models' parameters have on stock price returns as well as on the implied volatility surface. Chapter 3 considers pricing methods for the three models. More specifically, we examine the application of the fast Fourier transform to vanilla European option pricing under these models. The framework that we follow is that laid out by Carr and Madan [13]. We also consider the paper by Broadie and Kaya [11], which provides a detailed analysis of two Monte Carlo methods that can be applied to the models. Chapter 4 examines the calibration of the models to synthetic as well as to market data. Specifically, we calibrate the models to option price data from the South African All Share Index (ALSI) as well as the S&P 500 index. Options on the ALSI are futures options and so our modelling of these options in a stochastic volatility setting amounts to assuming that the dynamics of the underlying forward price are described by one of the three stochastic volatility models analysed in this dissertation. In this chapter, we compare three calibration methods based on three optimisation routines, namely the genetic algorithm, adaptive simulated annealing and a non-linear least squares method, lsqnonlin, available with the MATLAB software. Finally, chapter 5 examines hedging methods that can be applied to vanilla call options whose underlying assets follow the dynamics of the Heston, Bates and SVJJ models. Specifically, we focus on hedging methods using option price sensitivities to the underlying parameters. Such an analysis would also be useful in the setting of hedging methods for exotic options.

The purpose of this document is to provide a thorough overview of the three models and pricing, calibration and hedging techniques that can be used to implement the models in practical settings. As such, the dissertation is aimed more at practitioners than mathematicians and a major emphasis of the work is on the numerical implementation of the numerous techniques. We intend that the subject matter contained here will give readers a good understanding of the dynamics of the different models as well as a consistent framework for approaching the core issues behind the implementation of these models

MATLAB was used extensively as a means of simulating the pricing, calibration and hedging routines presented in this dissertation. Some of the code for these routines is presented in the appendix. All results were obtained via implementation of code in MATLAB 2010b (running in Microsoft Windows 7), on a desktop supercomputer incorporating an Intel Core i7-970 3.2GHz hexacore CPU, 24GB DDR3 RAM and a C2050 Tesla GPU.

Finally, it is important to note some topics that are beyond the scope of this dissertation. Investigations into these topics in further research reports would provide valuable extensions to our work. We have not considered no-arbitrage bounds for the market implied volatility surfaces in this project. In practice, these are very important to ensure that the calibrated surfaces are free from arbitrage. Such bounds are usually set up to ensure that call spreads, butterfly spreads and calendar spreads cannot be constructed off the surfaces to produce arbitrage strategies. An extension of the subject matter in Chapter 4 would be to explore the literature on such no-arbitrage bounds and thus further the investigation into calibrating stochastic volatility models to South African implied volatility data. A particularly useful paper for such an investigation is by Carr and Madan [14]. Moreover, we have not considered the temporal stability of option price parameters, nor have we considered the fitting of models to historical data on the underlying asset. Instead, we have examined the calibration of stochastic volatility models to implied volatility surfaces at single points in time. Obviously, this only gives us an idea of the (risk-neutral) dynamics of the underlying asset process at that time. A valuable extension to this approach would be to evaluate how model parameters change over time and to examine risk premia in the market. Lastly, we have not explored other methods for dealing with non-constant volatility. Such alternatives include local volatility models (see, for example, Gatheral [25]) and GARCH type models (see, for example, Pakel et al. [46]). These alternatives are explored extensively in the financial mathematics literature and provide different approaches for dealing with the volatility surface in option markets.

Chapter 2

Stochastic Volatility Models

2.1 The Heston Model

The Heston model was introduced in the 1993 paper by Steven Heston [28]. The model specifies the following risk-neutral stock price dynamics:

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{V_t} S_t d\widetilde{W}_t^{(1)} && (2.1) \\ dV_t &= \kappa\left(\theta - V_t\right) dt + \sigma_v \sqrt{V_t} d\widetilde{W}_t^{(2)} && (2.2) \\ d\widetilde{W}_t^{(1)} d\widetilde{W}_t^{(2)} &= \rho dt, && (2.3) \end{aligned}$$

where r is the risk-neutral rate of return, and $\widetilde{W}_t^{(1)}$ and $\widetilde{W}_t^{(2)}$ are two correlated Brownian motions under the risk-neutral measure. Here we consider only the risk-neutral dynamics of the stock price process. In chapter 5, we will explore the existence of equivalent martingale measures and examine the transformation from real-world dynamics to those under the risk-neutral measure. From the specification above, we can see that the Heston model is a pure diffusion model — it does not permit jumps in the stock price or the variance processes. The stock price process is similar to that specified under the Black-Scholes model. Here, however, the constant volatility term that appears in the Black-Scholes model has been replaced by a stochastic one which follows the same mean-reverting square root process used by Cox et al. [21] in their famous interest rate model. Figure 2.1 gives an example of ten Heston stock price paths.

The main parameters of interest in the Heston Model are κ, ρ and σ_v. The rate at which the variance process reverts to its long run average θ is given by the parameter κ. High values of κ essentially turn the stochastic volatility into a time dependent deterministic one, since any deviations in the variance from θ are immediately pulled back. The parameter ρ affects the skewness of the returns distribution (see Figure 2.2) and hence the skewness in the implied volatility smile. Negative values of ρ induce a negative skewness in the returns distribution since lower returns will be accompanied by higher volatility which will stretch the left tail of the distribution. The reverse is true for positive correlation. The parameter σ_v affects the kurtosis of the returns distribution and hence the steepness of the implied volatility smile (see Figure 2.3). Large values of σ_v cause more fluctuation in the volatility process (provided κ is not too large) and hence stretch the tails of the returns distribution in both directions. Figure 2.4 shows the effects that ρ and σ_v have on the implied volatility surface.

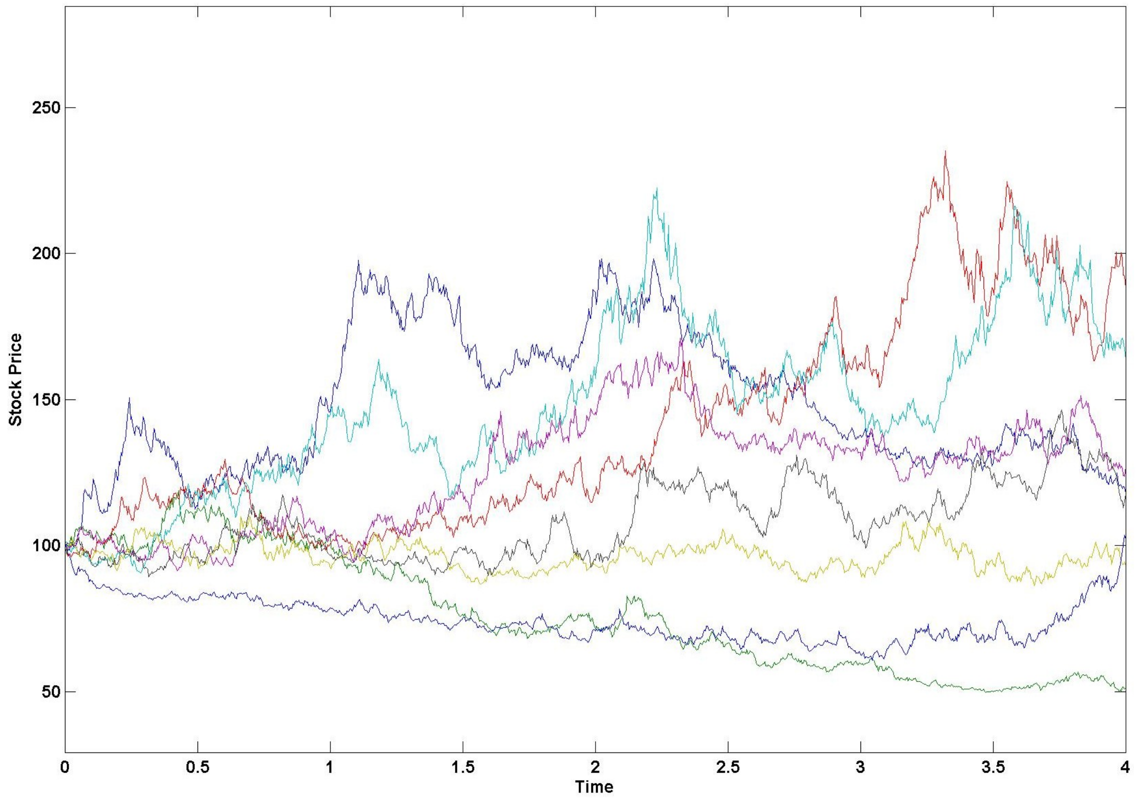

Figure 2.1 Sample Heston stock price paths for $S_0 = 100$, $\kappa = 1.5$, $\theta = V_0 = 0.04$, $\sigma_v = 0.2$, $\rho = 0.8$. The plot was produced using Euler Monte Carlo methods with 1000 time steps.

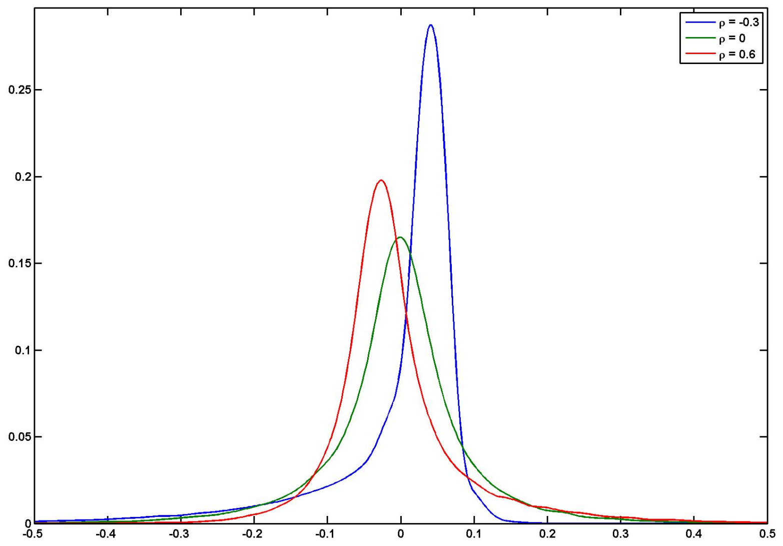

Figure 2.2 The effect of ρ on the distribution of stock price returns under the Heston model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. We can see how negative values of ρ induce negative skewness in the stock price returns distribution and vice versa.

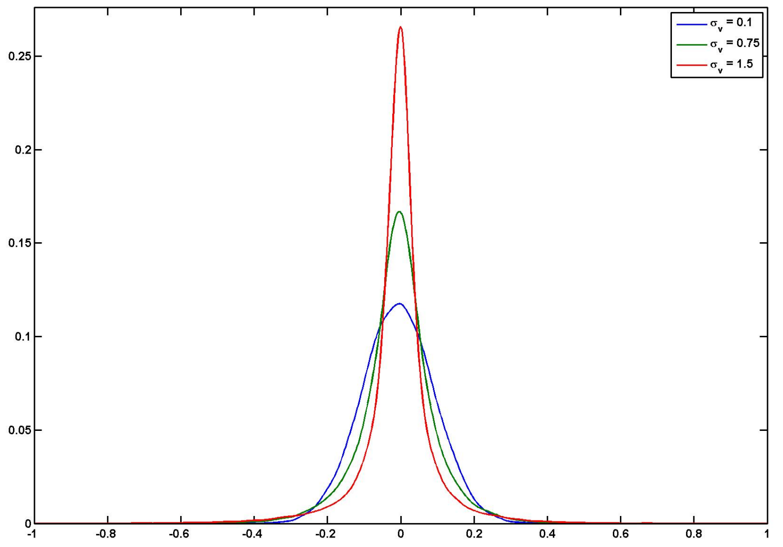

Figure 2.3 The effect of σ_v on the distribution of stock price returns under the Heston model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. We can see how larger values of σ_v increase the kurtosis in the returns distribution.

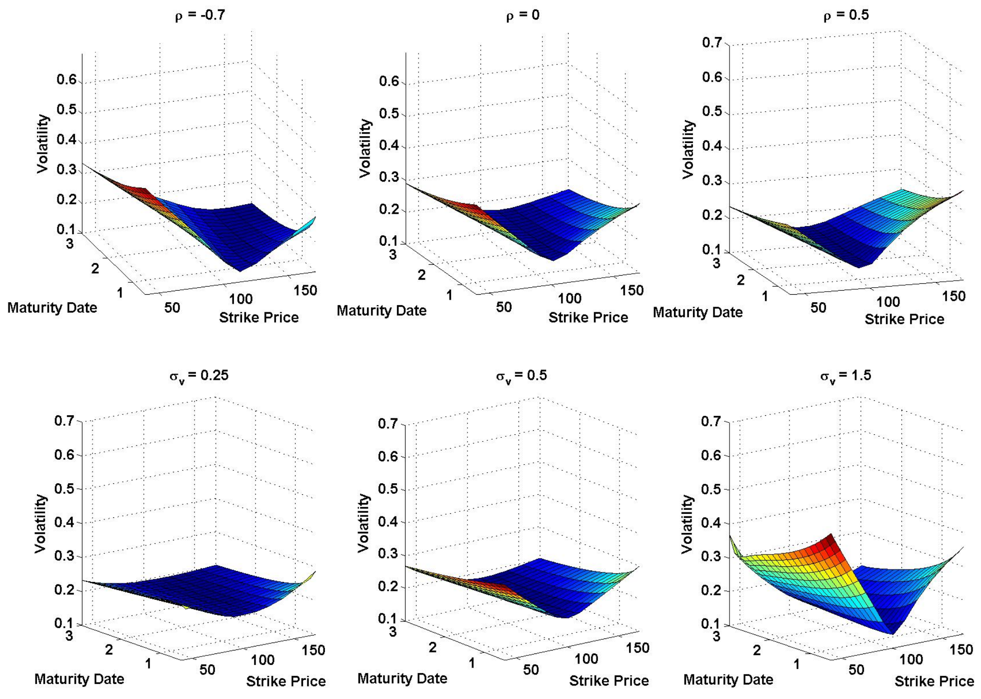

Figure 2.4 The effect of ρ and σ_v on the Heston implied volatility surface. The figure was produced using FFT pricing techniques. The top three plots show how the skewness in the volatility surface changes for positive and negative values of ρ. The bottom three plots show how the steepness increases for increasing values of σ_v.

2.2 The Bates Model

The Bates model was introduced by David Bates [5] in his 1996 paper and is an extension of the Heston model to include jumps in the stock price process. The model has the following risk-neutral dynamics defining the evolution of S_t:

$$\begin{aligned} dS_t &= (r - \lambda\mu_J)\, S_t dt + \sqrt{V_t} S_t d\widetilde{W}_t^{(1)} + J S_t d\widetilde{N}_t & (2.4)\\ dV_t &= \kappa\,(\theta - V_t)\, dt + \sigma_v \sqrt{V_t} d\widetilde{W}_t^{(2)} & (2.5)\\ d\widetilde{W}_t^{(1)} d\widetilde{W}_t^{(2)} &= \rho dt. & (2.6) \end{aligned}$$

Appendix A gives some intuition for the form of the above stock price process. The

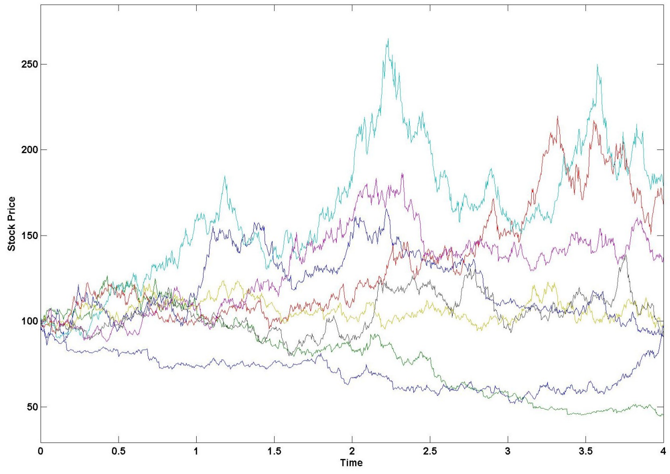

Figure 2.5 Sample Bates stock price paths for $S_0 = 100$, $\kappa = 1.5$, $\theta = V_0 = 0.04$, $\sigma_v = 0.2$, $\rho = 0.8$, $\lambda = 3$, $\mu_S = -0.05$, $\sigma_S = 0.0001$. The plot was produced using Euler Monte Carlo methods with 1000 time steps.

volatility process V_t is the same as that in the Heston model and the driving Brownian motions in the two processes have an instantaneous correlation equal to ρ. The process $\widetilde{N}_t$ represents a Poisson process under the risk neutral measure, with jump intensity λ. It is independent of the two Brownian motions in the stock price and variance processes. The percentage jump size of the stock price is dictated by the random variable J, with

$$1 + J \sim \text{log-normal}\left(\mu_S, \sigma_S^2\right),$$

where the relationship between μ_S and μ_J is given by

$$\mu_J = \exp\left\{\mu_S + \frac{\sigma_S^2}{2}\right\} - 1.$$

Figure 2.5 gives an example of ten Bates stock price paths. It is apparent that adding a jump term to the stock price process produces more volatile price movements than those displayed by the Heston model.

Since the Bates model is an extension of the Heston model, the parameters κ, ρ and σ_v have the same effect on the returns distribution and implied volatility surface as they do in the Heston model. In addition to these, the parameters defining the jump term in the stock-price process are of particular interest. The parameter μ_S influences the skewness of the stock price returns distribution, as can be seen in Figure 2.6. Positive values of μ_S lead to a positive skew in the distribution of returns. Negative values of μ_S have the opposite effect. The parameter σ_S affects the kurtosis of the stock price returns distribution. Larger values of σ_S increase the variance of stock price jump sizes and hence increase the kurtosis of the returns distribution. The effect of σ_S on the returns distribution can be seen in Figure 2.7. The Poisson process intensity parameter λ dictates how frequently jumps occur and its effect on the distribution of stock price returns can be seen in Figure 2.8. Larger values of λ increase the occurrence of jumps in the stock price process and this raises the overall level of volatility in the stock price. As a result, λ affects the kurtosis in the returns distribution. Figures 2.9 and 2.10 show the effects that μ_S, σ_S and λ have on the implied volatility surface. Note, specifically, how the jump parameters influence the short end of the skew more than they influence the long end. This is one of the advantages of including jumps in a stock price model — the jump terms allow for more flexibility in fitting the short end of the skew. Combining jumps and stochastic volatility makes it easier to fit both the long and short end of the skew.

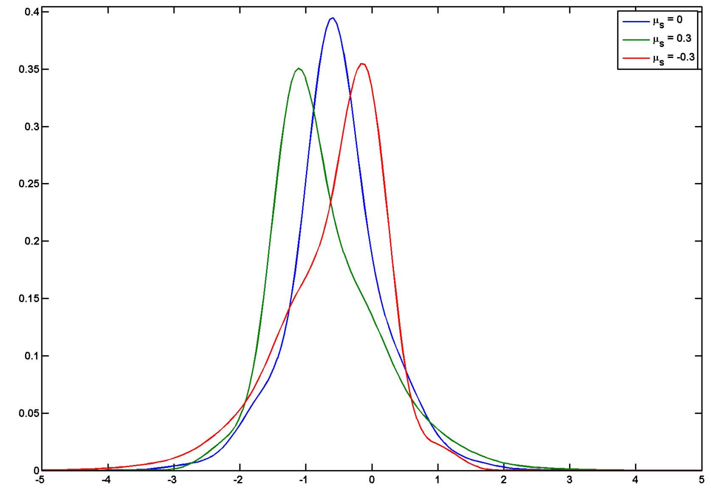

Figure 2.6 The effect of μ_S on the distribution of stock price returns under the Bates model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. The plot demonstrates how negative values of μ_S produce negative skewness in the returns distributions under the Bates model. The reverse holds for positive values of μ_S.

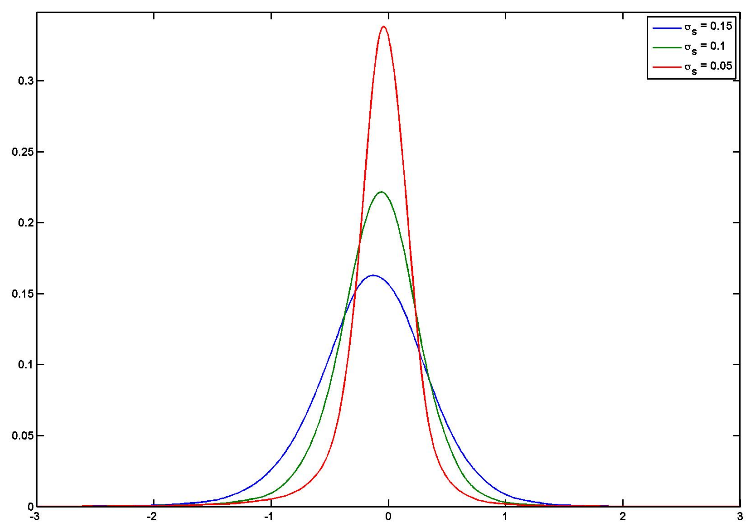

Figure 2.7 The effect of σ_S on the distribution of stock price returns under the Bates model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. We can see how larger values of the parameter increase the kurtosis in the returns distribution.

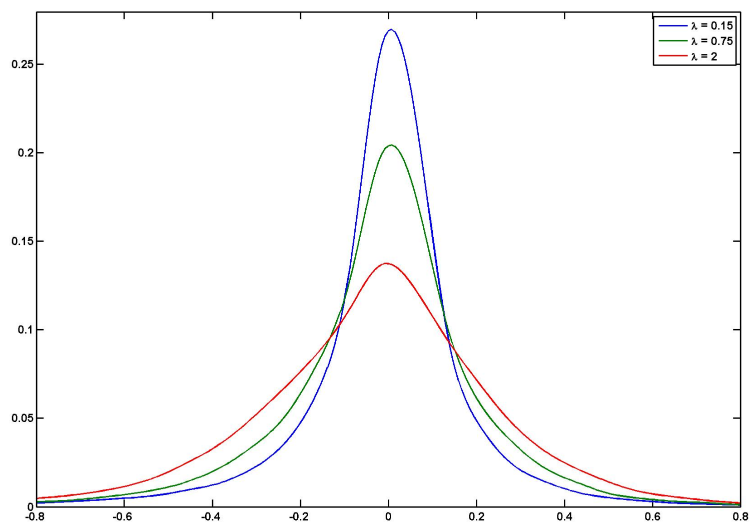

Figure 2.8 The effect of λ on the distribution of stock price returns under the Bates model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. It shows that larger values of λ yield more kurtosis in the returns distribution.

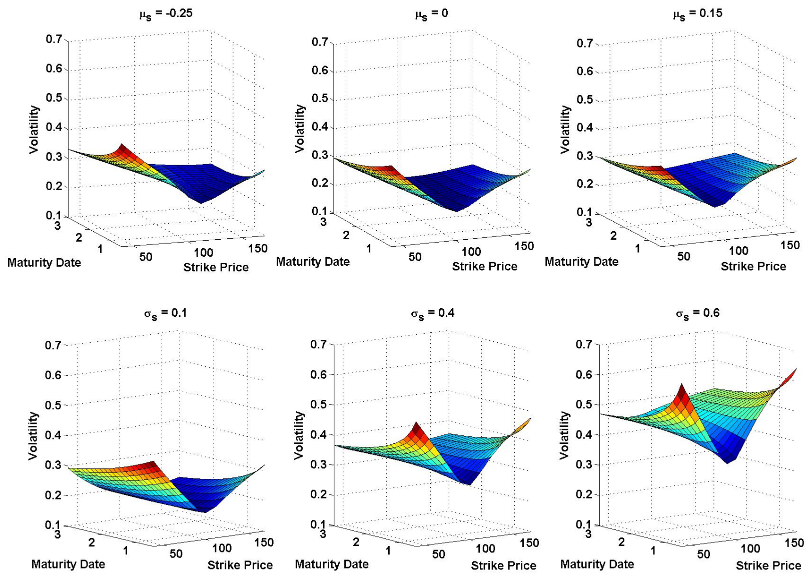

Figure 2.9 The effect of μ_S and σ_S on the Bates implied volatility surface. The figure was produced using the FFT pricing framework. The top three plots show how the skewness in the volatility surface changes for positive and negative values of μ_S. The bottom three plots show how the steepness increases for increasing values of σ_S.

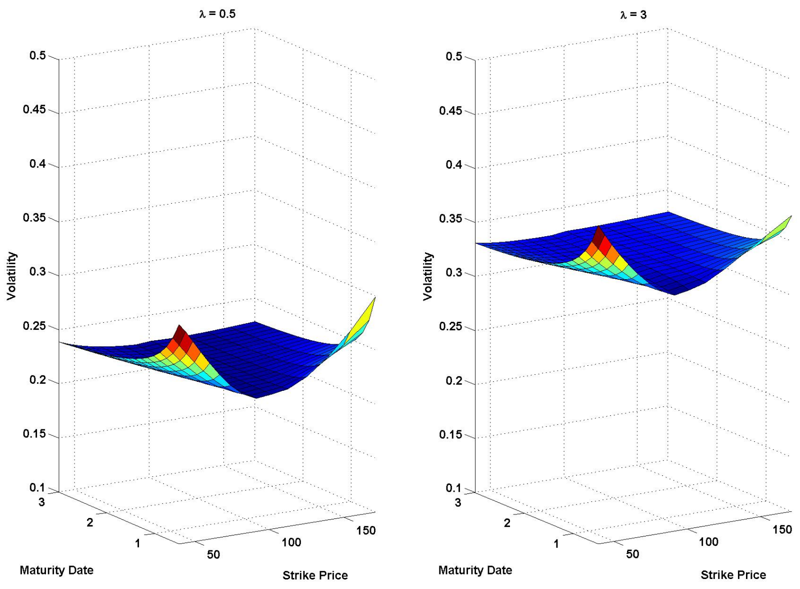

Figure 2.10 The effect of λ on the Bates implied volatility surface. The figure was produced using the FFT pricing framework. The plots show how the level of volatility increases as λ increases.

2.3 The Double Jump Stochastic Volatility Model

A natural extension of the Bates model is to include jumps in the volatility process in addition to those in the stock price process. Intuitively, it makes sense that a jump in the stock price process should trigger a correlated jump in the volatility process in that sudden, large movements in the stock price would cause increased market anxiety around that stock. As a result, we review the double jump stochastic volatility model (SVJJ) in this subsection.

Works by Broadie et al. (BCJ) [10]; Broadie and Kaya (BK) [11]; Duffie et al. (DPS) [22] and Gatheral [25] all review this model. In particular, the works by BCJ and Gatheral explore the merits and drawbacks of the SVJJ model over Bates-style models. BCJ argue in favour of a stochastic volatility model that incorporates jumps in both the stock price and variance processes, while Gatheral finds that a stochastic volatility model with jumps in the stock price process only produces the best fit to the implied volatility surface. In their analysis, BCJ use option futures data on the S&P 500 over the period from 1987 to 2003, a much longer period than many of the other empirical studies of this kind. They argue that since jumps occur relatively infrequently in stocks, it is wise to use an extended period of observation in order to reduce bias in the data. They also propose that any jump in the stock price should trigger a simultaneous jump in the underlying volatility process. The model that they consequently advocate is the SVCJ model — a stochastic volatility model with contemporaneous jumps in the stock price and its volatility. Notably, the simple stochastic volatility model (Heston) and the stochastic volatility model with jumps in the stock price process only (Bates) are specific cases of this model.

In our formulation of the SVJJ model, we follow the framework by DPS [22] closely. This model has the following risk-neutral dynamics[1]:

$$dS_t = (r - \lambda\mu_J)\, S_t dt + \sqrt{V_t} S_t d\widetilde{W}_t^{(1)} + J S_t d\widetilde{N}_t \tag{2.7}$$

$$dV_t = \kappa(\theta - V_t)\, dt + \sigma_v \sqrt{V_t} d\widetilde{W}_t^{(2)} + Z d\widetilde{N}_t \tag{2.8}$$

$$d\widetilde{W}_t^{(1)} d\widetilde{W}_t^{(2)} = \rho dt. \tag{2.9}$$

Again, $\widetilde{N}_t$ represents a Poisson process under the risk neutral measure, with jump intensity λ. The jump terms in the model are defined as follows:

$$\begin{aligned} Z &\sim \text{Exponential}(\mu_V) \\ (1+J) \mid Z &\sim \text{log-normal}\left(\mu_S + \rho_J Z, \sigma_S^2\right), \end{aligned}$$

[1] See Appendix A for an explanation of the form of the stock price process in jump-diffusion models under risk-neutral dynamics.

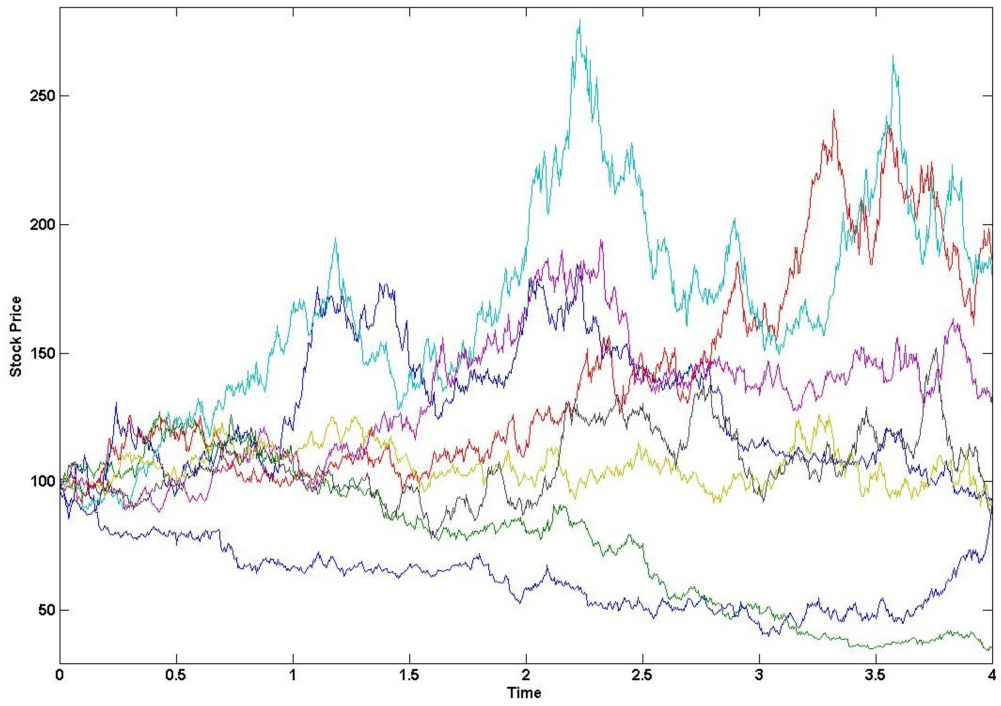

Figure 2.11 Sample SVJJ stock price paths for $S_0 = 100$, $\kappa = 1.5$, $\theta = V_0 = 0.04$, $\sigma_v = 0.2$, $\rho = 0.8$, $\lambda = 3$, $\mu_S = -0.05$, $\sigma_S = 0.0001$, $\rho_J = -0.4$, $\mu_V = 0.01$. The plot was produced using Euler Monte Carlo methods with 1000 time steps.

where

$$\mu_J = \frac{\exp\left\{\mu_S + \frac{\sigma_S^2}{2}\right\}}{1 - \rho_J \mu_V} - 1.$$

Figure 2.11 gives an example of ten paths produced using the SVJJ model. These paths exhibit even more volatility than that displayed by the Bates stock paths.

The parameters of interest in this model are ρ_J and μ_V, since the other eight parameters are the same as those in the previous models. The parameter ρ_J impacts on the skewness of the returns distribution in much the same way that ρ does. The effects of ρ_J are, however, more prevalent in the short term. Positive values for the parameter will cause jumps in the volatility process to augment those in the stock price process, inducing a positive skew in stock price returns distributions. The reverse will occur for negative values of ρ_J. This is displayed by Figure 2.12. The effects of μ_V on the stock price returns distribution are seen in Figure 2.13. Since μ_V affects the size of the jumps in the volatility process, larger values for the parameter raise the level of volatility in the stock price. This also increases the kurtosis of the returns distribution. Figure 2.14 shows how the parameters ρ_J and μ_V impact on the SVJJ implied volatility surface.

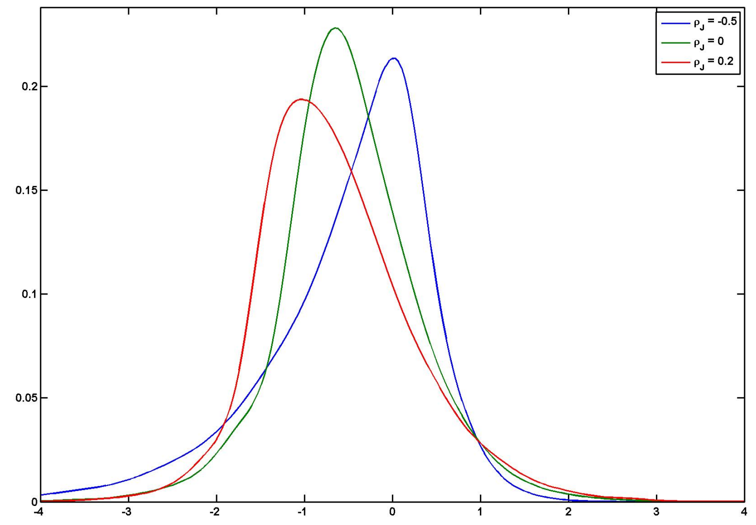

Figure 2.12 The effect of ρ_J on the distribution of stock price returns under the SVJJ model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. In a similar way to the parameter ρ — the effects of which are shown under the Heston model subsection in this chapter — ρ_J can be seen to influence the skewness of the returns distribution.

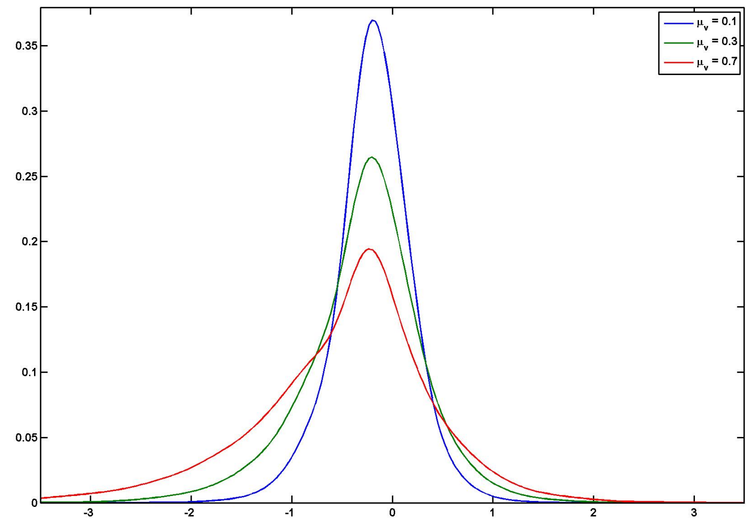

Figure 2.13 The effect of μ_V on the distribution of stock price returns under the SVJJ model. The plot was produced using Euler Monte Carlo methods with 100,000 paths and 100 time steps. Larger values of μ_V clearly increase the kurtosis in the returns distribution.

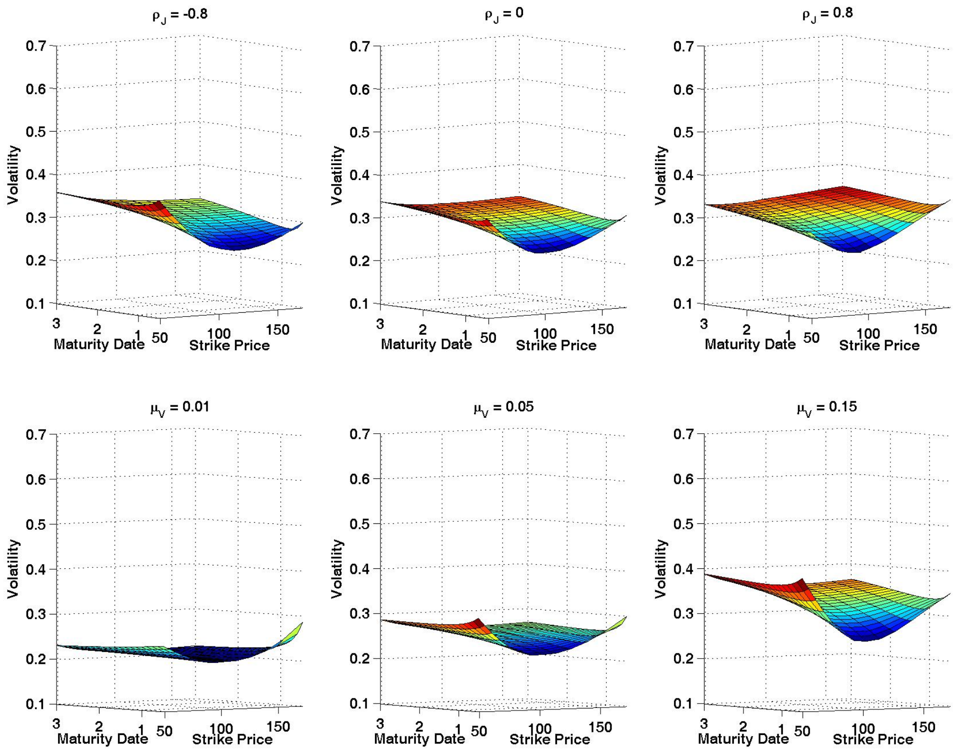

Figure 2.14 The effect of ρ_J and μ_V on the SVJJ implied volatility surface. The figure was produced using the FFT pricing methodology. The top three plots show how the skewness in the volatility surface changes for positive and negative values of ρ_J. The bottom three plots show how the steepness and level of volatility increase for increasing values of μ_V.

2.4 Price Path Comparisons for the Heston, Bates and SVJJ Models

In this section, we compare the three models considered above and examine JSE Top 40 and S&P 500 index data in our consideration of the merits and drawbacks of the different models. Figure 2.21 gives a comparison of stock price paths for the different models[2]. To give a meaningful comparison, we have ensured that the same random numbers and same jump times are used to generate all the paths. The most striking aspect of the plot is how the inclusion of jumps increases the potential for large stock price movements. The Bates model paths, and even more so, the SVJJ model paths jump at numerous points in the 4 year time horizon. This allows for large rises and drops in the stock price over small intervals in time. The Heston model, on the other hand produces a much more subdued price path than the other two models produce. Thus, the Bates and SVJJ models are able to generate returns distributions with more skewness and more kurtosis than those produced by the Heston model. This is especially true in the short term. The exclusion of jumps from the Heston model clearly limits the price movements that can be generated by the model.

The plot of the JSE Top 40 index as well as that of the S&P 500 index (Figures 2.15 and 2.18) both give evidence of large movements, as well as jumps in the index values. Notably, the market crash of 1987 is highlighted by the sharp drop in the index value in Figure 2.18. Such movements might quite possibly be modelled by the presence of jumps in the process driving the index value. The plot of the S&P 500 index also shows that there was a large decline in the value of the index between 2000 and 2003 and both index plots highlight the recent market crash. Particularly, we see rapid declines in both indices around the middle of 2008. At other times, the index plots hint at relatively calm market behaviour, with few large value movements. Stochastic volatility models that incorporate jump processes can capture these characteristics by allowing for periods of market stability and also periods of instability characterised by large movements and even jumps in stock prices. We can see such price movements in the Bates and SVJJ stock paths in Figure 2.21.

Looking further at the volatility processes of the different models, displayed in Figure 2.22, we see that the Heston and the Bates models produce identical movements in the stock price volatility. The SVJJ model, on the other hand, allows for jumps in the volatility process and this induces large, sudden movements in the process. All three plots also illustrate that high volatility values and low volatility values tend to cluster together. Specifically, the

[2]Note that the parameters chosen to produce these plots are based on reasonable results observed in the literature on such models. Different parameters would yield different plots to the ones observed here.

SVJJ volatility path demonstrates that when a jump is experienced, the level of volatility remains high for a while, before reverting to a mean level. This induces the clustering effect seen in the return time series plots for the models (Figure 2.23). Such characteristics can also be observed in Figures 2.16 and 2.19, induced by periods of high and low market volatility. The Black-Scholes model, conversely, does not exhibit any of these features. This illustrates, to an extent, the inability of the Black-Scholes model to produce empirically consistent stock price movements and returns distributions and gives credence to the use of stochastic volatility models, rather than the Black-Scholes model, in modeling stock price dynamics.

We have also seen the ability of the three stochastic volatility models to produce returns distributions which are skewed and have excess kurtosis. The Black-Scholes model, on the other hand, is only capable of producing returns which are normally distributed. Considering Figures 2.17 and 2.20, it is clear that the returns on these two indices are not normally distributed. Rather, they both seem to give evidence of distributions that are slightly negatively skewed and which have fat tails.

All these observations indicate that stochastic volatility models are far more capable of replicating market dynamics than the Black-Scholes model is able to. The inclusion of stochastic volatility and jumps in stock price models is justified by market phenomena such as volatility clustering and market crashes. It consequently seems natural that the topic of stochastic volatility and jumps should be explored further for pricing and hedging purposes. Specifically in less liquid markets, such as exotic options markets, it seems that it would be wise to use such models to obtain more reliable prices and better hedging strategies. It is largely these observations, as well as numerous empirical studies (Bakshi et al. [3], Bates [5], Broadie et al. [10], Duffie et al. [22], Gatheral [25], Heston [28]) of stochastic volatility and jumps in the price and volatility paths of stocks that has sparked our interest in this topic.

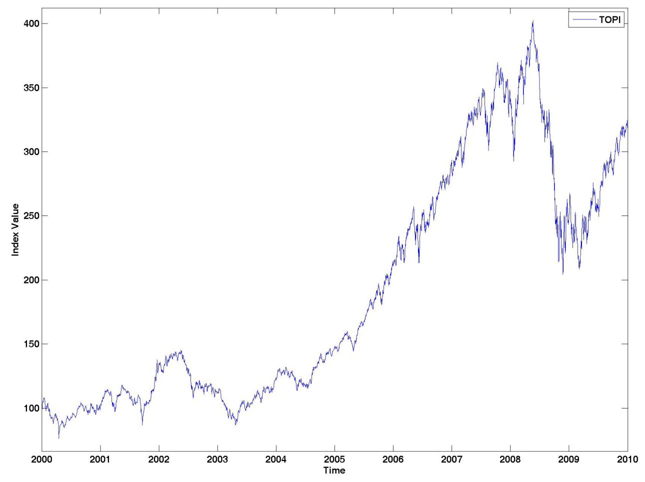

Figure 2.15 JSE Top 40 index (TOPI) plot between 1 January 2000 and 31 December 2009. In the plot, we have set the starting value of the index to 100. The plot gives evidence of large price movements, particularly from 2008 onwards.

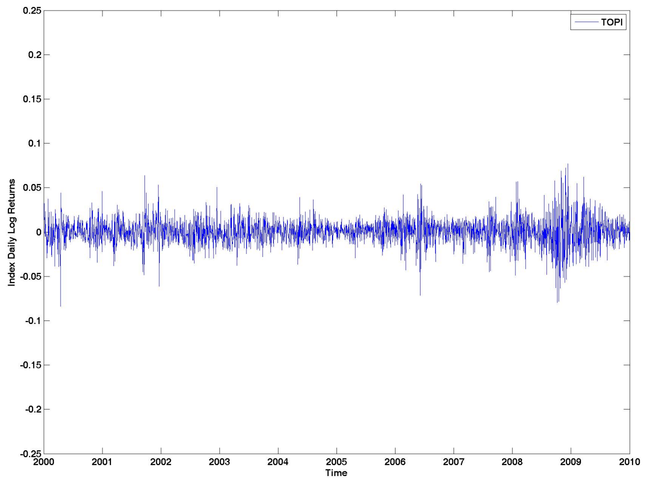

Figure 2.16 Daily returns corresponding to the JSE Top 40 index plot above. Evidence of volatility clustering is evident in the plot. The plot also shows a number of jumps in stock price returns and a large amount of volatility around the 2008 market crash. Such characteristics can be captured by stochastic volatility and jump processes.

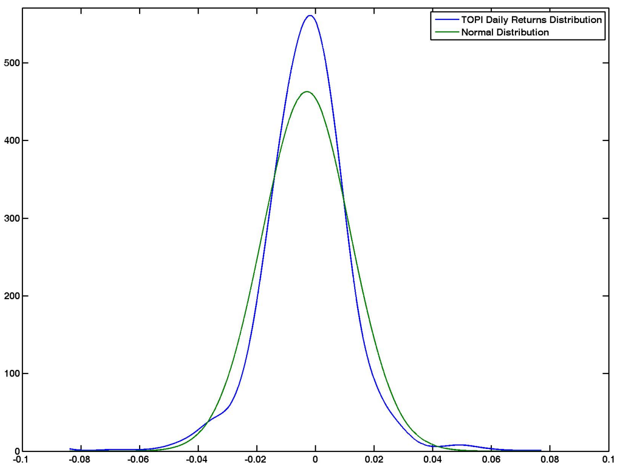

Figure 2.17 Comparison of the distribution of daily returns on the JSE Top 40 index and the normal distribution. We see here that the distribution of returns on the index has fatter tails and a taller peak than the normal distribution has. The returns distribution also seems to be slightly negatively skewed.

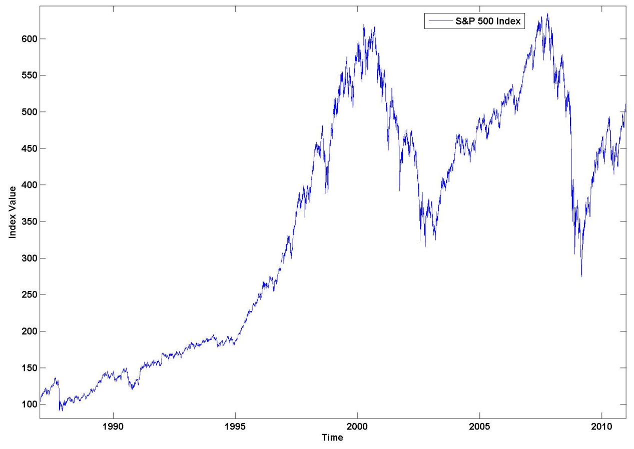

Figure 2.18 S&P 500 index plot between 1 January 1987 and 31 December 2010. In the plot, we have set the starting value of the index to 100. The plot gives evidence of large price movements, possibly due to the effects of stochastic volatility and jumps.

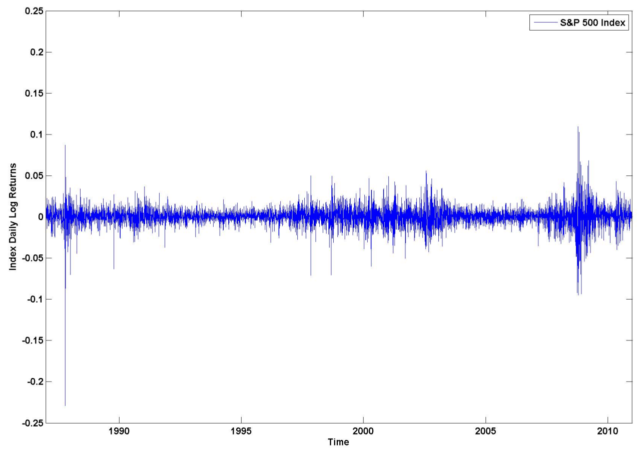

Figure 2.19 Daily returns corresponding to the S&P 500 index plot above. We can see evidence of volatility clustering as well as price jumps in the returns distribution. The stock market crashes of 1987 and 2008 stand out in the plot.

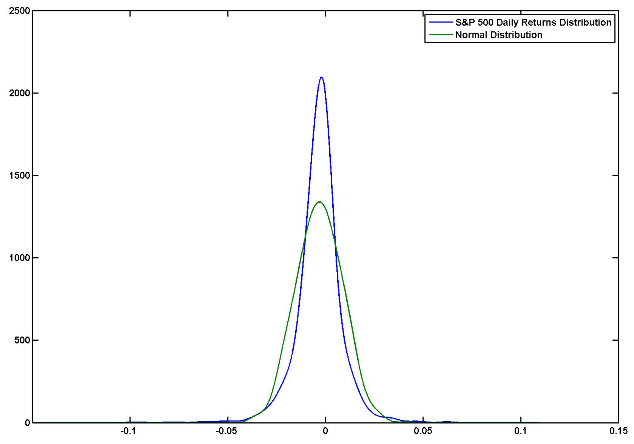

Figure 2.20 Comparison of the distribution of daily returns on the S&P 500 index and the normal distribution. We see here that the distribution of returns on the index has fatter tails and a taller peak than the normal distribution has. The returns distribution is also slightly negatively skewed. Such characteristics can be produced by stochastic volatility models.

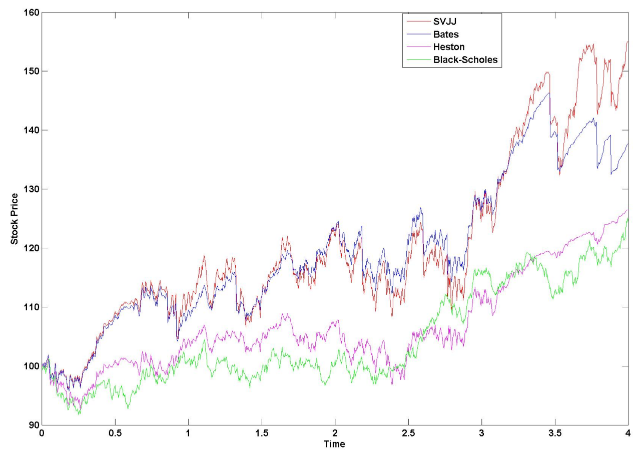

Figure 2.21 Stock price paths for the Black-Scholes, Heston, Bates and SVJJ models. The same random numbers were used to generate all the paths. Model parameters: $\kappa = 1.5$, $\theta = V_0 = 0.008$, $\sigma_v = 0.2$, $\rho = -0.8$, $\lambda_J = 3$, $\mu_S = -0.05$, $\sigma_S = 0.0001$, $\rho_J = -0.4$ and $\mu_V = 0.01$. The SVJJ and Bates paths exhibit the greatest price movements, while the Heston and Black-Scholes paths are more subdued.

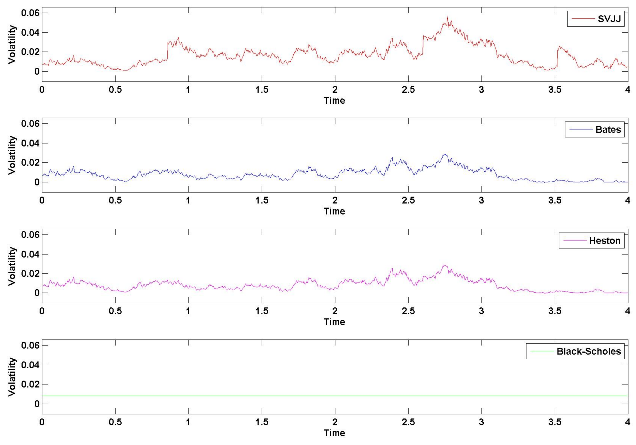

Figure 2.22 Volatility paths corresponding to the above stock price paths. We can clearly see jumps in the SVJJ volatility path, resulting in larger volatility movements than in the Bates and Heston models. The three volatility paths corresponding to the three stochastic volatility models all show signs of volatility clustering. The Black-Scholes volatility path is obviously flat.

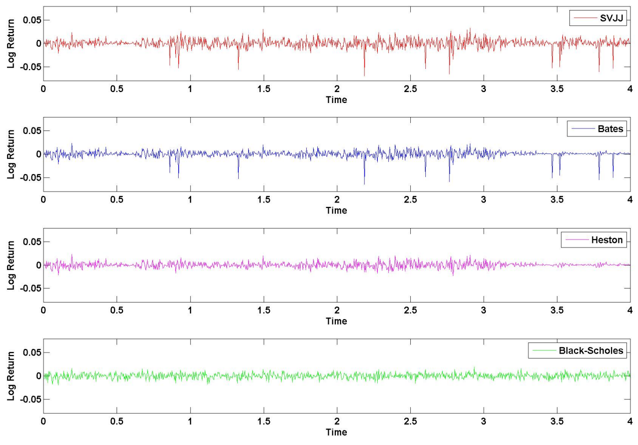

Figure 2.23 Returns corresponding to the stock price paths above. The SVJJ and Bates paths show evidence of jumps in stock returns. All three stochastic volatility models give signs of volatility clustering. This makes the models more realistic than the Black-Scholes model, which exhibits none of these characteristics.

Chapter 3

Pricing Methods

One of the main advantages of the Black-Scholes-Merton framework (Black and Scholes [7]; Merton [42]) is that it allows for the derivation of closed form option pricing formulas for vanilla options as well as many types of exotic options. The models considered in the previous chapter do not provide pricing solutions quite as easily. Many authors (notably Bates [5], Duffie et al. [22] and Heston [28]) have derived integral representations for vanilla European option prices in such situations through the use of partial differential equations and Fourier transform techniques. The use of these solutions, however, often requires the implementation of somewhat complex numerical methods. A number of the more popular methods include the fast Fourier transform (FFT) and direct integration schemes. As an alternative to deriving and implementing closed form pricing techniques, Monte Carlo methods are also very popular and robust tools for finding option prices under the dynamics of stochastic volatility models.

The application of the FFT to option pricing was made popular by Carr and Madan [13] and enables the rapid computation of option prices across a large grid of strikes. The ability of the Carr and Madan method to simultaneously compute prices for numerous options with equally spaced strike prices is one of its major computational advantages.

A review of direct integration methods is given by Gatheral [25] as well as Zhu [59]. A common way of implementing this method is to express the price of (for example) a vanilla call option as

$$C\left(S_0, K, T\right) \;=\; S_0 P_1 - K e^{-rT} P_2,$$

where, in a similar way to the Black-Scholes formula, P_1 and P_2 represent the delta and exercise probability of the option respectively. The terms P_1 and P_2 involve complicated integral expressions which can be computed using numerical integration techniques, such as the trapezoidal rule, Simpson's rule or Gaussian quadrature methods. The method of Attari [1] can also be used with direct integration schemes.

Monte Carlo methods are very popular in mathematical finance. They allow for the pricing of options by simulating stock paths under the risk neutral measure and averaging the discounted option payoffs produced by the different paths. These methods are particularly useful in the valuation of exotic options, as well as for the computation of option price sensitivities. Their popularity arises largely as a consequence of their ability to simulate stock paths of even the most complicated stock price models. They can provide option pricing and hedging solutions when no closed form alternatives are available. A drawback here, however, is that they are usually much slower than methods such as the FFT. They are also subject to statistical error: a problem that does not plague the FFT method.

We choose to focus specifically on the Carr and Madan FFT pricing method and Monte Carlo methods in the sections that follow. The FFT method by Carr and Madan is very fast and easy to implement and its ability to compute numerous option prices at once makes it useful as a calibration tool.

3.1 The Carr and Madan Fast Fourier Transform Pricing Framework for Vanilla European Call Options

The application of the FFT to vanilla option pricing gives a method of rapidly computing option prices. This method can be used whenever the characteristic function of the underlying stock price process can be derived analytically. Consequently, it has great potential for computing real time option prices, where the dynamics of the stock price process are more complex than those of geometric Brownian motion. We follow the method of Carr and Madan [13] in our application of the FFT to vanilla option pricing.

3.1.1 Introductory Definitions

Definition 1 (The Fourier Transform)**.** The Fourier transform of a square-integrable function, $g(x)$, is given by:

$$\hat{g}(u) = \int_{-\infty}^{\infty} e^{iux} g(x)\, dx. \tag{3.1}$$

Definition 2 (The Inverse Fourier Transform)**.** The inverse Fourier transform of a square-integrable function, $\hat{g}(u)$, is given by:

$$g(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-iux} \hat{g}(u)\, du. \tag{3.2}$$

The Fourier transform of the function $g(x)$ is essentially the transformation of the function from the real domain, to the frequency domain, denoted by u. Furthermore, in order to recover the function $g(x)$ from the Fourier transform, we apply the inverse Fourier transform. Next we look at the Fourier transform of the probability density function of a random variable, which is of particular importance to the implementation of the FFT.

Definition 3 (The Characteristic Function)**.** The characteristic function of a random variable, X_T, is given by:

$$\begin{aligned} \phi_{X_T}(u) &= \mathbb{E}\left[e^{iuX_T}\right] \\ &= \int_{-\infty}^{\infty} e^{iuX_T} p(X_T)\, dX_T, \end{aligned} \tag{3.3}$$

where $p(X_T)$ is the probability density function of X_T at some time $T > 0$.

3.1.2 The Fourier Transform for ATM and ITM Call Options

The first stage of the application of the FFT to call option pricing is to find the Fourier transform of the call pricing function. When evaluating the Fourier transform of this function, we follow one method for in-the-money (ITM) and at-the-money (ATM) options and a slightly different method for out-of-the-money (OTM) options. Suppose that the pricing function of a European call option is given by $c_T(k)$. Here, we denote the maturity of the option by T and the log-strike by k. Furthermore, define the price of the underlying stock at the maturity of the option to be S_T and let the risk-neutral density of $s_T = \log(S_T)$ be given by the function $\widetilde{p}(s_T)$. Then

$$c_T(k) = \int_k^{\infty} e^{-rT}\left(e^{s_T} - e^k\right)\widetilde{p}(s_T)\, ds_T. \tag{3.4}$$

Evaluating the limit as $k \to -\infty$, we see that

$$\begin{aligned} \lim_{k\to-\infty} c_T(k) &= \lim_{k\to-\infty} \int_k^{\infty} e^{-rT}\left(e^{s_T} - e^k\right)\widetilde{p}(s_T)\, ds_T \\ &= S_0. \end{aligned}$$

Consequently, $c_T(k)$ does not converge to 0 in the limit and is thus not square-integrable. Since we cannot apply the Fourier transform to a function which is not square-integrable, we need to consider a new call pricing function which is square-integrable. We do this by applying a dampening factor to $c_T(k)$ to get

$$C_T(k) := e^{\alpha k} c_T(k), \tag{3.5}$$

where α is a positive constant.

The Fourier transform of $C_T(k)$ is given by:

$$\begin{aligned}
\psi_T(u) &= \int_{-\infty}^{\infty} e^{iuk} C_T(k)\,dk \\
&= \int_{-\infty}^{\infty} e^{-rT} e^{iuk} e^{\alpha k} \int_{k}^{\infty} \left(e^{s_T} - e^{k}\right) \widetilde{p}(s_T)\,ds_T dk \\
&= \int_{-\infty}^{\infty} e^{-rT} \widetilde{p}(s_T) \int_{-\infty}^{s_T} \left(e^{s_T+\alpha k} - e^{(\alpha+1)k}\right) e^{iuk} dk ds_T \\
&\quad \text{(by changing the order of integration)} \\
&= \int_{-\infty}^{\infty} e^{-rT} \widetilde{p}(s_T) \int_{-\infty}^{s_T} \left(e^{s_T+(\alpha+iu)k} - e^{(\alpha+1+iu)k}\right) dk ds_T \\
&= \int_{-\infty}^{\infty} e^{-rT} \widetilde{p}(s_T) \left[\frac{e^{is_T(u-(\alpha+1)i)}}{(\alpha+iu)(\alpha+1+iu)}\right] ds_T \\
&= \frac{e^{-rT}}{(\alpha+iu)(\alpha+1+iu)} \int_{-\infty}^{\infty} e^{i(u-(\alpha+1)i)s_T} \widetilde{p}(s_T)\,ds_T \\
&= \frac{e^{-rT}\phi_{s_T}(u-(\alpha+1)i)}{(\alpha+iu)(\alpha+1+iu)}. \qquad (3.6)
\end{aligned}$$

Here, $\phi_{s_T}(\cdot)$ denotes the characteristic function (under the risk-neutral measure) of the log-stock price. Now, considering the inverse Fourier transform of $C_T(k)$, we see that

$$\begin{aligned}
e^{\alpha k} c_T(k) &= C_T(k) \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-iuk} \psi_T(u)\,du
\end{aligned}$$

and hence that

$$\begin{aligned}
c_T(k) &= \frac{e^{-\alpha k}}{2\pi} \int_{-\infty}^{\infty} e^{-iuk} \psi_T(u)\,du \\
&= \frac{e^{-\alpha k}}{\pi} \int_{0}^{\infty} \mathrm{Re}\left[e^{-iuk} \psi_T(u)\right] du. \qquad (3.7)
\end{aligned}$$

The above holds because $\mathrm{Re}\left[e^{-iuk}\psi_T(u)\right]$ is an even function (See Carr and Madan [13], Lee [39]). Consequently, the price of a European call option is given by

$$c_T(k) = \frac{e^{-\alpha k}}{\pi} \int_{0}^{\infty} \mathrm{Re}\left[e^{-iuk} \frac{e^{-rT}\phi_{s_T}(u-(\alpha+1)i)}{(\alpha+iu)(\alpha+1+iu)}\right] du. \qquad (3.8)$$

Choosing an Appropriate Value for α

We include the factor $e^{\alpha k}$ when performing the Fourier transform of our call pricing function to ensure that the consequent modified call pricing function is integrable over negative values of k. Since α is positive, however, this factor worsens the integrability of the modified call pricing function over positive values of k. In order to ensure that the modified call pricing function is integrable over all values of k, Carr and Madan [13] state that it is sufficient to ensure that the Fourier transform of our modified call pricing function is finite at 0. From equation (3.6), it can be seen that this will be so provided that $\phi_{s_T}\left(-\left(\alpha+iu\right)\right)$, and hence $\widetilde{\mathbb{E}}\left[S_T^{\alpha+1}\right]$ are finite (note that $\widetilde{\mathbb{E}}\left[\cdot\right]$ is the expectation operator under the risk-neutral measure). An upper bound for α can now be found by considering the analytical expression for the characteristic function. A popular choice for the value of α is a quarter of this upper bound.

Truncating the Call Pricing Function

In order to calculate option prices from equation (3.8), we need to use numerical methods to compute the integral in that equation. Consequently, we need to truncate the integral in (3.8) at some point a. This will leave us with an approximation for $c_T\left(k\right)$ given by

$$\hat{c}_T\left(k\right) = \frac{e^{-\alpha k}}{\pi}\int_0^a \operatorname{Re}\left[e^{-iuk}\frac{e^{-rT}\phi_{s_T}\left(u-\left(\alpha+1\right)i\right)}{\left(\alpha+iu\right)\left(\alpha+1+iu\right)}\right]du. \tag{3.9}$$

Now, the absolute error of this approximation will be

$$\begin{aligned}\left|c_T\left(k\right)-\hat{c}_T\left(k\right)\right| &= \left|\frac{e^{-\alpha k}}{\pi}\int_0^\infty \operatorname{Re}\left[e^{-iuk}\psi_T\left(u\right)\right]du-\frac{e^{-\alpha k}}{\pi}\int_0^a \operatorname{Re}\left[e^{-iuk}\psi_T\left(u\right)\right]du\right| \\ &= \left|\frac{e^{-\alpha k}}{\pi}\int_a^\infty \operatorname{Re}\left[e^{-iuk}\psi_T\left(u\right)\right]du\right|.\end{aligned}$$

To minimise this error, we need to choose a value of a large enough so that the value of this integral is small. Carr and Madan [13] show that, for some desired truncation error, ϵ, a must be chosen such that

$$a > \frac{e^{-\alpha k}}{\pi}\frac{\sqrt{A}}{\epsilon},$$

where A is a constant chosen such that $\widetilde{\mathbb{E}}\left[S_T^{(\alpha+1)}\right]^2 \leq A$. For a more in depth analysis of this method, see Carr and Madan [13] and Pillay [47].

3.1.3 The Fourier Transform for OTM Call Options

The method for evaluating call option prices given by equation (3.8) is effective for ATM and ITM options. When pricing fairly deep OTM call options which are close to maturity, however, the integrand in (3.8) becomes quite oscillatory. This is as a result of such options tending to their intrinsic values as they near maturity (Carr and Madan [13]). As a consequence, Carr and Madan suggest a different approach in the case of OTM options to circumvent this problem. They consider the "time value" of an OTM option.

The time value of an option is equal to the difference between the value of the option and its intrinsic value. Since an OTM option has an intrinsic value of 0, its time value is simply equal to its value. Carr and Madan thus consider a function, $z_T(k)$, which takes the value of either a T maturity call or put option (with log-strike k), whichever is out-of-the-money at inception. Defining $\zeta_T(u)$ to be the Fourier transform of $z_T(k)$, we can obtain OTM option prices through the application of the inverse Fourier transform given by:

$$z_T(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-iuk}\zeta_T(u)\,du. \tag{3.10}$$

Now, $z_T(k)$ is defined by the following relation (assuming, for simplicity, that $S_0 = 1$):

$$\begin{aligned} z_T(k) &= e^{-rT}\int_{-\infty}^{\infty}\left[\left(e^k - e^{s_T}\right)\mathbf{1}_{\{s_T<k,k<0\}}\right]\widetilde{p}(s_T)\,ds_T \\ &\quad + e^{-rT}\int_{-\infty}^{\infty}\left[\left(e^{s_T} - e^k\right)\mathbf{1}_{\{s_T>k,k>0\}}\right]\widetilde{p}(s_T)\,ds_T. \end{aligned} \tag{3.11}$$

Applying the Fourier transform to $z_T(k)$

$$\begin{aligned} \zeta_T(u) &= \int_{-\infty}^{\infty} e^{iuk} z_T(k)\,dk \\ &= \int_{-\infty}^{\infty} e^{iuk}e^{-rT}\int_{-\infty}^{\infty}\left[\left(e^k - e^{s_T}\right)\mathbf{1}_{\{s_T<k,k<0\}}\right]\widetilde{p}(s_T)\,ds_T dk \\ &\quad + \int_{-\infty}^{\infty} e^{iuk}e^{-rT}\int_{-\infty}^{\infty}\left[\left(e^{s_T} - e^k\right)\mathbf{1}_{\{s_T>k,k>0\}}\right]\widetilde{p}(s_T)\,ds_T dk \\ &= \int_{-\infty}^{0} e^{iuk}e^{-rT}\int_{-\infty}^{k}\left(e^k - e^{s_T}\right)\widetilde{p}(s_T)\,ds_T dk \\ &\quad + \int_{0}^{\infty} e^{iuk}e^{-rT}\int_{k}^{\infty}\left(e^{s_T} - e^k\right)\widetilde{p}(s_T)\,ds_T dk \\ &= \int_{-\infty}^{0} e^{-rT}\int_{s_T}^{0} e^{iuk}\left(e^k - e^{s_T}\right)\widetilde{p}(s_T)\,dk ds_T \\ &\quad + \int_{0}^{\infty} e^{-rT}\int_{0}^{s_T} e^{iuk}\left(e^{s_T} - e^k\right)\widetilde{p}(s_T)\,dk ds_T \\ &\quad \text{(by changing the order of integration)} \\ &= e^{-rT}\left[\frac{1}{1+iu} - \frac{e^{rT}}{iu} - \frac{\phi_{s_T}(u-i)}{u(u-i)}\right]. \end{aligned} \tag{3.12}$$

As with the transform for ITM and ATM options, we need to include a dampening factor here. When $k = 0$ and as T approaches 0, $z_T(k)$ becomes quite oscillatory and including the factor $\sinh(\alpha k)$ helps to counteract this. The Fourier transform of $\sinh(\alpha k)\, z_T(k)$ is given by:

$$\begin{aligned} \gamma_T(u) &= \int_{-\infty}^{\infty} e^{iuk}\sinh(\alpha k)\, z_T(k)\,dk \\ &= \frac{\zeta_T(u-i\alpha) - \zeta_T(u+i\alpha)}{2}. \end{aligned} \tag{3.13}$$

Hence, by making use of the inverse Fourier transform, the price of an OTM option is given by:

$$\begin{aligned} z_T(k) &= \frac{1}{2\pi \sinh(\alpha k)} \int_{-\infty}^{\infty} e^{-iuk} \gamma_T(u)\, du \\ &= \frac{1}{\pi \sinh(\alpha k)} \int_{0}^{\infty} \operatorname{Re}\left[e^{-iuk}\gamma_T(u)\right] du. \end{aligned} \tag{3.14}$$

3.1.4 Using the Fast Fourier Transform to Find the Call Option Price

In this section, we consider the pricing of ATM and ITM call options using the FFT algorithm. The same procedure is followed by Carr and Madan [13] and can easily be extended to the case for OTM call options.

Discretising the integral in the pricing function,

$$\hat{c}_T(k) = \frac{e^{-\alpha k}}{\pi} \int_0^a \operatorname{Re}\left[e^{-iuk}\psi_T(u)\right] du,$$

by using the trapezoidal rule gives us:

$$\hat{c}_T(k) \approx \operatorname{Re}\left[\frac{e^{-\alpha k}}{\pi} \sum_{j=1}^{N} e^{-iu_j k}\psi_T(u_j)\,\Delta\right], \tag{3.15}$$

where Δ gives us the distance between successive points on our discretised integration grid, $u_j = \Delta(j-1)$ and $a = N\Delta$.

Now, the FFT is an efficient method of computing the sum

$$w(v) = \sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(v-1)} x(j) \tag{3.16}$$

for $v = 1, 2, \ldots, N$. Consequently, we want to manipulate (3.15) to look like (3.16). This can be achieved by defining

$$k_v = -b + \eta(v-1), \tag{3.17}$$

where $b = \frac{N\eta}{2}$. Equation (3.17) gives us N log-strike values at regular intervals of width η, ranging from $-b$ to b. Finally, setting $\eta\Delta = \frac{2\pi}{N}$, we get

$$\begin{aligned} \hat{c}_T(k_v) &\approx \operatorname{Re}\left[\frac{e^{-\alpha k_v}}{\pi} \sum_{j=1}^{N} e^{-i\eta\Delta(j-1)(v-1)} e^{ibu_j}\psi_T(u_j)\,\Delta\right] \\ &= \operatorname{Re}\left[\frac{e^{-\alpha k_v}}{\pi} \sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(v-1)} e^{ibu_j}\psi_T(u_j)\,\Delta\right]. \end{aligned} \tag{3.18}$$

Factoring in Simpson's rule weightings to the above will help to obtain more accurate prices for large values of Δ (and hence small spaces between successive strike prices). This gives us

$$\hat{c}_T(k_v) = \operatorname{Re}\left[\frac{e^{-\alpha k_v}}{\pi}\sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(v-1)} e^{ibu_j}\psi_T(u_j)\frac{\Delta}{3}\left(3+(-1)^j-\mathbf{1}_{\{j=1\}}\right)\right], \quad (3.19)$$

where $\mathbf{1}$ is the indicator function. This is almost identical to equation (3.16), with

$$x(j) = e^{ibu_j}\psi_T(u_j)\frac{\Delta}{3}\left(3+(-1)^j-\mathbf{1}_{\{j=1\}}\right).$$

Out-of-the-Money Options. For OTM options we have

$$\hat{c}_T(k) \approx \frac{1}{\pi\sinh(\alpha k)}\int_0^a \operatorname{Re}\left[e^{-iuk}\gamma_T(u)\right]du.$$

Discretising this in a similar way to before, we get

$$\hat{c}_T(k_v) = \operatorname{Re}\left[\frac{1}{\pi\sinh(\alpha k_v)}\sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(v-1)} e^{ibu_j}\gamma_T(u_j)\frac{\Delta}{3}\left(3+(-1)^j-\mathbf{1}_{\{j=1\}}\right)\right]. \quad (3.20)$$

3.1.5 The Fast Fourier Transform Algorithm

The power of the FFT (see Zhu [59]) lies in its ability to reduce the number of operations required to compute sums such as those in equations (3.19) and (3.20). Computing N option prices using either of these would require a number of arithmetical operations of the order of $O(N^2)$. The FFT, however, drastically reduces this number. Consider the discretised call pricing function for ATM/ITM options given by equation (3.19). We re-write it with

$$x(j) = e^{ibu_j}\psi_T(u_j)\frac{\Delta}{3}\left(3+(-1)^j-\mathbf{1}_{\{j=1\}}\right),$$

to get

$$\hat{c}_T(k_v) = \frac{e^{-\alpha k_v}}{\pi}\sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(v-1)}x(j),$$

where we ignore the use of the operator $\operatorname{Re}[\cdot]$ for simplicity. We can now split this sum into two parts by setting $M = \frac{N}{2}$. From Zhu [59]:

$$\begin{aligned}
\hat{c}_T(k_v) &= \frac{e^{-\alpha k_v}}{\pi}\sum_{j=1}^{\frac{N}{2}} e^{-i\frac{2\pi}{N}[2(j-1)](v-1)}x(2j-1) + \frac{e^{-\alpha k_v}}{\pi}\sum_{j=1}^{\frac{N}{2}} e^{-i\frac{2\pi}{N}[2(j-1)+1](v-1)}x(2j) \\
&= \frac{e^{-\alpha k_v}}{\pi}\left\{\sum_{j=1}^{M} e^{-i\frac{2\pi}{M}(j-1)(v-1)}x(2j-1) + e^{-i\frac{2\pi}{N}(v-1)}\sum_{j=1}^{M} e^{-i\frac{2\pi}{M}(j-1)(v-1)}x(2j)\right\}.
\end{aligned}$$

Splitting this into two parts, we see that

$$\hat{c}_T(k_v) = \frac{e^{-\alpha k_v}}{\pi}\left[\hat{c}_T^{(\text{odd})}(v) + e^{-i\frac{2\pi}{N}(v-1)}\hat{c}_T^{(\text{even})}(v)\right] \tag{3.21}$$

if $v < M+1$, and

$$\hat{c}_T(k_v) = \frac{e^{-\alpha k_v}}{\pi}\left[\hat{c}_T^{(\text{odd})}(v-M) + e^{-i\frac{2\pi}{N}(v-1)}\hat{c}_T^{(\text{even})}(v-M)\right] \tag{3.22}$$

if $v \geq M+1$. Now for a given value of v, say v^* where $1 \leq v^* \leq M$,

$$\begin{aligned}&\left[\hat{c}_T^{(\text{odd})}(v) + e^{-i\frac{2\pi}{N}(v-1)}\hat{c}_T^{(\text{even})}(v)\right]\Bigg|_{v=v^*}\\ &\quad= \left[\hat{c}_T^{(\text{odd})}(v-M) - e^{-i\frac{2\pi}{N}(v-1)}\hat{c}_T^{(\text{even})}(v-M)\right]\Bigg|_{v=v^*+M},\end{aligned}$$

and so we only need to compute the values of 3.21 and we will automatically have those for 3.22. Furthermore, we can break down each of the sub-sequences $\hat{c}_T^{(\text{even})}(v)$ and $\hat{c}_T^{(\text{odd})}(v)$ into two further sub-sequences. Continuing this way, we will eventually arrive at a series of sub-sequences, each of length 1. Ultimately, this allows us to reduce the number of computations required to compute the discretised Fourier transform from $O(N^2)$ to $O(N\log_2 N)$. The reduced number of computations means that the FFT can provide solutions to Fourier transforms much faster than simple summation routines are able to. This is specifically useful when dealing with large values of N.

3.1.6 Characteristic Functions for the Heston, Bates and SVJJ Models

In this subsection, we consider the characteristic functions of the Heston, Bates and SVJJ models. For a more in depth overview of these, see Appendix B.

The Heston Model Characteristic Function

The characteristic function of the log-stock price under the Heston model is given by (Duffie et al. [22], Gatheral [25], Heston [28], Kilin [35]):

$$\begin{aligned}\phi_{s_T}(u) &= \widetilde{\mathbb{E}}\left[e^{ius_T}\right]\\ &= \exp\{C(u,T)\theta + D(u,T)V_0 + iu(\log(S_0) + rT)\},\end{aligned} \tag{3.23}$$

where V_0 is the initial value of the variance process, T is the expiration date of the option and

$$C(u,T) = \kappa\left[r_{\text{neg}}T - \frac{2}{\sigma_v^2}\log\left(\frac{1-ge^{-dT}}{1-g}\right)\right] \tag{3.24}$$

$$D(u,T) = r_{\text{neg}}\left[\frac{1-e^{-dT}}{1-ge^{-dT}}\right], \tag{3.25}$$

with

$$\begin{aligned} g &:= \frac{r_{\text{neg}}}{r_{\text{pos}}} \\ r_{\text{pos/neg}} &= \frac{\beta \pm d}{2\gamma} \\ d &= \sqrt{\beta^2 - 4\alpha\gamma} \\ \alpha &= \frac{\left(-u^2 - iu\right)}{2} \\ \beta &= \kappa - \rho\sigma_v iu \\ \gamma &= \frac{\sigma_v^2}{2}. \end{aligned}$$

The Bates Model Characteristic Function

The characteristic function of the log-stock price in the Bates model is the same as that in the Heston model, with the addition of a "jump part". This gives us (Bates [5], Duffie et al. [22], Gatheral [25], Kilin [35]):

$$\begin{aligned} \phi_{s_T}(u) &= \widetilde{\mathbb{E}}\left[e^{ius_T}\right] \\ &= \exp\left\{C(u,T)\,\theta + D(u,T)\,V_0 + P(u)\,\lambda T + iu\left(\log\left(S_0\right) + rT\right)\right\}, \end{aligned} \tag{3.26}$$

where

$$P(u) = -\mu_J iu + \left[\left(1+\mu_J\right)^{iu} e^{\sigma_S^2\left(\frac{iu}{2}\right)(iu-1)} - 1\right]. \tag{3.27}$$

The functions $C(u,T)$ and $D(u,T)$ have the same form as for the Heston model.

The SVJJ Model Characteristic Function

The characteristic function of the log-stock price in the SVJJ model is similar to that in the Bates model, however it allows for jumps in both the stock price and variance processes. As a result, we find that the characteristic function of this model has a similar form to that of the Bates model, with a more complicated jump component. This gives us (Duffie et al. [22], Gatheral [25]):

$$\begin{aligned} \phi_{s_T}(u) &= \widetilde{\mathbb{E}}\left[e^{ius_T}\right] \\ &= \exp\left\{C(u,T)\,\theta + D(u,T)\,V_0 + P(u,T)\,\lambda + iu\left(\log\left(S_0\right) + rT\right)\right\}, \end{aligned} \tag{3.28}$$

where

$$P(u,T) = -T\left(1 + iu\mu_J\right) + \exp\left\{iu\mu_S + \frac{\sigma_S^2 (iu)^2}{2}\right\}\nu \tag{3.29}$$

and

$$\begin{aligned}\nu &= \frac{\beta + d}{(\beta + d)\,c - 2\mu_V \alpha} T + \frac{4\mu_V \alpha}{(dc)^2 - (2\mu_V \alpha - \beta c)^2} \times \\ &\quad \log\left[1 - \frac{(d-\beta)\,c + 2\mu_V \alpha}{2dc}\left(1 - e^{-dT}\right)\right] \qquad (3.30) \\ c &= 1 - iu\rho_J \mu_V.\end{aligned}$$

Again, $C(u,T)$ and $D(u,T)$ have the same form as for the Heston model. The expressions for β, d and α are also the same as in the case for the Heston model.

3.1.7 The Complex Logarithm in the Heston Characteristic Function

Zhu [59] gives a concise overview of the problem with the complex logarithm in the Heston model. He also presents some of the popular methods of solving this issue.

The numerical implementation of Heston's [28] original formulation of the characteristic function for the model gives rise to numerical instability due to the presence of a complex logarithm. This issue, by extension, also effects the other two models that we are concerned with. Any complex number can be expressed as

$$\begin{aligned}z &= x + iy \\ &= ae^{ib} \\ &= a\left(\cos(b) + i\sin(b)\right),\end{aligned}$$

where $a = \sqrt{a^2 + b^2}$ and $b = b_0 + 2\pi m$ such that $b_0 \in [-\pi, \pi]$ and m is an integer. Thus,

$$\begin{aligned}z &= a\left(\cos(b_0) + i\sin(b_0)\right) \\ &= ae^{ib_0}\end{aligned}$$

by Euler's formula and the properties of sin and cos. Furthermore, the logarithm of z can be expressed as

$$\begin{aligned}\log(z) &= \log\left(ae^{ib}\right) \\ &= \log(a) + ib \\ &= \log(a) + i\left(b_0 + 2\pi m\right).\end{aligned}$$

This illustration shows that the complex number, z, is fully and uniquely characterised by a and b_0. The logarithm of this number, however, depends on a, b_0 and m in such a way that any selected value of m will yield the same value for the complex logarithm.

In general, computational software programs thus set m to zero and consider only the value of a and b_0 — the principal branch — in the computation of a complex number and its logarithm. While this approach is acceptable for individual computations of complex numbers, it causes problems in computations involving the characteristic function of the Heston model. Specifically, it leads to discontinuities in the integrand functions involving the Heston characteristic function in the option pricing expressions for the model. Ignoring these effects can often lead to erroneous option prices.

There are a number of algorithms to take care of this problem, some of which are reviewed by Zhu [59]. In our case, we use a different formulation of the characteristic function of the Heston model to that originally proposed by Heston [28]. This is one which is derived by Gatheral [25] and involves a modification to the complex logarithm in the characteristic function to ensure that it never crosses the negative real axis. This prevents unnecessary branch cuts in the complex logarithm and solves the discontinuity problem. Consequently, it is safe to implement this method without worrying about unwittingly obtaining incorrect option prices.

3.1.8 Drawbacks and Alternatives to the Fast Fourier Transform

The FFT method described above is a fast and efficient method for computing option prices, where the relevant stock price model does not produce a simple closed-form option pricing formula (whereas the Black-Scholes model, for example, does exhibit an easily computable closed-form option pricing formula). This is particularly relevant in our case, where none of the models that we have considered produces such a formula. The ability of the FFT to simultaneously compute option prices for a large range of strikes is also particularly useful. This property greatly reduces computation times for model calibration. The method does, however, have a number of drawbacks and there are alternative pricing methods that can also be used, instead of the FFT method, to find option prices.

One of the major drawbacks of the FFT scheme is that it forces log-strike prices to fall on the grid $k_v = -b + \lambda\,(v-1)$ (with equally spaced grid points). As a result, the method is limited to pricing only options whose corresponding log-strike prices fall on that grid. To price options with log-strikes that do not fall on the grid requires the use of an appropriate interpolation scheme. Deciding which one to use is not always easy and, regardless of the scheme chosen, some numerical inaccuracy will always result. This can, if not controlled correctly, negatively impact on pricing, calibration and hedging schemes.

Another drawback of the FFT method is that the value of N, specifying the number of grid points must always be a power of 2. This is apparent by considering the way in which the FFT algorithm reduces the number of computations required to compute the discretised inverse Fourier transform. This leads to a limitation in the specification of the upper bound of integration for the inverse transform.

A final drawback of the FFT method comes from the relationship $\lambda\Delta = \frac{2\pi}{N}$. As a result of this, the size of the spacings in the integration grid, and those in the strike grid are inversely related. If we want to have small spaces between points on the log-strike grid, then we must settle for large spaces between points on the integration grid (or vice versa). This obviously impacts negatively on the accuracy of the method. The inclusion of Simpson's rule weightings when computing the discrete inverse Fourier transform, as set out in the Carr and Madan [13] option pricing framework, can help to overcome this.

Alternatives to the FFT pricing method include the recently developed COS method by Fang et al. [23], as well as the fractional FFT and direct integration (DI) schemes. Kilin [35] presents an informative comparison of the FFT, fractional FFT and DI schemes. In his paper, he illustrates how an improvement to the FFT method of Carr and Madan — to yield the fractional FFT method — can greatly increase the computational speed of the pricing scheme. He further analyses a caching technique that can be used in conjunction with direct integration schemes to make the computation of option prices for a large range of strikes under the schemes more efficient. He concludes that this final method is the most efficient of the three.

3.2 Monte Carlo Methods

Monte Carlo methods are used extensively in mathematical finance. They provide a convenient way of simulating stock price distributions and pricing options where closed form solutions are difficult to derive, or do not exist at all. For these reasons, the use of Monte Carlo methods is particularly useful to us. Kloeden and Platen [36] provide a rigorous treatment on the simulation of stochastic differential equations. Of particular interest to us is their derivation of the Itô-Taylor expansion in that it forms the basis of the Euler-Maruyama simulation scheme. Gatheral [25] also examines the application of Monte Carlo methods to the simulation of stochastic volatility models. The paper by Broadie and Kaya [11] provides an excellent treatment on exact simulation schemes for the three models with which we are concerned. Such schemes allow for the simulation of stock price processes by sampling from the exact distributions of the stock price and volatility process increments.

We also draw from Poklewski-Koziell [48] for our treatment on Monte Carlo methods for the Heston model.

In the sections that follow, we present the Euler-Maruyama and exact simulation schemes for the Heston, Bates and SVJJ models. We also look at the application of these schemes to vanilla call pricing.

3.2.1 The Itô-Taylor Expansion

Consider a one dimensional Itô stochastic differential equation (SDE) given by (see Kloeden and Platen [36])

$$dX_t = \alpha\left(X_t\right)dt + \beta\left(X_t\right)dZ_t, \tag{3.31}$$

or equivalently in integral form

$$X_t = X_0 + \int_0^t \alpha\left(X_u\right)du + \int_0^t \beta\left(X_u\right)dZ_u, \tag{3.32}$$

where $\alpha\left(X_t\right), \beta\left(X_t\right) \in C^2(\Re)$ are stochastic processes adapted to the natural filtration generated by X_t. As usual, Z_t is a standard Brownian motion. Next, by applying Itô's Lemma to the function $f\left(X_t\right) \in C^2(\Re)$ we get

$$f\left(X_t\right) = f\left(X_0\right) + \int_0^t L^0 f\left(X_u\right)du + \int_0^t L^1 f\left(X_u\right)dZ_u, \tag{3.33}$$

for all $t \geq 0$, where

$$\begin{aligned} L^0 &= \alpha\frac{\partial}{\partial x} + \frac{1}{2}\beta^2\frac{\partial^2}{\partial x^2} \\ L^1 &= \beta\frac{\partial}{\partial x}. \end{aligned}$$

Now, by applying Itô's Lemma to the processes $\alpha\left(X_t\right)$ and $\beta\left(X_t\right)$, it can shown that equation (3.32) becomes

$$X_t = X_0 + \alpha\left(X_0\right)\int_0^t du + \beta\left(X_0\right)\int_0^t dZ_u + R_1, \tag{3.34}$$

where the remainder term is defined by

$$\begin{aligned} R_1 = {} & \int_0^t\int_0^u L^0\alpha\left(X_y\right)dydu + \int_0^t\int_0^u L^1\alpha\left(X_y\right)dZ_ydu \\ & + \int_0^t\int_0^u L^0\beta\left(X_y\right)dydZ_u + \int_0^t\int_0^u L^1\beta\left(X_y\right)dZ_ydZ_u. \end{aligned} \tag{3.35}$$

If we do the same for $L^1\beta\left(X_t\right)$ in (3.35), we get

$$\begin{aligned} X_t &= X_0 + \alpha\left(X_0\right)\int_0^t du + \beta\left(X_0\right)\int_0^t dZ_u \\ &\quad + L^1\beta\left(X_0\right)\int_0^t\int_0^u dZ_y dZ_u + R_2, \end{aligned} \tag{3.36}$$

where we define the remainder term

$$\begin{aligned} R_2 &= \int_0^t\int_0^u L^0\alpha\left(X_y\right)dydu + \int_0^t\int_0^u L^1\alpha\left(X_y\right)dZ_y du \\ &\quad + \int_0^t\int_0^u L^0\beta\left(X_y\right)dydZ_u + \int_0^t\int_0^u\int_0^y L^0L^1\beta\left(X_v\right)dvdZ_y dZ_u \\ &\quad + \int_0^t\int_0^u\int_0^y L^1L^1\beta\left(X_v\right)dZ_v dZ_y dZ_u. \end{aligned} \tag{3.37}$$

The Itô-Taylor expansion forms the basis for the derivation of the Euler-Maruyama and one-dimensional Milstein schemes. In what follows, we implement the Euler-Maruyama scheme for the three stochastic volatility models. We choose to implement this method due to its simplicity and speed (relative to other Monte Carlo schemes).

3.2.2 The Euler-Maruyama Simulation Scheme

Euler Monte Carlo for the Heston Model

Truncating equation (3.34) just before the remainder term and applying it to the log-stock price and variance processes of the Heston model yields the Euler-Maruyama scheme for Heston. We consider the log of the stock process and not simply the stock process itself to enforce positive stock price values over all possible simulation paths. Applying Itô's Lemma to the function $f\left(S_t\right) = \log\left(S_t\right)$ yields

$$d\log S_t = rdt - \frac{1}{2}V_t dt + \sqrt{V_t}d\widetilde{W}_t^{(1)}. \tag{3.38}$$

We can apply the Cholesky decomposition to enforce correlation between the log-stock price and variance processes. This gives us

$$d\log S_t = rdt - \frac{1}{2}V_t dt + \sqrt{V_t}\left[\rho d\widetilde{Z}_t^{(2)} + \sqrt{1-\rho^2}d\widetilde{Z}_t^{(1)}\right] \tag{3.39}$$

$$dV_t = \kappa\left(\theta - V_t\right)dt + \sigma_v\sqrt{V_t}d\widetilde{Z}_t^{(2)}, \tag{3.40}$$

for all $t \geq 0$, where $\widetilde{Z}_t^{(1)}$ and $\widetilde{Z}_t^{(2)}$ are two independent Brownian motions. Discretising the two equations above according to equation (3.34) (excluding the remainder term) yields the Euler-Maruyama simulation scheme for the Heston model:

$$\Delta\log S_t = r\Delta t - \frac{1}{2}V_t\Delta t + \sqrt{V_t}\left[\rho\Delta\widetilde{Z}_t^{(2)} + \sqrt{1-\rho^2}\Delta\widetilde{Z}_t^{(1)}\right] \tag{3.41}$$

$$\Delta V_t = \kappa\left(\theta - V_t\right)\Delta t + \sigma_v\sqrt{V_t}\Delta\widetilde{Z}_t^{(2)}, \tag{3.42}$$

where Δ is used to represent the change in the respective variable.

Now, the form of the continuous variance process prevents it from ever going below zero. Discretising it, however, opens up the possibility for the occurrence of negative variance values. This is obviously an undesirable situation and a fix is required in case this should happen (which is inevitable when a large number of simulation paths is produced). To this end, Lord et al. [40] give a summary of a number of different, but simple fixes, all of which entail either reflecting or absorbing the discretised volatility process as soon as it goes negative. Reflection is achieved by applying the absolute value operator to negative variance terms. Absorption involves setting negative variance terms equal to zero. Such fixes ultimately distort the distribution of stock prices, although reflecting variance values which become "very negative" can induce a larger positive bias than merely setting them equal to zero. As a result, the absorption fix is often preferred. Other fixes entail reflecting or absorbing only terms which are contained within a square root. These do not necessarily solve the problem of negative variances, they only prevent complex stock price values from occurring, and can also lead to greater biases if the variance process becomes even more negative.

The five fixes considered by Lord et al. [40] for Heston's model are 1) the absorption fix; 2) the reflection fix; 3) the Higham and Mao fix, where only negative variance values in the square root term of the variance process are reflected; 4) the partial truncation fix, where only negative variance values in the square root term of the variance process are absorbed; and 5) the full truncation fix, where negative variance values in the square root term and in the drift term of the variance process are absorbed.

Figure 3.1 gives a graphic comparison of the five methods. As can be seen from the graph — and as shown by Lord et al. — the full and partial truncation schemes (but most notably, the full truncation scheme) give the fastest convergence to the true option price. These two fixes induce less bias than the others by allowing the variance process to become negative, instead of constantly forcing it to be greater than or equal to zero. We therefore make use of the full truncation fix in this project.

Euler Monte Carlo Extension to the Bates Model

A basic simulation scheme for the Bates model follows directly from the Euler-Maruyama scheme for the Heston model. The jump and diffusion parts of the stock price process under the Bates model can be simulated separately and multiplied together at the end. To simulate the diffusion part of the stock price process, we follow exactly the same procedure

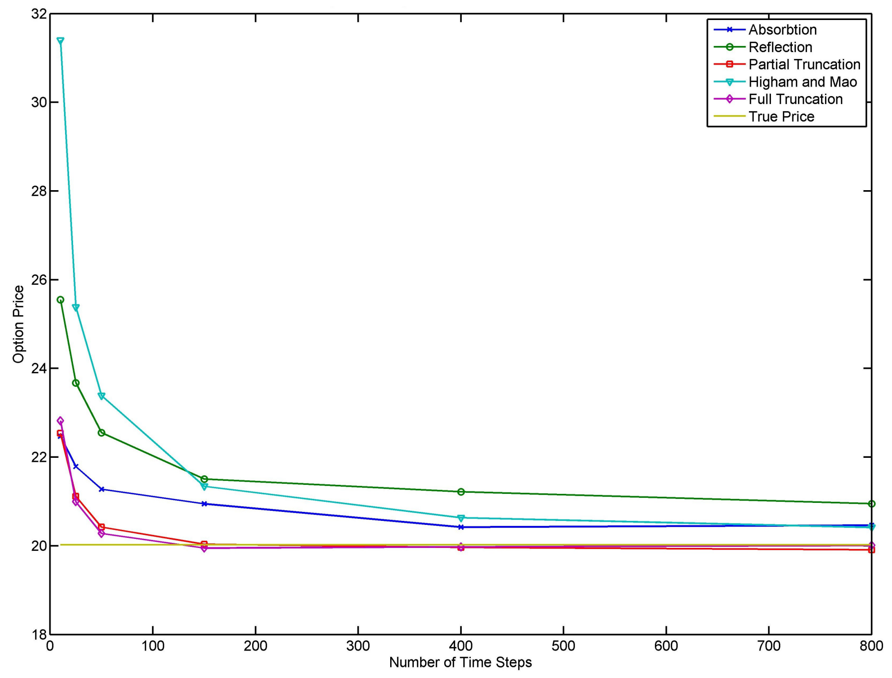

Figure 3.1 Comparison of different fixes for the occurrence of negative variance values in Euler-Maruyama simulation scheme for the Heston model (plot taken from Poklewski-Koziell [48]).

as above — we use the Euler-Maruyama discretisation scheme for the Heston model to find a preliminary value for S_t, say S_t^-. Next we simulate the jump part of the stock price process. To achieve this, we first need to simulate a Poisson process, $\widetilde{N}_t$, with intensity λ. The simulated process $\widetilde{N}_t$ then gives us the number of jumps that occur between times 0 and t. We denote this number by n. Next, we simulate jump sizes according to the distribution of $1+J$ — i.e. we generate n log-normal random variates with mean μ_S and variance σ_S^2. If we label each one of these jump sizes $\xi_i^{(S)}$ for $i = 1, \ldots, n$, then the final stock price under this scheme for the Bates model is given by the product of the final stock price generated by the Euler-Maruyama scheme for the Heston model and each of the $\xi_i^{(S)}$. This can be expressed as:

$$S_t = S_t^- \prod_{i=1}^{n} \xi_i^{(S)}. \tag{3.43}$$

Euler Monte Carlo Extension to the SVJJ Model

In a similar way to the Euler-Maruyama extension to the Bates Model, we can extend the Euler-Maruyama method that was used for the Heston model and apply it to the SVJJ model. The implementation of this method for the SVJJ model is, however, slightly more complicated than it was for the Bates model. This is due to the fact that jumps in the volatility process prevent us from simply simulating the diffusion part of the model separately from the jump part. Instead, we first need to simulate jump times and magnitudes for the two processes and then simulate their diffusion parts between the jump times.

We begin the simulation procedure by simulating a Poisson process, $\widetilde{N}_t$, between times 0 and t. This gives us the number, n, of jumps occurring between 0 and t and the times, t_i, at which the jumps occur, where $i = 1, \ldots, n$ $(0 \leq t_1 \leq \cdots \leq t_n \leq t)$. If $n = 0$, we ignore the jump part of the simulation scheme and simply simulate the whole process (up to time t) as we did for the Heston model. Otherwise, in each interval $[t_{i-1}, t_i]$ (note that we set $t_0 = 0$), we first simulate the diffusion parts of the two processes in the same way that we did for the Heston model. This gives us preliminary values for the stock price and variance processes at time t_i. We denote these by $S_{t_i}^-$ and $V_{t_i}^-$ respectively. We now proceed to simulate the jump sizes for the two processes at t_i. The size of the jump in the volatility process, $\xi_{t_i}^{(V)}$, has an Exponential$\,(\mu_V)$ distribution and the size of the jump in the stock price process, $\xi_{t_i}^{(S)}$, has a log-normal$\left(\mu_S + \rho_J \xi_{t_i}^{(V)}, \sigma_S^2\right)$ distribution. We can then update the values of the two processes to give us

$$\begin{aligned} S_{t_i} &= S_{t_i}^- \xi_{t_i}^{(S)} && (3.44) \\ V_{t_i} &= V_{t_i}^- + \xi_{t_i}^{(V)}, && (3.45) \end{aligned}$$

where S_{t_i} and V_{t_i} are the final values of the two processes at t_i. Once we have repeated this procedure for all values of i, we need to complete the Euler-Maruyama scheme by simulating the stock price and variance processes between time t_n and time t. If time $t_n = t$, then $S_t = S_{t_n}$, $V_t = V_{t_n}$ and we are done. Alternatively, if $t_n \leq t$, then no jumps occur in the interval $[t_n, t]$ and we apply the method used for the Heston model to simulate the processes between t_n and t and obtain the values of S_t and V_t.

3.2.3 The Exact Simulation Scheme

Exact Simulation for the Heston Model

The exact simulation scheme for the Heston model is laid out in Broadie and Kaya [11] and is in some sense the gold standard of simulation techniques for the model. It is a very accurate simulation method, but also a very computationally intensive and time consuming one. The scheme involves sampling from the exact distribution of the stock price and variance processes and so is, in a stochastic sense, bias free. This makes it more robust than the Euler-Maruyama method.

Considering the formulation of the model that was given earlier in the derivation of the Euler-Maruyama Monte Carlo method for the Heston model, we begin by integrating (3.39) and (3.40) so that

$$\begin{aligned} S_t &= S_0 \exp\left[rt - \frac{1}{2}\int_0^t V_u du + \rho \int_0^t \sqrt{V_u} d\widetilde{Z}_u^{(2)} \right. \\ &\quad \left. + \sqrt{1-\rho^2}\int_0^t \sqrt{V_u} d\widetilde{Z}_u^{(1)}\right] \qquad (3.46) \\ V_t &= V_0 + \kappa\theta t - \kappa \int_0^t V_u du + \sigma_v \int_0^t \sqrt{V_u} d\widetilde{Z}_u^{(2)}. \qquad (3.47) \end{aligned}$$

The variance process in the Heston model is the same as the interest rate model used by Cox et al. [21]. In their paper, they derive the distribution of this process at some time point $t > 0$, given that the value of the process is known at an earlier point, say time 0. If V_t and V_0 denote the values of the variance process at times t and 0 respectively, then the distribution of V_t given V_0 is a scaled non-central chi-square distribution such that

$$\begin{aligned} V_t &= \frac{\sigma_v^2(1-e^{-\kappa t})}{4\kappa}\chi_\gamma^2(\zeta) \\ \zeta &= \frac{4\kappa e^{-\kappa t}}{\sigma_v^2(1-e^{-\kappa t})}V_0 \\ \gamma &= \frac{4\theta\kappa}{\sigma_v^2}, \end{aligned}$$

where $\chi_\gamma^2(\zeta)$ denotes a non-central chi-square distribution with non-centrality parameter ζ and γ degrees of freedom.

Furthermore, Broadie and Kaya [11] state that if at time t, we know V_t then

$$\log S_t \sim N\left(\mu_{S_t}, \sigma_{S_t}^2\right), \qquad (3.48)$$

where,

$$\begin{aligned} \mu_{S_t} &= \log S_0 + rt - \frac{1}{2}\int_0^t V_u du + \rho\int_0^t \sqrt{V_u} d\widetilde{Z}_u^{(2)} \qquad (3.49) \\ \sigma_{S_t}^2 &= \left(1-\rho^2\right)\int_0^t V_u du. \qquad (3.50) \end{aligned}$$

Using these two distributions, we can sample values for the stock price and variance processes of the Heston model.

The complexity, as well as the computational "bottleneck", in this method is the computation of the integral

$$\int_0^t V_u du. \qquad (3.51)$$

Through the use of Laplace transform methods, Fourier inversion methods and the trapezoidal rule, Broadie and Kaya construct a method for estimating this integral. The characteristic function of the integral is given by (drawing directly from their paper):

$$\begin{aligned}\Phi(a) &= \mathbb{E}\left[e^{ia\int_0^t V_u du}\middle| V_0, V_t\right] \\ &= \left\{\frac{\gamma(a)e^{-0.5(\gamma(a)-\kappa)t}\left(1-e^{-\kappa t}\right)}{\kappa\left(1-e^{-\gamma(a)t}\right)}\right\} \\ &\quad \times \exp\left\{\frac{V_0+V_t}{\sigma_v^2}\left[\frac{\kappa\left(1+e^{-\kappa t}\right)}{1-e^{-\kappa t}} - \frac{\gamma(a)\left(1+e^{-\gamma(a)t}\right)}{1-e^{-\gamma(a)t}}\right]\right\} \\ &\quad \times \left\{\frac{I_{0.5d-1}\left[\sqrt{V_0V_t}\frac{4\gamma(a)e^{-0.5\gamma(a)t}}{\sigma_v^2\left(1-e^{-\gamma(a)t}\right)}\right]}{I_{0.5d-1}\left[\sqrt{V_0V_t}\frac{4\kappa e^{-0.5\kappa t}}{\sigma_v^2\left(1-e^{-\kappa t}\right)}\right]}\right\},\end{aligned} \tag{3.52}$$

where,

$$\gamma(a) = \sqrt{\kappa^2 - 2\sigma_v^2\, ia}$$
$$d = \frac{4\theta\kappa}{\sigma_v^2},$$

and $I_v(x)$ is a modified Bessel function of the first kind. Then, making use of the inverse Fourier transform and the trapezoidal rule, a discrete approximation of the probability distribution function of the integral can be found:

$$\begin{aligned}F(x) &= \text{Prob}\left[\int_0^t V_u du \leq x \middle| V_0, V_t\right] \\ &= \frac{2}{\pi}\int_0^\infty \frac{\sin(ux)}{u}\text{Real}\left[\Phi(u)\right]du \qquad (3.53) \\ &\approx \frac{hx}{\pi} + \frac{2}{\pi}\sum_{j=1}^{\infty}\frac{\sin(hjx)}{j}\text{Real}\left[\Phi(hj)\right] \\ &\approx \frac{hx}{\pi} + \frac{2}{\pi}\sum_{j=1}^{N}\frac{\sin(hjx)}{j}\text{Real}\left[\Phi(hj)\right]. \qquad (3.54)\end{aligned}$$

The main difficulty here is finding the best values of N and h to use so that we can obtain a good approximation for (3.53). In equation (3.54), there are two different types of errors to consider — the discretisation error, which is governed by our choice of h, and the truncation error, which is governed by our choice of N. We can thus write:

$$\begin{aligned}F(x) &= \frac{2}{\pi}\int_0^\infty \frac{\sin(ux)}{u}\text{Real}\left[\Phi(u)\right]du \\ &= \frac{hx}{\pi} + \frac{2}{\pi}\sum_{j=1}^{N}\frac{\sin(hjx)}{j}\text{Real}\left[\Phi(hj)\right] - \varepsilon_{\text{Disc}}(h) - \varepsilon_{\text{Trunc}}(N).\end{aligned}$$

Firstly, Broadie and Kaya [11] show that the discretisation error, $\varepsilon_{\text{Disc}}(h)$, is bounded below by 0, and above by $1 - F\left(\frac{2\pi}{h} - x\right)$. Setting $\xi_{\text{Disc}} = \frac{2\pi}{h} - x$, we can ensure that our discretisation error is bounded above by δ_{Disc} (where δ_{Disc} is a small positive number), by finding a value of h such that $h = \frac{2\pi}{x+\xi_{\text{Disc}}} \geq \frac{\pi}{\xi_{\text{Disc}}}$, where $\delta_{\text{Disc}} = 1 - F(\xi_{\text{Disc}})$ and $0 \leq x \leq \xi_{\text{Disc}}$. Actually solving for ξ_{Disc} is not trivial. Broadie and Kaya explain, however, that it is possible to find the moments of the distribution of (3.51) from its characteristic function and make ξ_{Disc} at least as large as the resulting mean plus five times the resulting standard deviation, to ensure that δ_{Disc} is "small enough". Next, since $|\sin(ux)|$ is bounded above by 1, and the characteristic function given by (3.52) is a monotonically decreasing function for increasing values of u, the integrand in (3.53) must always lie below $\frac{2|\text{Re}[\Phi(u)]|}{\pi u}$. Broadie and Kaya show that this, in turn, is bounded above by $\eta(u) = \frac{2|[\Phi(u)]|}{\pi u}$ and, since the integrand is an oscillating one, a good approximation of the truncation error is given by $\varepsilon_{\text{Trunc}}(N) = h\eta(Nh)$. Thus, to achieve a truncation error of δ_{Trunc}, we can select a value for N subject to the condition that $\frac{2|\Phi(hN)|}{\pi N} < \delta_{\text{Trunc}}$. Now, actually working out values for N and h in this way can be quite laborious. Instead, it can be easier to find these two parameters by trial and error. This is recommended by Broadie and Kaya and is also the way that we derive values for N and h. In our implementation of the exact simulation scheme, we pick $N = 800$ and $h = 0.5$.

The final step here is to actually sample from the integral. Setting $F(x) = U$, where U is a uniform random variable, we can use the inverse transform method to do so. MATLAB has a number of robust optimisation algorithms that can be used to find x — the sampled value of the integral.

Finally, we need to generate a sample from the integral,

$$\int_0^t \sqrt{V(u)}d\widetilde{Z}_u^{(2)}. \tag{3.55}$$

Since we have already found a value for (3.51), however, we can simply solve for (3.55) algebraically by rearranging (3.47). Having sampled values from the distributions of the integrals, (3.51) and (3.55), we can now obtain a random variate from the distribution of the log-stock price process, simply by generating a random number from a normal distribution with mean and variance given by (3.49) and (3.50).

In summary, the Broadie and Kaya exact simulation scheme for the Heston model can be carried out according to the following:

1. Generate the values V_0 and V_t from the scaled non-central chi-squared distribution of the variance process.

2. Randomly sample from the distribution of $\int_0^t V_u du$.
3. Solve for $\int_0^t \sqrt{V_u} d\widetilde{Z}_u^{(2)}$ from (3.47).
4. Generate a random value for the stock price process by sampling randomly from a normal distribution with mean (3.49) and variance (3.50) and taking the exponential of the resulting value.

Exact Simulation for the Bates Model

The exact simulation scheme for the Bates model follows a similar procedure to the Euler Monte Carlo method for the Bates model. Again, the jump and diffusion parts of the model can be evaluated separately and combined at the end. We start by simulating values for S_t^- (the value of the stock price before we include the jump part) and V_t using the exact simulation framework for the Heston model. To simulate the jump part of the stock price process, we simulate a Poisson process $\widetilde{N}_t$, with intensity λ, giving us the number of jumps, n, occurring between 0 and t. Next, we generate n log-normally distributed random variates with mean μ_S and variance σ_S^2. These give us the sizes of the jumps in the stock price process. Labeling the i^{th} jump size $\xi_i^{(S)}$ for $i = 1, \ldots, n$, our final stock price is given by

$$S_t = S_t^- \prod_{i=1}^{n} \xi_i^{(S)}. \tag{3.56}$$

Exact Simulation for the SVJJ Model

Again, we can extend the exact simulation scheme for the Bates model to that for the SVJJ model. The procedure that we follow is almost identical to that for the Euler-Maruyama method for the SVJJ model. We start by simulating a Poisson process, $\widetilde{N}_t$, with jump intensity λ to give us the number, n, of jumps occurring between times 0 and t. We denote the times at which the jumps occur by t_i, where $i = 1, \ldots, n$ and $0 \leq t_1 \leq \cdots \leq t_n \leq t$. For each interval $[t_{i-1}, t_i]$, we simulate the diffusion parts of the stock price and volatility processes according to the exact simulation scheme for the Heston model. This gives us preliminary values, $S_{t_i}^-$ and $V_{t_i}^-$, for our stock price and variance processes at t_i. Next, we simulate jump sizes for the jumps in the two processes in the same manner that we did in the Euler-Maruyama method for the SVJJ model. These jumps are given by $\xi_{t_i}^{(S)}$ and $\xi_{t_i}^{(V)}$ respectively. The final values of the two processes at time t_i can then be calculated:

$$S_{t_i} = S_{t_i}^- \xi_{t_i}^{(S)} \tag{3.57}$$

$$V_{t_i} = V_{t_i}^- + \xi_{t_i}^{(V)}. \tag{3.58}$$

Again, if $t \neq t_n$, then no jumps occur in the interval $[t_n, t]$ and we apply the method used for the exact simulation scheme for the Heston model to find the values of S_t and V_t.

3.3 A Comparison of Pricing Methods

Figure 3.2 gives a comparison of vanilla call pricing methods for the Heston, Bates and SVJJ models. In the plots, the flat red lines represent the FFT method prices for at-the-money vanilla call options (the parameter inputs are the same as for the synthetic data in the next chapter). It is evident in all three plots that the two Monte Carlo pricing methods converge to the FFT method price.

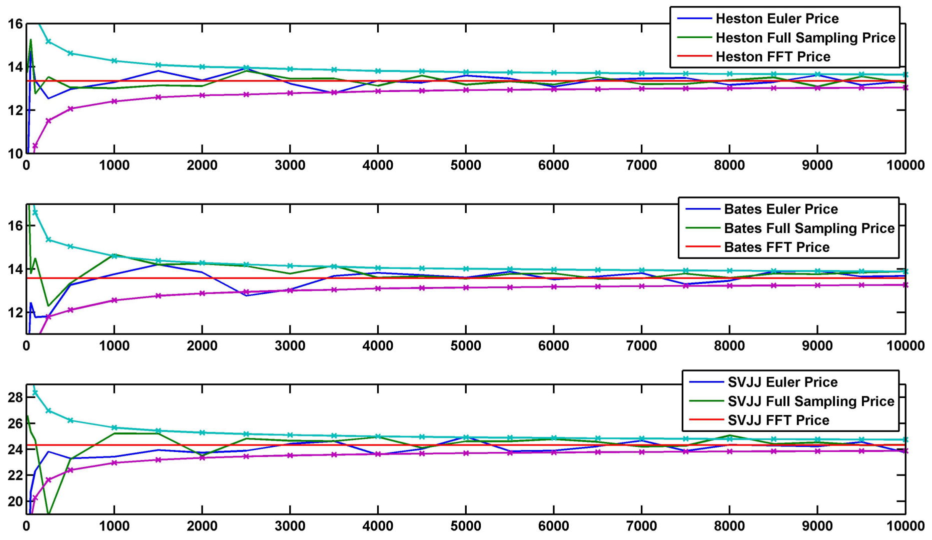

Figure 3.2 Comparison of the FFT, Euler Monte Carlo and Exact Simulation Monte Carlo methods for pricing vanilla European call options under the three models. On the horizontal axis, we have the number of sample paths and can see that both Monte Carlo techniques converge to the FFT price. The upper and lower bounds are the 95% confidence bounds.

Of particular interest, however, are the times taken for the respective methods to compute option prices. The FFT method simulation times were 0.011, 0.013, 0.015 seconds for the Heston, Bates and SVJJ models respectively. The table below sets out the results for the Monte Carlo pricing methods.

Monte Carlo Pricing Results — Simulation Times (seconds) and Errors						
Number of Paths		10	500	1000	5000	10000
Heston Euler	Simulation Time	~ 0	~ 0	0.02	0.06	0.09
	Deviation from FFT	−24.33%	−0.56%	3.09%	0.26%	1.19%
Heston Exact Simulation	Simulation Time	0.75	28.35	56.63	290.49	581.9
	Deviation from FFT	17.13%	4.54%	3.80%	0.98%	1.39%
Bates Euler	Simulation Time	0.02	0.02	0.02	0.11	0.22
	Deviation from FFT	29.65%	−3.63%	3.67%	−0.92%	0.83%
Bates Exact Simulation	Simulation Time	0.76	29.97	59.16	293.36	593.21
	Deviation from FFT	11.80%	−0.70%	5.73%	1.31%	0.01%
SVJJ Euler	Simulation Time	0.02	0.06	0.11	0.53	1.06
	Deviation from FFT	−23.30%	9.70%	0.52%	−2.41%	−1.48%
SVJJ Exact Simulation	Simulation Time	1.06	54.74	109.95	544.47	1082.66
	Deviation from FFT	26.14%	0.27%	−0.20%	−1.05%	−0.02%

For the purpose of pricing vanilla call options, the FFT method is much faster and more accurate than either of the two Monte Carlo methods. The problem arises, however, when we need to price exotic options. In this case, the FFT method does not (in general) provide us with a solution. Instead, we need to revert to Monte Carlo methods. As such, the two Monte Carlo methods that we have considered are extremely robust in that they provide a means by which to price almost any type of option. They are also useful in the computation of option price sensitivities, or “Greeks”.

Comparing the two Monte Carlo methods, it is clear that the Euler-Maruyama method is much faster than the exact simulation method. The exact simulation scheme is far more complicated than the Euler-Maruyama scheme and much of the computational “bottleneck” arises due to computation of the integral (3.51). The advantage of using this method, however, is that it is almost free from discretisation error. When using Monte Carlo methods as pricing tools, we need to be careful of two types of error: statistical error and discretisation error. Statistical error results from using simulation paths to estimate an average (e.g. an option price). This can be reduced through the application of the central limit theorem, by increasing the number of simulation paths. Figure 3.2 gives evidence of this phenomenon. Arguably of more importance, however, is the discretisation bias in a simulation scheme. By discretising a continuous process (such as any of the processes describing the models in this document), we open the scheme up to errors which cannot be reduced simply by increasing the number of sample paths in our simulation. Instead, we have to be very careful that the method we choose to use does not lead to a convergence to the wrong option price. This is particularly relevant when pricing exotic options, where we often have no other methods to check the accuracy of our final answer. In such situations, the use of the exact simulation scheme is more robust than the Euler-Maruyama method. It guarantees convergence to the correct option price as the number of sample paths is increased. Broadie and Kaya [11] provide a thorough analysis of these two types of errors, as well as a review of the Euler-Maruyama and exact simulation schemes for the Heston, Bates and SVJJ models.

A drawback of these Monte Carlo methods — especially the exact simulation scheme — is that they are computationally intensive to implement. We can, however, resort to parallel computing techniques to improve this.

3.4 Parallel Monte Carlo Methods for the Heston Model

An interesting area of research, particularly in terms of computational finance, is parallel computing. Parallelising computer code can produce great speed-ups for algorithms that would otherwise be labourious to implement. This is particularly relevant in the field of finance, due to constant time pressures in the practical implementation of financial algorithms.

We look briefly at the implementation of Heston Monte Carlo methods in parallel in MATLAB. The parallel computing toolbox in MATLAB is particularly convenient for this purpose. It facilitates the use of a number of commands for implementing code in parallel. Probably the most useful of these (at least for the purpose of Monte Carlo simulations) is the *parfor* routine that automatically implements for-loops in parallel.

In the table below, we show the results of the application of the *parfor* routine to Monte Carlo simulations in the Heston model. We see large speed-ups in the computation times for the exact simulation scheme for the Heston model as a result of parallelisation. The parallelisation here was implemented around the routine that samples from the integral (3.51), since this is where the major computational "bottleneck" lies in the algorithm. Interestingly, we actually see an increase in the simulation times for the Euler-Maruyama method for the Heston model. This results from altering our MATLAB code to allow for parallelisation. The original script for the Euler-Maruyama method makes use of partially vectorised code and a single for-loop going forward in time steps, to compute the sample stock price and volatility processes of the Heston model. To implement this code in parallel, we alter it to make use of two for-loops — one nested inside the other — with one computing across simulation paths and the other going forward in time steps. This allows us to parallelise the for-loop computing across simulation paths. Unfortunately, however, MATLAB handles the partially vectorised code better and can implement it faster than the parallelised code. Note that in running our simulations, we implement parallel code on the six cores of an Intel Core i7 processor. Increasing the number of cores would probably give us better results. Nonetheless, the speedups for the exact simulation method are significant and support the notion of investing time and resources into the parallelisation of Monte Carlo methods.

Parallel Monte Carlo Simulation Times (seconds) for the Heston Model

Number of Paths	Heston Euler	Parallel Heston Euler	Heston Exact Simulation	Parallel Heston Exact Simulation
10	~ 0	0.16	0.75	0.31
500	~ 0	0.08	28.35	5.58
1000	0.02	0.11	56.63	11.36
5000	0.06	0.55	290.49	56.78
10000	0.09	1.01	581.90	111.79

Chapter 4

Model Calibration

So far in this dissertation, we have reviewed a number of stochastic volatility models as well as some of the pricing techniques that can be used to produce option prices from these models. The calibration of these models to synthetic and market option data forms a major theme of this project and makes use of the techniques presented in the previous sections. Calibrating models to market data (either option prices or implied volatilities) allows us to infer the (risk-neutral) market parameters for the different models and thus use these models for pricing and hedging purposes. We do not consider fitting the models to historical data in this dissertation. This would be an interesting topic for a separate report.

One of the purposes of using complicated stock price models — specifically the ones we have considered so far — is to obtain better calibration fits to market data. This is particularly important for risk management and portfolio optimisation purposes. The cost of using such models, however, is that the calibration and pricing techniques that must be employed are usually quite onerous. The choice of a calibration routine thus requires a trade-off between its computational complexity and its accuracy. In this chapter, we present a least-squares calibration method and review local and global optimisation schemes for fitting the models to option data. We present some results obtained by fitting the models to both synthetic and market option prices. Throughout, we examine the merits and drawbacks of the routines, with reference to their accuracy and robustness, as well as to their complexity.

4.1 Least-Squares Optimisation

A well documented and popular method of fitting models to observed data is to find a set of model parameter values that minimises the square of the differences between the empirical values and the corresponding model values. In our case, this requires us to minimise the square differences of the option prices generated by each of our models and the option prices observed in the market. Note that we could also do this for model and market implied volatilities, however, this adds to the complexity of the calibration routine. Papers and books by Mikhailov and Nögel [43], Putschögl [49] and Zhu [59] give more insight into the application of the least-squares optimisation method to model calibration.

Suppose that we sample option data from N vanilla options in the market. Let Ψ_i, $i = 1, \ldots, N$, be the market price of the i^{th} option (either a call or a put option) and let $\widehat{\Psi}_i$ be the model price of the i^{th} option according to the model parameter set given by $\underline{\theta} \in \Re^n$. Then the sum of the square differences of the model and market prices is given by

$$\mathrm{SSD}\left(\underline{\theta}\right) = \sum_{i=1}^{N} w_i \left(\Psi_i \left(\sigma_i^{\mathrm{BS}}, T_i, K_i\right) - \widehat{\Psi}_i \left(\underline{\theta}, T_i, K_i\right) \right)^2, \tag{4.1}$$

where w_i is the weight given to the i^{th} squared-difference, σ_i^{BS} is the Black-Scholes implied volatility of the i^{th} option and T_i and K_i represent the time to maturity and strike price of the i^{th} option. Our calibration scheme now consists of finding the parameter set $\underline{\theta}^*$ that minimises the sum of squared differences:

$$\mathrm{SSD}\left(\underline{\theta}^*\right) = \min_{\underline{\theta}} \sum_{i=1}^{N} w_i \left(\Psi_i \left(\sigma_i^{\mathrm{BS}}, T_i, K_i\right) - \widehat{\Psi}_i \left(\underline{\theta}, T_i, K_i\right) \right)^2. \tag{4.2}$$

Another important consideration is the choice of weights w_i. One possible choice is to set $w_i = \frac{1}{N}$ for all $i = 1, \ldots, N$, making equation (4.1) a measure of mean squared errors (Zhu [59] follows this method). Alternatively, we could let $w_i = |\mathrm{bid}_i - \mathrm{ask}_i|^{-1}$ (as demonstrated by Moodley [45]). This would allow us to place more weight on options which are more liquid in the market. A third option that has also been suggested is to use the implied volatilities of the sampled options as weights (a method explored by Cont [19]). In this dissertation, we set the $w_i = \frac{1}{N}$ for simplicity.

The actual task of calibrating a chosen model to market data requires the use of optimisation techniques in order to find a model parameter set that minimises (4.1). We are dealing, however, with a non-linear problem and the function SSD $(\underline{\theta})$ is not convex, making the choice of an optimisation algorithm tricky. The function might also have many local minima or points which are not differentiable, making purely gradient based schemes ineffective and necessitating a careful choice of initial calibration parameters (Moodley [45]). As a result, we have to choose carefully between using a local or a global optimisation routine. Global optimisation schemes tend to be less sensitive to initial parameter estimates than local ones and should handle complicated objective functions better. They usually take longer to converge to a solution however.

4.2 Calibration Methods

Optimisation schemes can be broadly categorized into two groups — local and global schemes. Both have their merits and drawbacks. Local optimisation schemes tend to start from their initial parameter estimates and then choose new parameter estimates such that the value of the objective function always moves towards an optimum value. The schemes will continue until a stationary point is found, and as such, tend to locate local optimums rather than global ones. Simple gradient based methods are good examples of local optimisers. The choice of initial parameters in these schemes, especially in non-convex situations, is very important since a poor choice can cause the algorithm to settle in a local optimum instead of a global one. Their simplicity, however, tends to make them faster and easier to implement than global schemes.

Global optimisation schemes, on the other hand, are much less sensitive to initial parameter estimates. Ideally, they should even be independent of these. Many of these methods also fall into the category of stochastic optimisation schemes since they generate and use random variables to assist in locating an optimum. The two global optimisation schemes that we implement here are also stochastic optimisation schemes, since they use random variables to generate and accept new parameter values. This helps to prevent the algorithms from becoming "stuck" in the region of a local optimum, rather than a global one. Global schemes are usually significantly slower than local ones, thus their increased flexibility comes at a computational cost (see Mikhailov and Nögel [43] and Moodley [45] for a further discussion of local and global optimisation).

Below, we review three different optimisation schemes — the genetic algorithm (GA), adaptive simulated annealing (ASA) and the MATLAB least-squares non-linear optimisation routine, lsqnonlin. The first two schemes are global optimisation schemes, while the third is a faster local optimisation method. We implement these methods for the purpose of minimising the weighted sum of squared differences between market and model option prices.

4.2.1 Global Optimisation with the Genetic Algorithm

The primary optimisation routine that we implement is the genetic algorithm. The concept behind this algorithm is one of natural selection and evolution, where the stronger individuals of a population are selected over the weaker ones. The GA applies this to optimisation by evaluating how well individual points in a parameter space optimise the relevant objective function. These points, or individuals, are assigned fitness values based on how well they do this, and these fitness values are used to decide which individuals are allowed to "reproduce" in order to create subsequent population generations. By doing this, weaker members of the population start to die out and only those which best optimise the objective function survive. Selecting the fittest individual at the end of the algorithm should then allow the user to find the point at which the global optimum lies. We refer to the book by Coley [18] regularly in our treatment of the GA. The book provides a concise and informative overview of the algorithm and presents ways of implementing it, as well as numerous applications for the algorithm.

Consider the situation where we are trying to optimise an objective function with a given number of unknowns (or parameters). We do this by applying the GA to the problem and using it to find the values of the unknowns that achieve this. To initialise the algorithm, a population of N individuals is randomly chosen in the form of N strings of binary digits. The bit strings are all of equal length and each one can be thought of (in a biological sense) as the chromosome of the relevant individual. To proceed with the algorithm, the fitness of each individual in the initial population must be calculated. This is achieved by converting the bit string of each individual into real values representing possible solutions for the unknowns in the optimisation problem. These values are then substituted into the objective function and the corresponding bit strings are assigned fitness values based on how well they satisfy the optimisation task. From here, the individuals undergo selection, crossover and mutation in order that a new population of (hopefully) fitter individuals will emerge. The process of fitness evaluation, selection, crossover and mutation is then repeated over and over to create successive generations up to a prespecified maximum number of generations. The final generation should then contain the individual that provides a global optimum for the objective function. We explore the individual aspects of the algorithm in the paragraphs that follow. Works by Back et al. [2], Coley [18] and Putschögl [49] also give insight into the workings of the GA.

Population Initialisation. Suppose that we are trying to find the global optimum of an objective function with n unknowns (parameters). To generate a population of N individuals, we generate N random bit strings of length nl, where l represents the substring length that we assign to each unknown. Each individual in the population will then be characterised by a bit string of length nl. Assigning bit string lengths in this way makes it easy for us to convert between the bit strings and the real values of the unknowns. For example, if we are trying to optimise a function with two unknowns and we assign bit string lengths of 4 to each unknown, then two possible members of the initial population might be

$$\texttt{1 0 1 0 1 1 1 0} \qquad \text{and} \qquad \texttt{0 0 0 1 1 0 1 1}.$$

We can generate the entire population of the first generation by creating a matrix of dimensions N by nl with randomly arranged 0's and 1's.

Converting these to real values can be done in two steps. First, we convert the strings to integer values by dividing each string into n equal parts (representing each unknown) and performing a binary-to-decimal number conversion on each part. For the two example individuals considered above,

$$\texttt{1 0 1 0 1 1 1 0} \qquad \text{becomes} \qquad [10\ 14]\,,$$

and

$$\texttt{0 0 0 1 1 0 1 1} \qquad \text{becomes} \qquad [1\ 11]\,.$$

We then convert the integer values to real values through the use of

$$r \;=\; \frac{r_{\text{ub}} - r_{\text{lb}}}{2^l - 1} z + r_{\text{lb}},$$

where r is the real value, r_{ub} and r_{lb} are the upper and lower bounds associated with the unknown parameter and z is the integer value. If we have upper and lower bounds [3 4] and [4 6] respectively, then the values of the unknowns associated with the two individuals are:

$$\frac{4-3}{2^4-1}10+3 \;=\; 3.67 \qquad \text{and} \qquad \frac{6-4}{2^4-1}14+4 \;=\; 5.867 \qquad \text{for} \qquad \texttt{1 0 1 0 1 1 1 0}$$

and

$$\frac{4-3}{2^4-1}1+3 \;=\; 3.067 \qquad \text{and} \qquad \frac{6-4}{2^4-1}11+4 \;=\; 5.467 \qquad \text{for} \qquad \texttt{0 0 0 1 1 0 1 1}.$$

For improved accuracy in terms of the real values that can be generated from the bit strings, we should favour longer bit strings and narrower parameter bounds.

Fitness Evaluation and Scaling. The most important aspect about fitness evaluation in our treatment of the GA is to ensure that fitness values are always positive. The reason for this will become apparent when we consider the selection component of the algorithm. In our case, we are interested in least-squares optimisation, meaning that the objective function we deal with is a positive one. As a result, we can simply use the value of the objective function implied by each individual in the population (at each generation of the algorithm) to determine the fitness of that individual. Importantly, the GA automatically tries to find the global maximum of any objective function, since there is a positive relationship between the fitness value of a given individual, and the probability that that individual will be selected to "reproduce". If we are trying to minimise the objective function, we can simply subtract fitness values from some constant and turn the minimisation problem into a maximisation one. In our case, we define the fitness value of the i^{th} $(i = 1, \ldots, N)$ individual in a given generation of the algorithm to be

$$\text{Fitness}^{(i)} = C - \text{ObjectiveFunction}^{(i)},$$

where C is a constant and the value of $\text{ObjectiveFunction}^{(i)}$ is found by evaluating the objective function with parameter inputs given by the real values implied by the bit string of the i^{th} individual (as illustrated in the previous subsection).

Once we have determined the fitness values of all the individuals in a given generation of the algorithm, we can apply a linear scaling to each of the values. The purpose of this is to aid the selection component of the algorithm by preventing any individual from dominating the algorithm, or a large group of individuals from having very similar fitness values. The first instance might cause the algorithm to converge to a value which is not the global optimum of the objective function. The second could slow down the rate of convergence of the algorithm. As a result we increase or decrease the spread of fitness values in a given generation by considering the scaled fitness value of each individual. For the i^{th} individual we have

$$\text{ScaledFitness}^{(i)} = m \times \text{Fitness}^{(i)} + c,$$

where

$$\begin{aligned} m &= \frac{(k-1) \times \text{AveFitness}}{\text{MaxFitness} - \text{AveFitness}} \\ c &= (1-m) \times \text{AveFitness} \end{aligned}$$

and k is the scaling constant, AveFitness is the average fitness of the individuals in the given generation (prior to linear scaling) and MaxFitness is the maximum fitness value in that generation (also prior to linear fitness scaling). Performing the fitness scaling in this way means that the average fitness of the individuals will remain the same before and after scaling. Furthermore, the fitness value of the fittest individual after the scaling will simply be its fitness value before the scaling, multiplied by the scaling constant. This helps us to decide on the magnitude of the constant in each generation of the algorithm. We can now use these scaled fitness values in the selection component of the algorithm.

Selection. The selection component of the GA is the part where we select certain individuals to "reproduce" and discard the rest. The selection routine is based on the scaled fitness values of the individuals in the current generation of the algorithm. An individual with a higher scaled fitness value is assigned a greater probability of being selected than one with a lower scaled fitness value. We can achieve this by selecting individuals in the following way:

1. Generate a random number between 0 and the sum of the scaled fitness values of all the individuals in the population.

2. Starting with the first individual in the population, find the cumulative sum of the scaled fitness values of all the individuals in the population.

3. Select the bit string of the first individual responsible for making the cumulative sum larger than the random number generated in step 1.

As a consequence of performing selection in this way, we cannot allow fitness values to be less than 0. We select two individuals at a time in this manner, allow these two to "reproduce", and then repeat the selection procedure until as many individuals have been selected as the number of individuals in the original population. Obviously, this method allows a single individual to be selected more than once.

Crossover. Crossover is the section of the algorithm where selected individuals "reproduce". It is convenient to perform the crossover immediately after each set of two individuals has been selected. As a result, the selection and crossover components of the algorithm are usually performed simultaneously. To perform crossover, we first sample a random integer between 1 and the bit string length of the individuals in the population. This random integer gives us the point in the bit strings of two selected individuals where the crossover is to be performed. The actual crossover occurs by swapping the tails of the two bit strings after this point, thus creating two new individuals. For every pair of selected individuals we repeat this process (sampling a new random integer each time to perform the crossover). It is also common to occasionally prevent crossover from occurring and simply allow the two selected individuals to "reproduce" exact replicas of themselves. A probability is assigned to the occurrence of this.

For example, if the bit strings of two selected individuals are

`1 0 1 0 1 1 1 0`   and   `0 0 0 1 1 0 1 1`,

and the crossover point is 5, then the bit strings of the "reproduced" individuals would be

`1 0 1 0 1 0 1 1`   and   `0 0 0 1 1 1 1 0`.

Once selection and crossover have been performed, a new population of individuals arises and replaces the old population. The operations of mutation and elitism are then performed on this new population, after which, it is used to create the next generation of individuals in the algorithm, and so the process continues.

Mutation. The operation of mutation is a simple one. Considering the bit strings of all the individuals in the new population (after selection and crossover have been performed), we simply change each bit in the bit string of each individual from a 0 to a 1 or a 1 to a 0, with a certain probability. This probability is popularly given by 1 over the total bit string length of the individuals. Performing mutation adds an element of randomness to the algorithm and is another way to prevent the algorithm from "becoming stuck" at a local optimum.

Elitism. The final operation that we consider in our treatment of the GA is elitism. This operation is carried out by ensuring that the fittest individual across all generation remains in the population until the completion of the algorithm. We do this in each generation of the algorithm by simply checking if the fittest individual in the old population is fitter than the fittest individual in the new population. If this is the case, then we replace the bit string of a random individual in the new population by the fittest individual in the old population.

The GA is an effective method to use when conducting optimisation routines on objective functions which are non-linear, non-convex and multi-modal. This is especially true when comparing the GA to simpler, local optimisation methods under these conditions. A drawback of this routine is that it is computationally intensive. As a result, local optimisation methods tend to be much faster than the GA. The GA also does not guarantee that a global optimum will be found for the objective function, however it should at least find a point very close to the global optimum (see Ingber and Rosen [33] and Coley [18]).

Layout of the Genetic Algorithm

The routine followed to implement the genetic algorithm is as set out below.

1. Generate an N-by-nl matrix of randomly arranged 0's and 1's.
2. Convert the bit string of each individual into real numbers.
3. Evaluate the fitness of each individual.
4. Perform linear fitness scaling.

```
for Generations = 2 → Maximum Number of Generations
    (i) Perform selection and crossover.
    (ii) Perform mutation.
    (iii) Perform elitism.
    (iv) Evaluate the fitness of each individual.
    (v) Perform linear fitness scaling.
endfor
```

5. Select the fittest individual in the final generation.

Calibration With the Genetic Algorithm

The GA provides a robust method of calibrating the models already considered in this project to market data. We can then ascertain the parameters which best allow each model to explain market prices. The calibration of these models via the GA can be done by implementing the following routine:

1. The most important input for this algorithm is the data that is used for the calibration scheme. We need option price data consisting of option prices, along with the strikes and maturities of the respective options. In our algorithms in this project, we use option price data as opposed to implied volatility data. It is also important that we know the risk-free rate attached to each of our option prices, along with the dividend yields on the underlying stocks.

2. Input the necessary parameter constraints to prevent the algorithm generating values outside the parameter bounds.

3. After the first two steps, we are ready to pass all the required data to the GA routine.

   (a) Define `NoOfUnknowns` to be the number of unknowns in the objective function that we are trying to estimate. Define `LB` and `UB` to be the lower and upper parameter bounds respectively. Define the variable `minmax` to be `'MIN'`, indicating that we want to minimise the objective function.

   (b) Call the function `GeneticAlgorithm`, which allows us to implement the genetic algorithm in MATLAB:

   ```
   x = GeneticAlgorithm(CostFunction,NoOfUnknowns,LB,UB,minmax).
   ```

(c) The function `GeneticAlgorithm` in turn calls the function `CostFunction` where

$$\text{CostFunction} = \sum_{i=1}^{N} w_i \left(\text{MarketPrice}^{(i)} - \text{ModelPrice}^{(i)}\right)^2.$$

(d) Finally, the algorithm should converge to a model parameter set that minimises the cost function. This parameter set will be given as a vector output `x`.

A Note on the Implementation of the Genetic Algorithm

The MATLAB code for the genetic algorithm used in this project has been written by the author. Initially, we set out the algorithm in the exact manner described above, but encountered two problems with this specification. The first was that the method of selection and linear fitness scaling that we used frequently allowed one individual to dominate the algorithm, thus hampering the ability of the algorithm to find the globally fittest individual. The second problem was that the algorithm would come close to, but not actually settle at the global minimum. This last issue is one which plagues the genetic algorithm in general — it does not guarantee convergence to the global minimum of a system.

The first problem was overcome by adapting the method of selection. Instead of using the method described above, we combined three selection methods. The first of these automatically selects the fittest individuals in the population (usually the top 10% of the population) to progress to the next generation. The second is a "tournament" selection method, which randomly groups members of the population together (five members in our implementation), arranges them according to each of their fitness levels and then probabilistically chooses one individual to advance to the next generation. The fittest individuals obviously have a greater chance of advancing than the weakest ones. In our implementation, this method was used to generate around 80% of the new population. Finally, the third method randomly generates new individuals to complete the new generation. This adds randomness and diversity to the selection routine. We found that these three methods together provided a better selection routine for the genetic algorithm and they prevent any individual from dominating the population. This routine also eliminates the need for linear fitness scaling.

The second problem was overcome by selecting the parameter sets implied by the five fittest individuals at the end of the GA routine and subjecting these to the least-squares optimisation routine in MATLAB (lsqnonlin). This allows us to hone the results of the genetic algorithm and ensure that the parameters produced by the optimisation routine do, in fact, lie at a minimum. Using more than one of the fittest individuals helps to ensure that the algorithm does not settle in a local minimum.

4.2.2 Global Optimisation with Adaptive Simulated Annealing

Adaptive simulated annealing (ASA) is a global optimisation scheme which was developed by Lester Ingber in the early 1990's. It is arguably the most efficient of a number of different simulated annealing (SA) schemes. The C-language code that can be used to implement it is freely available on Lester Ingber's homepage (see Ingber [30]). In addition, it is possible to implement this routine in MATLAB thanks to a freely available function, ASAMIN, written by Shinichi Sakata (see Moins [44], Sakata [50]). This function allows MATLAB to interface directly with the C-language code and even allows MATLAB to change some of the options in the ASA routine.

Simulated Annealing

Simulated annealing arose as a Monte-Carlo style optimisation scheme in 1983 to deal with optimisation problems which involved highly nonlinear objective functions (Ingber and Wilson [34]). The name of this algorithm is derived from the process of the annealing of materials. Physically, this involves using heat-treatment in order to change the properties of some solid material (often some type of metal) so that it can serve a specific purpose. The solid is usually heated to very high temperatures and then cooled according to a specific cooling (or temperature) schedule in order to achieve the desired result (e.g. to increase the hardness of a metal so that it can be used to manufacture swords). SA works in the same manner in that it has a "temperature" parameter which controls the search area of the algorithm. Initially, this parameter will be set quite high, permitting the algorithm to explore much of the objective (or cost) function surface in its search for a minimum value. As the "temperature" parameter is lowered, the algorithm is forced to settle in a specific region of the cost function and ultimately, converge to a minimum value. It has been shown that the SA algorithm is statistically guaranteed to find a global minimum for the objective function as long as the temperature schedule is carefully controlled. This is, however, not guaranteed to occur in a finite amount of time (see Ingber and Wilson [34], Moins [44]).

The procedure followed by the SA algorithm can be laid out as follows (see Moins [44]):

1. Initialise the algorithm by stipulating the initial value of the temperature parameter and providing an initial guess for the parameter values of the cost function.

2. Use the chosen starting point of the algorithm to calculate the initial value of the cost function.

3. Generate a random step size for the algorithm (randomly generate new parameter values for the cost function) and calculate the new value of the cost function.

4. Subtract the new value of the cost function from the current value:

   IF: CurrentCost − NewCost > 0, then accept the new state of the cost function (i.e. accept the new parameter values for the function).

   ELSEIF: $\exp\left\{\frac{\text{CurrentCost}-\text{NewCost}}{\text{Temperature}}\right\} > \text{Uniform}\,(0,1)$ accept the new state of the cost function.

   ELSE: reject the new state of the cost function.

5. Decrease the "temperature" parameter according to the cooling schedule.

6. Exit the optimisation scheme if it has converged to a minimum. Otherwise repeat the routine from step 3.

Evaluating the steps above gives a good indication of how the algorithm works and how it can be used to find a global minimum value for the cost function. Firstly, the scheme is not always forced to move downwards towards the nearest "trough" in the objective function. It permits itself to make upward movements, away from any minimum values, with a certain probability. This probability is controlled by the "temperature" parameter: when the "temperature" is high, the probability that the algorithm accepts an upward movement is close to one. As this parameter is decreased, so the algorithm settles in a specific region of the objective function. This essentially allows the algorithm to "jump around" the space occupied by the function until it can settle in a region where the global minimum lies. Secondly, the generation of new parameter values depends on the value of the "temperature" parameter and hence, the area of the objective function that the optimisation routine is permitted to explore can be decreased by lowering this parameter. The cooling schedule attached to the "temperature" parameter is consequently very important to the success of the algorithm. In the case of SA, this is manually controlled by the user, making it very difficult for the user to obtain the fastest convergence to a minimum value. As a result, faster versions of the algorithm were developed. In 1987, the introduction of fast annealing (FA) made it possible to statistically guarantee finding the optimum solution for a system in a finite amount of time. Later in the same year, very fast simulated reannealing (VFSR) was developed and ultimately became known as adaptive simulated annealing. This provided further, significant decreases in the computational time of the SA procedure (see Ingber and Wilson [34], Moins [44], Moodley [45]).

Adaptive Simulated Annealing

As previously stated, the purpose of ASA is to provide a global optimisation scheme for non-linear, non-convex objective functions which occupy an N dimensional space. ASA does this by decreasing the "temperature" parameter of the i^{th} unknown in the system, $\Upsilon^{(i)}$, according to the schedule

$$\Upsilon_{t_k}^{(i)} = \Upsilon_{t_0}^{(i)} \exp\left(-c_i k^{\frac{1}{N}}\right), \tag{4.3}$$

where $i = 1, \ldots, N$, t_k refers to annealing time k and the parameter c_i is used to help adapt the algorithm to specific problems. Doing so automates much of the SA algorithm and allows for a fast convergence of the algorithm to a statistically global minimum (see Ingber [31, 32], Ingber and Wilson [34]).

Now, the ASA algorithm uses this parameter to sample new points in the N-dimensional parameter space as follows. Consider a parameter $\theta_{t_k}^{(i)}$ with a range $\left[\alpha^{(i)}, \beta^{(i)}\right]$ in the i^{th} parameter dimension and at annealing time k. The value of this parameter at annealing time t_{k+1} can be generated according to

$$\theta_{t_{k+1}}^{(i)} = \theta_{t_k}^{(i)} + z_{t_k}^{(i)} \left(\alpha^{(i)} - \beta^{(i)}\right), \tag{4.4}$$

where $z_{t_k}^{(i)}$ is a random variable lying between -1 and 1. Ingber and Wilson [34] specify a distribution for $z_{t_k}^{(i)}$ in their paper. If we denote the cumulative density function of $z_t^{(i)}$ at some time t by $F_t^{(i)}(z)$, we can sample from the distribution of $z_t^{(i)}$ by applying the inverse transform method. Here, we set $F_t^{(i)}(z) = u_t^{(i)}$, where $u_t^{(i)}$ is distributed Uniform $(0, 1)$, and solve for z. Ingber and Wilson [34] show that ASA samples from the distribution of $z_{t_k}^{(i)}$ at annealing time k according to:

$$z_{t_k}^{(i)} = \operatorname{sign}\left(u_{t_k}^{(i)} - \frac{1}{2}\right) \Upsilon_{t_k}^{(i)} \left[\left(1 + \frac{1}{\Upsilon_{t_k}^{(i)}}\right)^{\left|2u_{t_k}^{(i)} - 1\right|} - 1\right]. \tag{4.5}$$

As illustrated above, the key aspect of the simulated annealing and the adaptive simulated annealing schemes is the control of the "temperature" parameter. This parameter influences much of how the algorithm functions and an adequate schedule for it is vital in order to achieve a successful optimisation result. Other powerful options that ASA incorporates into its optimisation routine are reannealing and quenching. Reannealing essentially allows the annealing-time specified by k to be rescaled according to the sensitivity of the parameters of the objective function. This gives the algorithm more flexibility in how it samples the parameter space. Quenching allows for the "temperature" parameter to be quickly cooled in order to focus the algorithm on a specific region of the objective function and consequently speed up the optimisation task. Quenching can, however, have the negative effect of preventing the algorithm from finding the true global minimum. Such options can be selected according to the user's requirements. This makes ASA a robust optimisation routine and one that we make use of in our calibration algorithms. Further documentation on the ASA is provided in Ingber [30].

ASAMIN

The ASAMIN programme developed by Shinichi Sakata (Sakata [50]) allows MATLAB to interface with the C-language ASA code. This is most convenient for the purposes of this dissertation as it allows us to use ASA to minimise the sum of squared market-model differences in our calibration routines. Calling ASAMIN in MATLAB is achieved by including the following command in the optimisation script:

```
[fout,xout,grad,hess,exit] = asamin('minimize',fun,x0,LB,UB,type),
```

with

`fun`	-	The objective function.	`fout`	-	Value of the objective function at `xout`.
`x0`	-	Initial parameter estimates.	`xout`	-	Output vector.
`LB`	-	Lower parameter bounds.	`grad`	-	Gradient of the objective function at `xout`.
`UB`	-	Upper parameter bounds.	`hess`	-	The Hessian of the objective function at `xout`.
`type`	-	A vector of +1's and -1's indicating integer or real outputs.	`exit`	-	Exit state of the algorithm.

Calibration With Adaptive Simulated Annealing

We are now in a position to use the ASA method for the purpose of model calibration to option data. The ASA method is robust, but time-consuming. It can be implemented by following the steps below.

1. As with the GA method, the most important input for this scheme is the option price data. Again, we require option prices, along with maturities, spot prices of the underlying, the risk-free rate of return and the dividend yield of the underlying.

2. We decide on a starting point for the model parameters. The choice of starting parameters in the case of the ASA method is not as delicate as that for local optimisation schemes. Nonetheless, a good starting point will allow the method to converge faster.

3. After the first two steps, we are ready to pass all the required data to the optimisation routine.

   (a) Denote the initial parameter estimates by `x0` and the upper and lower parameter bounds by `UB` and `LB`.

   (b) Call the ASAMIN function in MATLAB:

   ```
   xout = asamin('minimize','CostFunction',x0',LB',UB',-[1,...,1]).
   ```

   (c) The ASAMIN function in turn calls, and attempts to minimise, the function `CostFunction` where

   $$\text{CostFunction} = \sum_{i=1}^{N} w_i \left(\text{MarketPrice}^{(i)} - \text{ModelPrice}^{(i)}\right)^2.$$

   (d) Finally, the algorithm should converge to a model parameter set that minimises the cost function. This parameter set will be given as a vector output `xout`.

4.2.3 Local Optimisation with MATLAB lsqnonlin

MATLAB has a number of optimisation routines built into its Optimisation Toolbox. An effective one for our purposes is the least-squares non-linear optimisation routine, lsqnonlin. As its inputs, it takes a vector function (the sum of squares of which is to be minimised) initial estimates for the function parameters as well as upper and lower parameter bounds. The output is then the set of parameters which minimises the sum of squares. Importantly, this routine is a local optimisation scheme and is sensitive to the initial parameter estimates. This makes it a difficult routine to implement as a starting point for the algorithm is tricky to select. The algorithm implemented by lsqnonlin by default, is the trust-region-reflective method. Alternatively, it is possible to instruct the optimisation routine to use the Levenberg-Marquardt method. More information on the algorithms used by lsqnonlin can be obtained in the MATLAB literature (The MathWorks [56]).

Calibration With MATLAB lsqnonlin

The routine followed to implement a calibration routine with lsqnonlin is much like that for the previous two routines. We implement it as follows:

1. Collect the same option data as for the previous two methods.

2. Like the ASA routine, lsqnonlin requires the input of initial parameter estimates. Since it is a local optimisation scheme, however, the choice of these initial parameter

values is important. Poorly chosen starting points will cause the algorithm to converge to the incorrect answer.

3. After the first two steps, we are ready to pass all the required data to the lsqnonlin routine.

    (a) Let `x0` be the initial parameter input vector. Define `UB` and `LB` to be the upper and lower parameter bounds.

    (b) Call the lsqnonlin function:

    ```
    x = lsqnonlin(CostFunction,x0,LB,UB).
    ```

    (c) In a similar, but not identical way to the previous routines, the lsqnonlin algorithm in turn calls the vector function `CostFunction`, where the i^{th} entry in the vector is given by

    $$\text{CostFunction}^{(i)} \;=\; w_i \left(\text{MarketPrice}^{(i)} - \text{ModelPrice}^{(i)}\right).$$

    The optimisation routine then computes the sum of squared model-market differences implicitly.

    (d) Finally, the algorithm should converge to a model parameter set that minimises the cost function. This parameter set will be given as a vector output `x`.

4.3 Calibration Results Using Synthetic Data

We turn our attention now to the application of the above methods to the calibration problems for the Heston, Bates and SVJJ models. To start with, we test our methods on synthetic data. In order to generate this data, we need to devise our own model parameters for the three models, use the fast Fourier transform pricing method to infer option prices from the models and then calibrate the models to this pseudo-market data. This seems like a rather "round-about" way of testing our calibration schemes. The purpose of doing this, however, is to examine the speed and efficiency of the different calibration methods in fitting the models to data which we know they ought to fit perfectly. Unlike the case where real market data is used and we are uncertain about which model provides the best fit to the data, or what the parameters of that model should be, this method gives us something to aim at in our calibration routines. It also gives us good perspective about how sensitive the schemes are to their initial inputs. We create the data by using model parameters[1] as specified in the following sections as well as maturity dates ranging from 0.25 years to 4

[1]Parameter values were chosen to be similar to those commonly seen in the literature.

years in quarterly intervals and 25 strikes for each maturity. We keep the strikes constant across all maturities. Some of the inspiration for this section is to extend the results of Moodley [45].

We also evaluate below how our calibration procedures can be sped up by fixing the rate of mean reversion in the volatility process. The motivation for doing this comes from Zhu [59], who gives reasons and methods for doing so. To start with, the model parameter κ is a very unstable parameter to fit. Setting it to some constant value can improve the stability of the optimisation algorithms and reduce the time taken for the calibration procedures. We examine the merits of doing so below.

4.3.1 Calibration of the Heston Model to Synthetic Data

Our synthetically generated data are obtained from the Heston model by using the FFT pricing method and the following parameters:

$$\begin{aligned} \kappa &= 1 \\ \theta &= 0.04 \\ \sigma_v &= 0.2 \\ \rho &= -0.3 \\ V_0 &= 0.04. \end{aligned}$$

Heston Calibration with the Genetic Algorithm

We start with the calibration of the Heston model to synthetically generated data. As explained above, we use the GA routine in conjunction with the MATLAB lsqnonlin procedure to ensure convergence to a global minimum. In all our implementations of the GA in this section we use the following algorithm settings:

- A population size of 300.
- A binary string length of 100 for each parameter.
- A total number of generations equal to 60.
- A crossover probability of 0.9.
- A mutation probability of 20/the total bit-string length of each individual.

In addition to this, each new generation is formed by selecting 70% of the new population by tournament selection, 20% by taking the fittest individuals of the current population and 10% by generating completely new individuals.

Below we see that the results of this calibration are very good[2]. In both cases, where κ is not fixed and where it is fixed, the calibration routine manages to converge to the true parameter set. Figure 4.1 gives a graphic depiction of the deviation of the prices produced by the calibrated parameter set from the original prices. We can see that there is very little deviation. The value of the objective function with the calibrated parameters as inputs is also very small, indicating convergence to a global minimum. Figure 4.2 depicts this convergence.

Heston GA Calibration			
	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	n/a	1.0001	-
θ	n/a	0.0400	0.0400
σ_v	n/a	0.2000	0.2000
ρ	n/a	-0.3000	-0.3000
V_0	n/a	0.0400	0.0400
Cost Function Value		1.4961×10^{-11}	1.4961×10^{-11}
Calibration Time		1090 seconds	1065 seconds

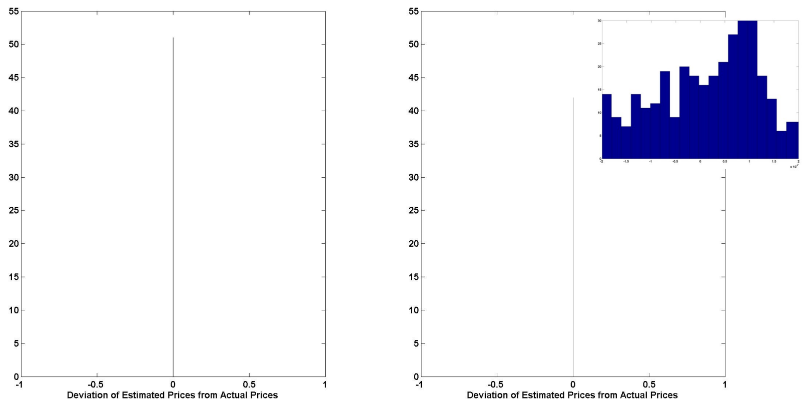

Figure 4.1 Histograms illustrating the fit of the Heston model to synthetic data using the GA routine. The plots show the deviation of the model implied prices from the original, synthetic data. The plot on the right illustrates the calibration performance when κ is held constant at 1. We can see that the GA and MATLAB lsqnonlin combination yields a good fit to the synthetic option prices. Note that the insert gives a magnified view of the histogram on the right and shows that there is some deviation from the market prices.

[2]Note that for the global optimisation schemes in this section, we ensure that the search range for each parameter is at least twice as large as the actual parameter value.

Holding κ constant at one, we see that the accuracy of the calibration does not change much. The time taken by the algorithm also remains much the same for both cases. As a result, fixing κ does not improve our calibration with the GA to the Heston model data. The times taken by the calibration routines indicate that this method is fairly computationally intensive.

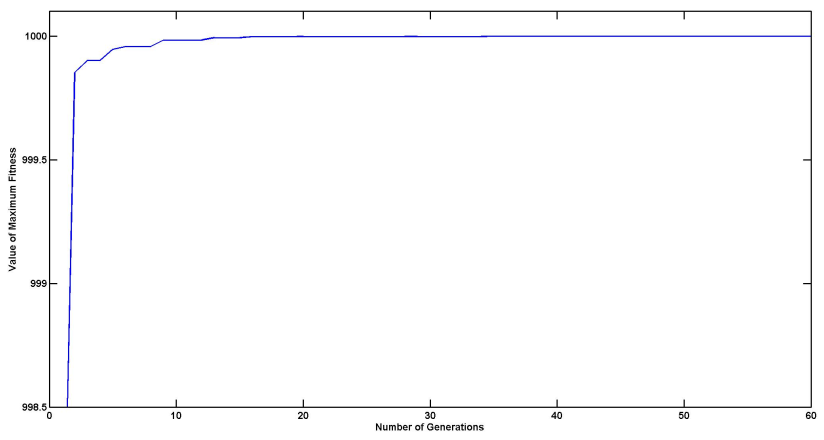

Figure 4.2 Plot showing the evolution of the fittest individual in the algorithm across all generations for the Heston model calibration with the GA routine. We can see that the fitness value of this individual approaches 1000 quickly. (Note that we calculate our fitness value for each individual here by subtracting the mean square differences of the model-market prices from 1000.) This plot gives further evidence of the convergence of the algorithm to a global minimum value.

Heston Calibration with Adaptive Simulated Annealing

The ASA routine is the second that we implement for the purpose of calibrating the Heston model to synthetic option data. Although the C-language ASA code has many options that can be set prior to implementation, many of these cannot be altered in MATLAB. This makes the MATLAB implementation of the optimisation scheme slightly limited. For the application of this method to the calibration problem in the Heston model, we set the initial parameter temperature value to 1000 and leave all the other ASA options at their default values. As a result, we implement a more general version of the ASA routine, rather than fine-tuning it to our specific situation.

In the tables and graphs below, we see good results for the scheme. For the set of parameter inputs, the method manages to converge adequately to the true parameter values associated with the synthetic data. Holding κ constant improves the calibration further by reducing the simulation time. Importantly, this method is not very sensitive to the initial parameter sets and so yields similar calibration results, irrespective of the inputs. This optimisation routine is even more computationally intensive than the GA method, although the fit is not quite as good as can be seen from the values of the cost function for the two calibrated parameter sets.

Figure 4.3 provides evidence of good calibration fits to the data. The final plot — Figure 4.4 — shows the convergence of the objective function to 0. This plot gives insight into how the ASA routine works — it allows the objective function to "jump around" before settling in an area close to the global minimum.

Heston ASA Calibration			
	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	1.5000	1.0067	-
θ	0.4000	0.0400	0.0400
σ_v	0.6000	0.2015	0.2011
ρ	-0.6000	-0.2986	-0.2978
V_0	0.4000	0.0400	0.0400
Cost Function Value		3.5233×10^{-6}	1.6825×10^{-4}
Calibration Time		3880 seconds	1150 seconds

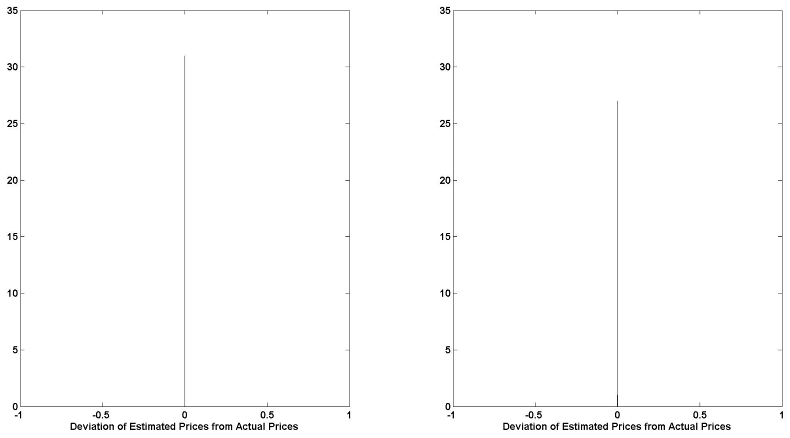

Figure 4.3 Histograms illustrating the fit of the Heston model to synthetic data using the ASA routine. We can see that the chosen starting points for the ASA calibration routine yielded good fits for the Heston model. This is to be expected, since ASA should not be very sensitive to starting points. Note that the plot on the right shows the calibration result when κ was fixed to 1.

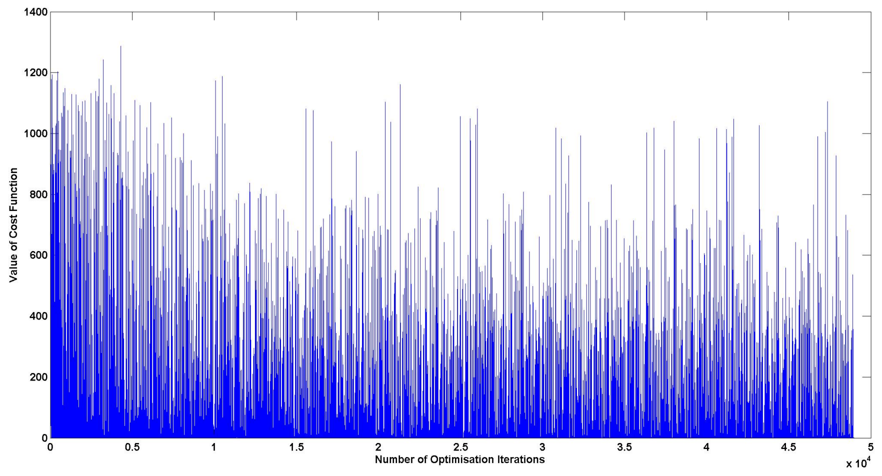

Figure 4.4 Plot showing the convergence of the objective function to a minimum during the calibration of the Heston model to synthetic data via ASA. This plot shows of how the ASA routine searches the parameter space before settling down at optimum point.

Heston Calibration with MATLAB lsqnonlin

The MATLAB lsqnonlin optimisation routine is the final one that we implement for calibration purposes for the Heston model. As mentioned earlier, it is a local optimisation scheme and as such can be highly sensitive to initial parameter inputs. In the case of the Heston model, we see good fits to the synthetic data. Many other initial parameter sets that we used yielded similar results. The relatively low dimensional scale of the problem means that the routine is quickly able to find the global minimum. Figures 4.5, 4.6 and 4.7 all give evidence of the success of this scheme. Notably, this routine is also much faster than the previous two — making it an attractive one to implement. Later on, however, we shall see that it suffers from over-sensitivity to initial inputs.

Heston MATLAB lsqnonlin Calibration (without fixing κ)				
	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
κ	1.5000	1.0001	4.2000	1.0000
θ	0.0900	0.0400	0.8500	0.0400
σ_v	0.1000	0.2000	0.0010	0.2000
ρ	−0.5000	−0.2999	−0.9000	−0.2999
V_0	0.0900	0.0400	0.9000	0.0400
Cost Function Value		4.8301×10^{-10}		2.0965×10^{-12}
Calibration Time		3.5 seconds		6.3 seconds

We also implement this scheme whilst fixing κ to the value one. The intention of doing so is to speed-up and improve the calibration of the model to the synthetic data. In the case of the Heston model, our calibration scheme is already successful without fixing κ. Nonetheless, doing so does decrease the calibration time and, as seen from the final values of the cost-function, improves the fit.

Heston MATLAB lsqnonlin Calibration (holding κ constant)				
	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
θ	0.0900	0.0400	0.8500	0.0400
σ_v	0.1000	0.2000	0.0010	0.2000
ρ	-0.5000	-0.3000	-0.9000	-0.3000
V_0	0.0900	0.0400	0.9000	0.0400
Cost Function Value		4.7781×10^{-13}		6.7364×10^{-13}
Calibration Time		3.3 seconds		5 seconds

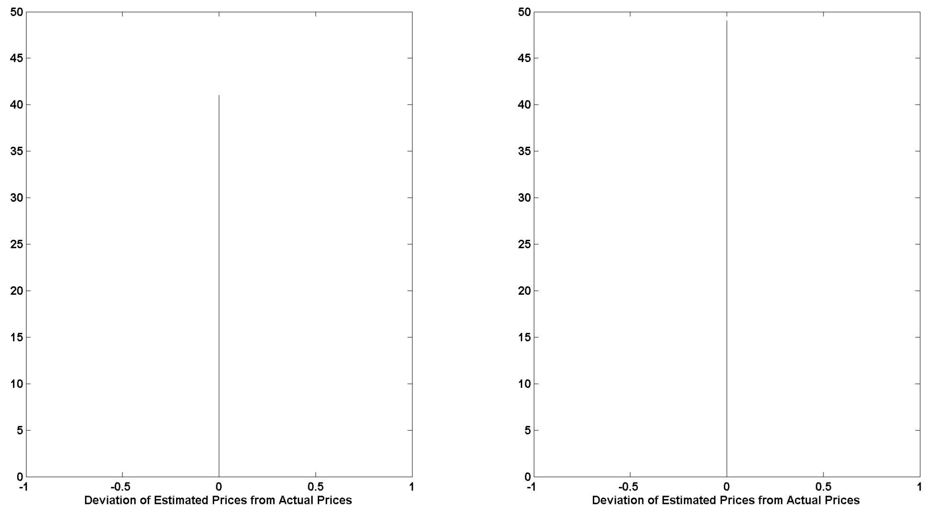

Figure 4.5 Histograms illustrating the deviation of the Heston model prices from the original, synthetic data after calibration via the MATLAB lsqnonlin routine (without fixing the value of κ). Both histograms show a good fit to the data.

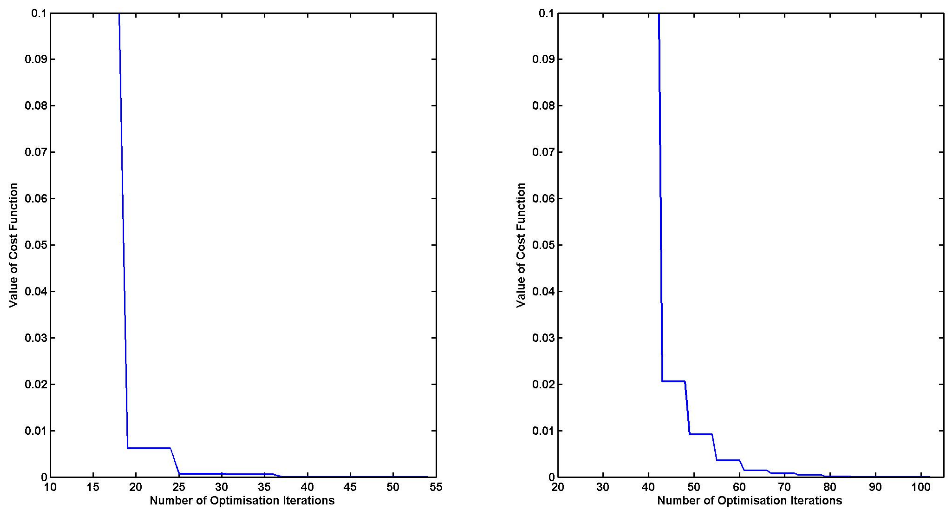

Figure 4.6 Plot showing the convergence of the objective function to a minimum value during the calibration of the Heston model to synthetic data via MATLAB lsqnonlin. Both plots show the objective function converging to 0, indicating that the optimisation routine has found the global minimum in both cases.

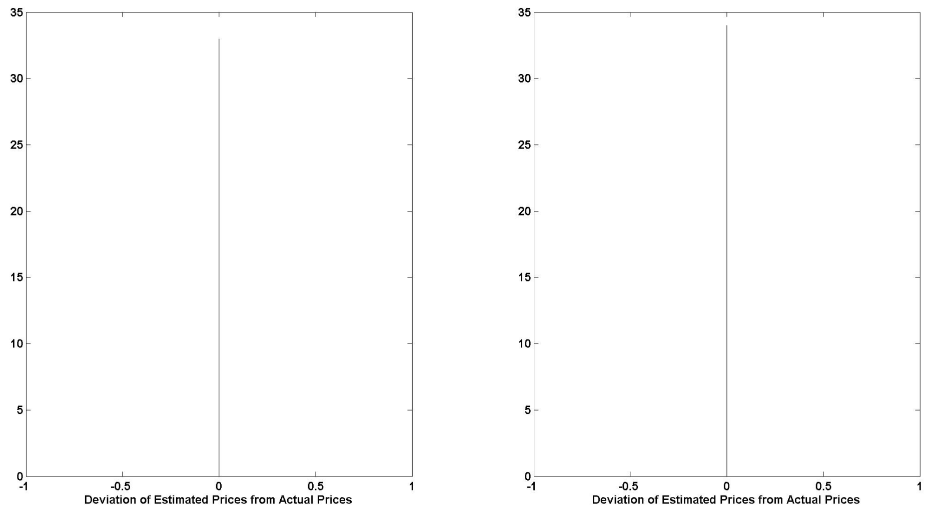

Figure 4.7 Histograms illustrating the deviation of the Heston model prices from the original, synthetic data after calibration via the MATLAB lsqnonlin routine. We have fixed κ to 1 during the calibration. Again, both histograms give evidence of a good calibration fit.

4.3.2 Calibration of the Bates Model to Synthetic Data

Our synthetically generated data are obtained from the Bates model by using the FFT pricing method and the following parameters:

$$\begin{aligned}
\kappa &= 1\\
\theta &= 0.04\\
\sigma_v &= 0.2\\
\rho &= -0.3\\
V_0 &= 0.04\\
\mu_J &= -0.12\\
\sigma_S &= 0.15\\
\lambda &= 0.11.
\end{aligned}$$

Bates Calibration with the Genetic Algorithm

As with the Heston model, we implement a calibration technique for the Bates model based on the GA. We use the same algorithm settings as we did in the case for the Heston model. Again, the tables, histograms and final cost function values below show evidence of successful calibrations to the synthetic data. The higher dimensionality of the problem does make the implementation time for the method slightly longer. The jump parameters are also more sensitive than the diffusion ones and, as a result, do not converge quite as well to their true values. Nonetheless, the GA proves to be a robust method to use for the calibration problem in the Bates model — especially since initial parameters are not required to start the algorithm. Holding κ constant has little effect on the calibration result.

Bates GA Calibration

	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	n/a	0.9865	-
θ	n/a	0.0403	0.0403
σ_v	n/a	0.1998	0.1987
ρ	n/a	-0.2916	-0.2949
V_0	n/a	0.0403	0.0404
μ_J	n/a	-0.1803	-0.1747
σ_S	n/a	0.1232	0.1352
λ	n/a	0.0749	0.0727
Cost Function Value		9.4088×10^{-8}	1.4794×10^{-7}
Calibration Time		1400 seconds	1320 seconds

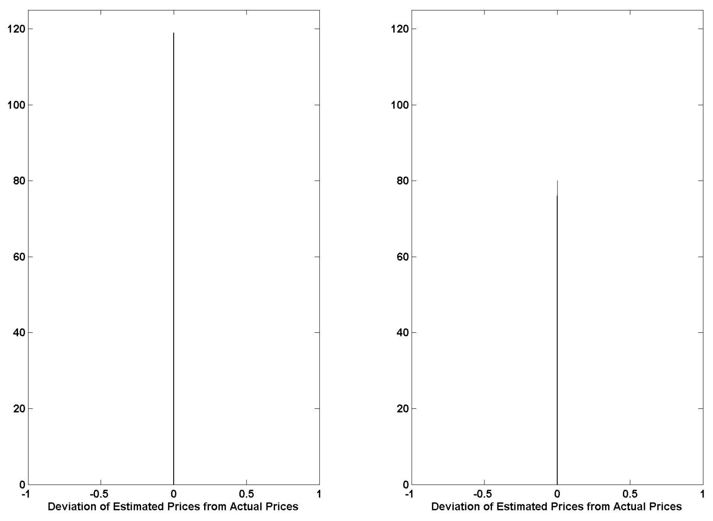

Figure 4.8 Histograms illustrating the deviation of the Bates model prices from the original, synthetic data after calibration using the GA. The histogram on the right shows the calibration result when κ is held constant at 1. We can see that the GA and MATLAB lsqnonlin combination yield a good fit to the synthetic option prices.

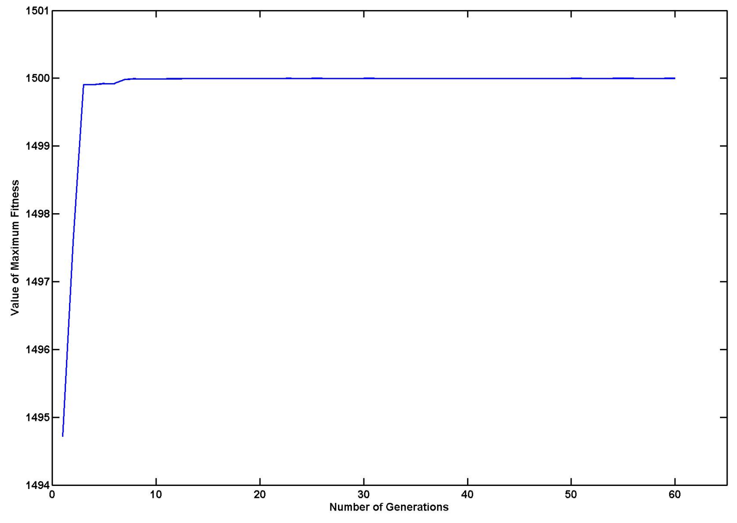

Figure 4.9 Plot showing the evolution of the fittest individual in the algorithm across all generations for the calibration of the Bates model to synthetic data via the GA. We can see that the fitness value of this individual approaches 1500 quickly. (Note that we calculate our fitness value for each individual by subtracting the mean square differences of the model-market prices from 1500). This plot gives further evidence of the convergence of the algorithm to a global minimum value.

Bates Calibration with Adaptive Simulated Annealing

Our results for the ASA calibration scheme for the Bates model are given below.

Bates ASA Calibration			
	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	2.5000	1.002	-
θ	0.2400	0.0382	0.0404
σ_v	0.1000	0.2364	0.2122
ρ	-0.4500	-0.2931	-0.2856
V_0	0.2400	0.0371	0.04
μ_J	-0.0700	-0.1174	-0.1637
σ_S	0.3000	0.0037	0.0948
λ	0.0500	0.4413	0.1031
Cost Function Value		8.1382×10^{-5}	5.897×10^{-6}
Calibration Time		8610 seconds	8690 seconds

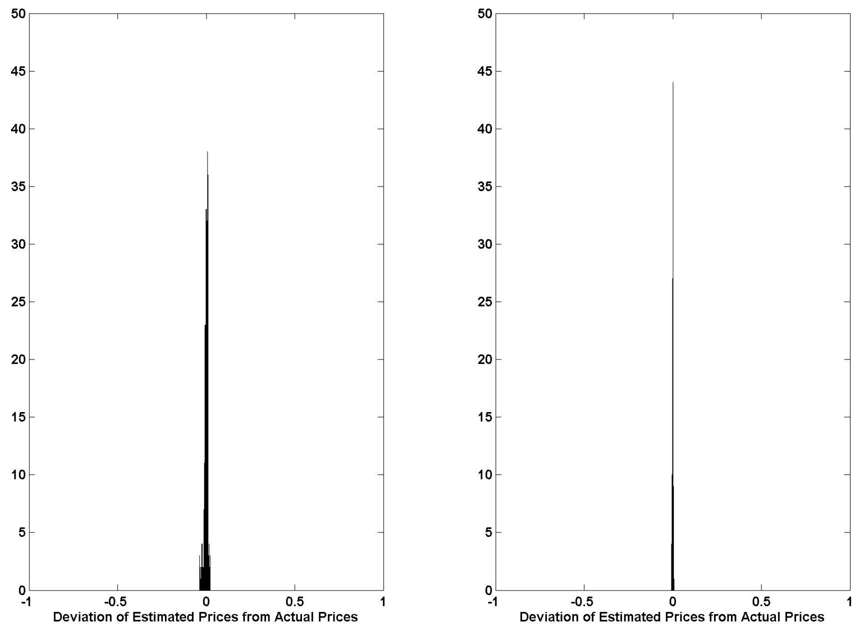

Figure 4.10 Histograms illustrating the deviation of the Bates model prices from the original, synthetic data after calibration via ASA. The plot on the right shows the result where κ was held constant at a value of 1. The two results indicate that while the ASA routine did not provide a poor fit to the data, it did not work as well as the GA calibration method did.

The results indicate a reasonable calibration fit, although not as good a result as that obtained using the GA. We tried changing the initial parameter temperatures in an effort to force the ASA algorithm to search the parameter space more thoroughly before settling down. The results below were obtained using an initial parameter temperature of 1000. Ideally, this method should easily locate the global minimum in the optimisation problem. This has not quite been the case for the Bates model. A more thorough investigation into the operations of the ASA routine as well as the implementation of the code in C would probably yield better results. Nonetheless, this routine is quite time consuming and not very practical to implement. Other initial parameter sets yielded similar results for the calibration routine.

Bates Calibration with MATLAB lsqnonlin

With the calibration of the Bates model to synthetic data via the MATLAB lsqnonlin routine, we see less convincing results compared to those for the Heston model. The first initial parameter set yields good calibration results, as can be see from the histograms and final cost function values below. The second initial parameter set, however, causes the routine to converge to a parameter set quite dissimilar from the true set. The failure of the MATLAB lsqnonlin routine to converge to a global minimum in this instance is shown by the second plot of Figure 4.12 and by the final value of the objective function for this calibration trial. These results begin to illustrate the sensitivity of the method to the initial parameter set that is passed to the routine. Nonetheless, the initial parameters that resulted in a breakdown of the method were rather extreme and for most initial sets chosen reasonably close to the true set, the scheme yielded a successful calibration. The scheme still provides a very fast way of calibrating the model to market data.

Bates MATLAB lsqnonlin Calibration (without fixing κ)

	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
κ	0.7000	0.9869	0.0050	0.4893
θ	0.0800	0.0402	0.9000	0.0259
σ_v	0.1900	0.1998	0.9000	1.0000
ρ	-0.2000	-0.2921	-0.9500	0.0000
V_0	0.0800	0.0402	0.9000	0.0126
μ_J	-0.2000	-0.1605	-0.0010	-0.1663
σ_S	0.2500	0.1328	0.0010	0.0000
λ	0.0600	0.0858	0.0010	1.0000
Cost Function Value		3.5604×10^{-7}		6.7505×10^{-2}
Calibration Time		16.1 seconds		45.4 seconds

Keeping κ fixed (i.e. excluding it from the calibration routine) can improve the calibration result (and speed) for poorly chosen initial parameters. We see this from the table below.

Bates MATLAB lsqnonlin Calibration (holding κ constant)

	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
θ	0.0800	0.0403	0.9000	0.0389
σ_v	0.1900	0.2016	0.9000	0.2251
ρ	-0.2000	-0.2927	-0.9500	-0.3081
V_0	0.0800	0.0403	0.9000	0.0379
μ_J	-0.2000	-0.1665	-0.0010	-0.1230
σ_S	0.2500	0.1285	0.0010	0.0072
λ	0.0600	0.0820	0.0010	0.3494
Cost Function Value		2.0549×10^{-7}		6.8103×10^{-5}
Calibration Time		8.3 seconds		22.2 seconds

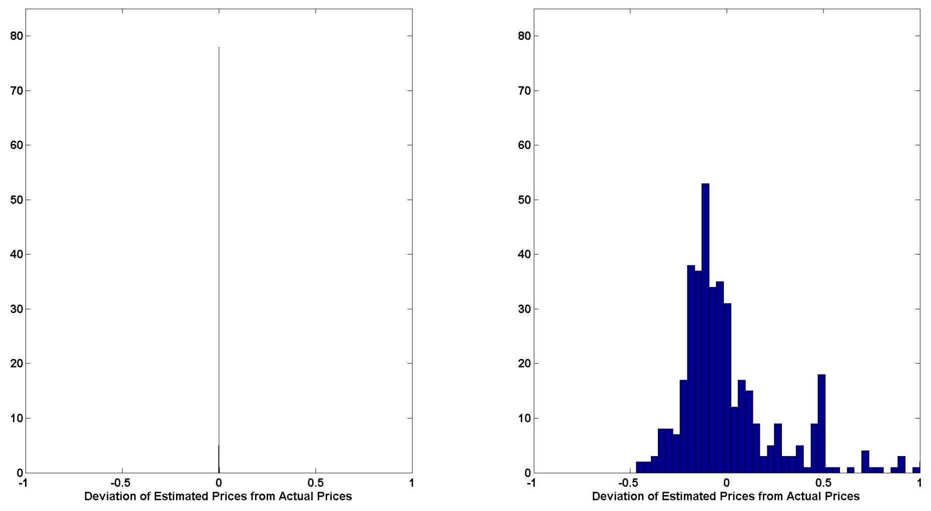

Figure 4.11 Histograms illustrating the deviation of the Bates model prices from the original, synthetic data after calibration via MATLAB lsqnonlin (without fixing the value of κ). The histogram on the left shows that the optimisation routine yielded a good fit to the data. The histogram on the right, however, gives evidence of a poor calibration result due to poorly chosen initial parameters.

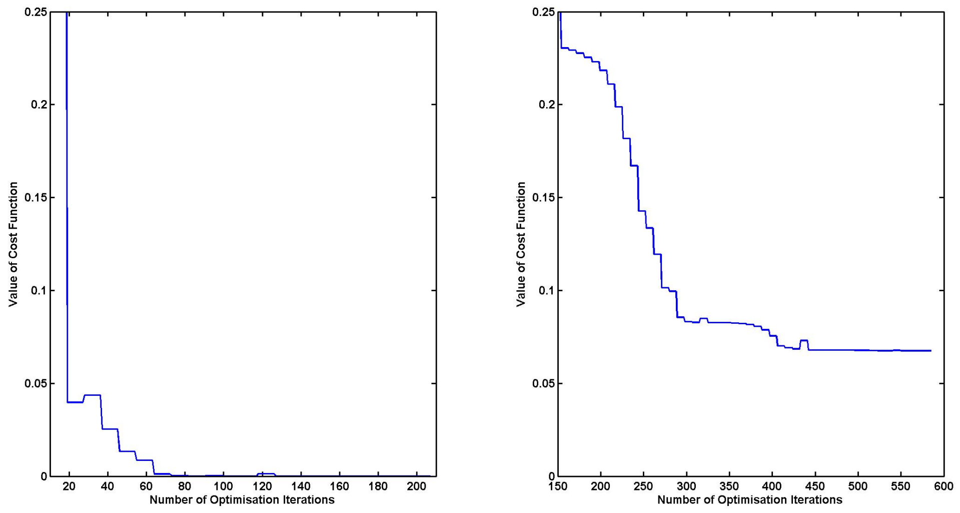

Figure 4.12 Plot showing the convergence of the objective function to a minimum value during the calibration of the Bates model to synthetic data via MATLAB lsqnonlin. The plot on the left gives evidence of convergence to a global minimum, whilst that on the right shows convergence only to a local minimum. Both plots show that the MATLAB lsqnonlin routine moves steadily downwards.

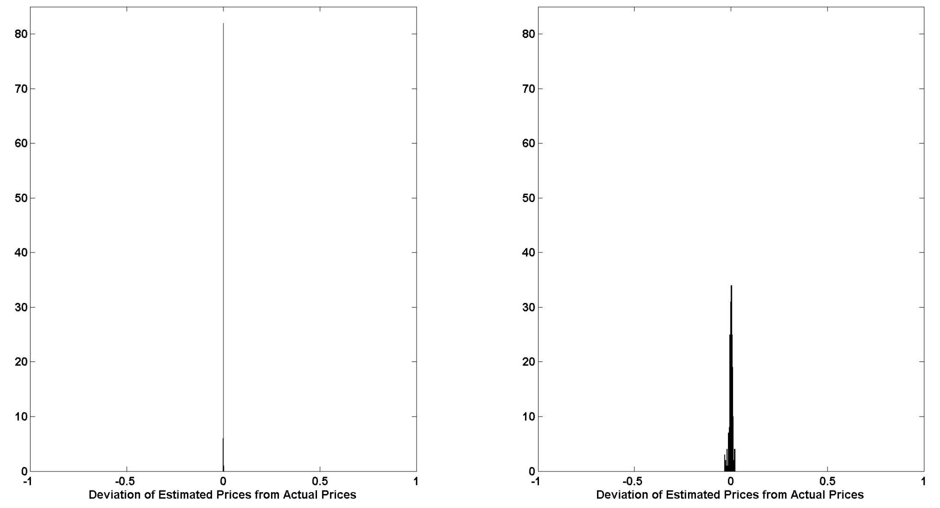

Figure 4.13 Histograms illustrating the deviation of the Bates model prices from the original, synthetic data after calibration via MATLAB lsqnonlin. We have fixed κ to 1 during the calibration. An improvement in the fit as a result of fixing κ is evident.

4.3.3 Calibration of the SVJJ Model to Synthetic Data

Our synthetically generated data are obtained from the SVJJ model by using the FFT pricing method and the following parameters:

$$\begin{aligned}
\kappa &= 3.5 \\
\theta &= 0.008 \\
\sigma_v &= 0.2 \\
\rho &= -0.8 \\
V_0 &= 0.008 \\
\lambda &= 0.5 \\
\mu_S &= -0.9 \\
\sigma_S &= 0.0001 \\
\rho_J &= -0.4 \\
\mu_V &= 0.05.
\end{aligned}$$

SVJJ Calibration with the Genetic Algorithm

As with the previous two models, the GA calibration method yields a good fit to the synthetic SVJJ model data.

SVJJ ASA Calibration

	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	n/a	3.4930	-
θ	n/a	0.0080	0.0080
σ_v	n/a	0.2030	0.2029
ρ	n/a	-0.7863	-0.7859
V_0	n/a	0.0080	0.0080
λ	n/a	0.5000	0.5000
μ_S	n/a	-0.8999	-0.8999
σ_S	n/a	0.0005	0.0010
ρ_J	n/a	-0.4000	-0.3992
μ_V	n/a	0.0500	0.0501
Cost Function Value		1.5921×10^{-9}	1.3181×10^{-7}
Calibration Time		1610 seconds	1520 seconds

The GA calibration routine has proved to be quite a robust method. It has provided good fits to all three data sets in reasonable time, without the need for any initial inputs.

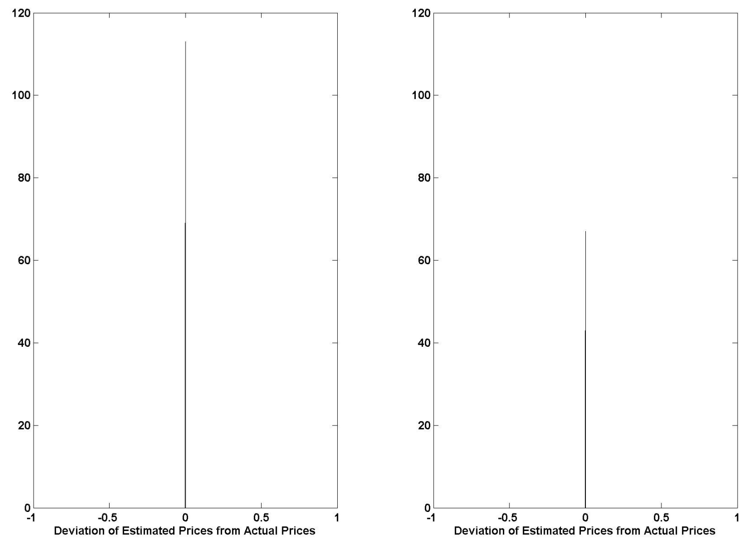

Figure 4.14 Histograms illustrating the deviation of the SVJJ model prices from the original, synthetic data after calibration via the GA. The histogram on the right shows the calibration result for κ held constant at 3.5.

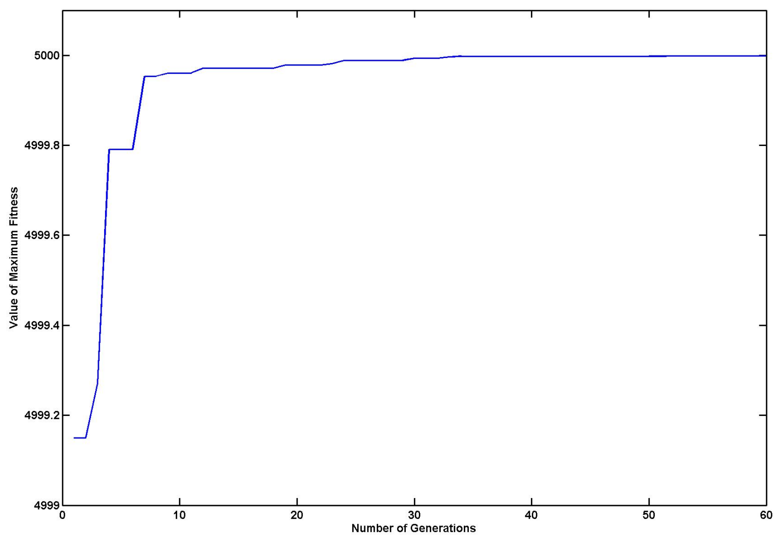

Figure 4.15 Plot showing the evolution of the fittest individual in the algorithm across all generations during the calibration of the SVJJ model to synthetic data via the GA. We can see that the fitness value of this individual gets very close to 5000 as the algorithm proceeds (note that we calculate the fitness value for each individual by subtracting the mean square differences of the model-market prices from 5000). This plot gives further evidence of the convergence of the algorithm to a global minimum value.

This has made it a very easy method to implement. Figures 4.14 and 4.15 show the success of the GA calibration routine in providing a calibration fit for the SVJJ model. The final cost function values also indicate that the calibration fit is a good one. As with the GA calibration routine for the Heston and Bates models, keeping κ constant does not provide a significant improvement in the calibration fit.

SVJJ Calibration with Adaptive Simulated Annealing

Our implementation of the ASA calibration routine for the SVJJ model yields similar results to those for the application of the method to the Bates model. Again, we set the initial temperature parameter of the ASA algorithm to 1000. The table and figure below show that the method is reasonably successful. It does not, however, outperform the calibration routing involving the GA. The higher dimensionality of the Bates and SVJJ models seems to reduce the effectiveness of the ASA optimisation scheme. As we did for the application of this method to the other two models, we can fix κ to a constant value throughout the calibration procedure. Setting κ to 3.5 (essentially removing it from the calibration routine) does provide an improvement to the calibration fit (as was the case for the Bates model).

SVJJ ASA Calibration			
	Initial Parameter Set	Parameter Output (without fixing κ)	Parameter Output (holding κ constant)
κ	2.5000	3.3808	-
θ	0.0030	0.0098	0.0071
σ_v	0.1000	0.3340	0.1694
ρ	-0.9500	-0.5595	-0.9319
V_0	0.0030	0.0075	0.0084
λ	0.7500	0.4980	0.5009
μ_S	-0.4500	-0.9057	-0.9025
σ_S	0.0003	0.0007	0.0009
ρ_J	-0.1500	-0.3969	-0.2817
μ_V	0.3500	0.0389	0.0585
Cost Function Value		1.0882×10^{-4}	1.8934×10^{-5}
Calibration Time		8638 seconds	10244 seconds

SVJJ Calibration with MATLAB lsqnonlin

Our implementation of the MATLAB lsqnonlin calibration routine yields similar results in its application to the SVJJ model as it did for the Bates model. Again, it is the fastest of the three calibration methods to implement. It suffers, however, from sensitivity to its initial parameter inputs. This can be seen by comparing the results for the first and second

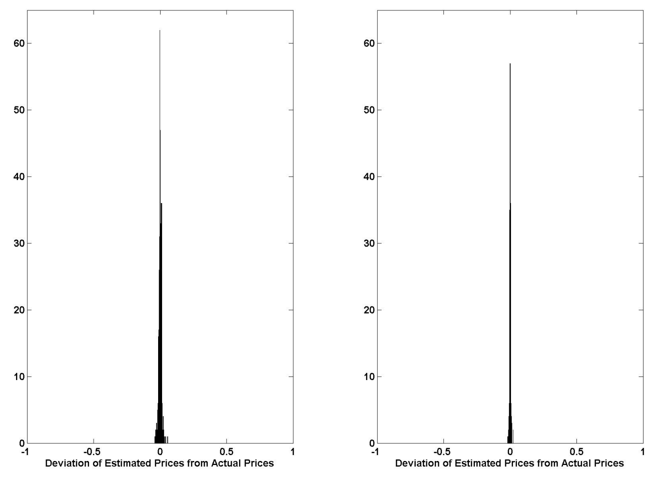

Figure 4.16 Histograms illustrating the deviation of the SVJJ model prices from the original, synthetic data after calibration via ASA. The plot on the left shows the calibration result without fixing κ, whilst that on the right shows the result where κ was held constant. The two results indicate that while the ASA routine did not provide a poor fit to the data, it did not work as well as the GA calibration method did.

initial parameter sets. The first set yields a good calibration fit, but the second set does not. Figures 4.17 and 4.18 give evidence of this, showing that the lsqnonlin optimisation routine only manages to locate a local minimum for this parameter set. The very high dimensionality of the problem means that many of the parameters, especially the jump parameters, are quite sensitive.

SVJJ MATLAB lsqnonlin Calibration (without fixing κ)

	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
κ	4.5000	4.0127	0.3000	0.4389
θ	0.0180	0.008	0.4080	0.0481
σ_v	0.4000	0.2285	0.9500	0.9999
ρ	-0.9000	-0.7546	-0.1000	0.0000
V_0	0.0180	0.0081	0.4080	0.0030
λ	0.6500	0.5000	0.9500	0.5013
μ_S	0.7500	-0.8935	-0.0500	-0.9040
σ_S	0.000095	0.0007	0.000095	0.0010
ρ_J	-0.2000	-0.4816	-0.9500	-0.0012
μ_V	0.2500	0.0554	0.9500	0.0142
Cost Function Value		3.2514×10^{-6}		3.39×10^{-2}
Calibration Time		19 seconds		85 seconds

Keeping κ constant throughout the calibration procedure does improve the fit as can be seen in the table below. Consequently, we find that this is not a good calibration routine to use for the SVJJ model, unless the value of κ can be fixed to an appropriate value and good initial parameters can be found.

SVJJ MATLAB lsqnonlin Calibration (holding κ constant)

	Initial Parameter Set 1	Parameter Output 1	Initial Parameter Set 2	Parameter Output 2
θ	0.0180	0.0080	0.4008	0.0080
σ_v	0.4000	0.2004	0.9500	0.2024
ρ	-0.9000	-0.7980	-0.1000	-0.7893
V_0	0.0180	0.0080	0.4080	0.0080
λ	0.6500	0.5000	0.9500	0.4999
μ_S	0.7500	-0.8998	-0.0500	-0.8990
σ_S	0.000095	0.0003	0.000095	0.0000
ρ_J	-0.2000	-0.4043	-0.9500	-0.4252
μ_V	0.2500	0.0499	0.9500	0.0497
Cost Function Value		1.5723×10^{-9}		4.6602×10^{-8}
Calibration Time		18 seconds		31 seconds

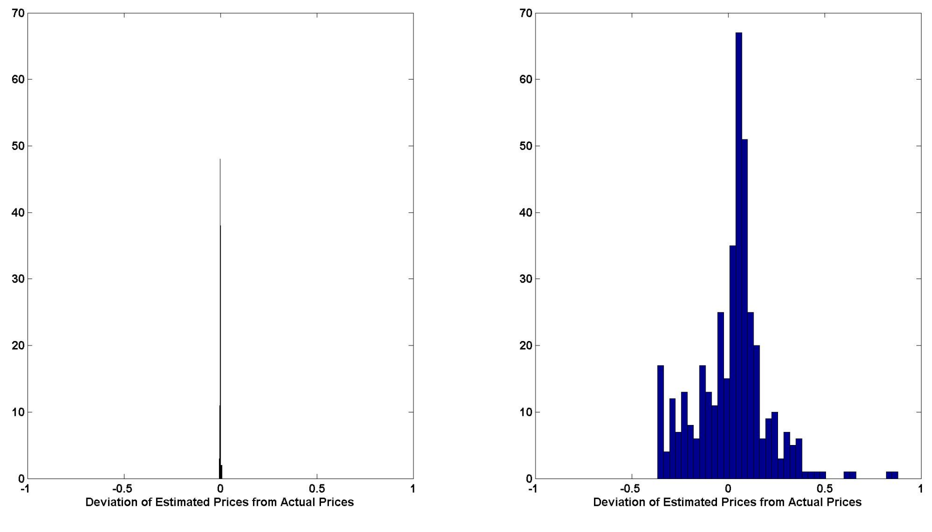

Figure 4.17 Histograms illustrating the deviation of the SVJJ model prices from the original, synthetic data after calibration via MATLAB lsqnonlin (without fixing the value of κ). The histogram on the left shows that initial parameters chosen for the optimisation routine provided a good fit to the data. The histogram on the right, however, gives evidence of a poor calibration result due to poorly chosen initial parameters.

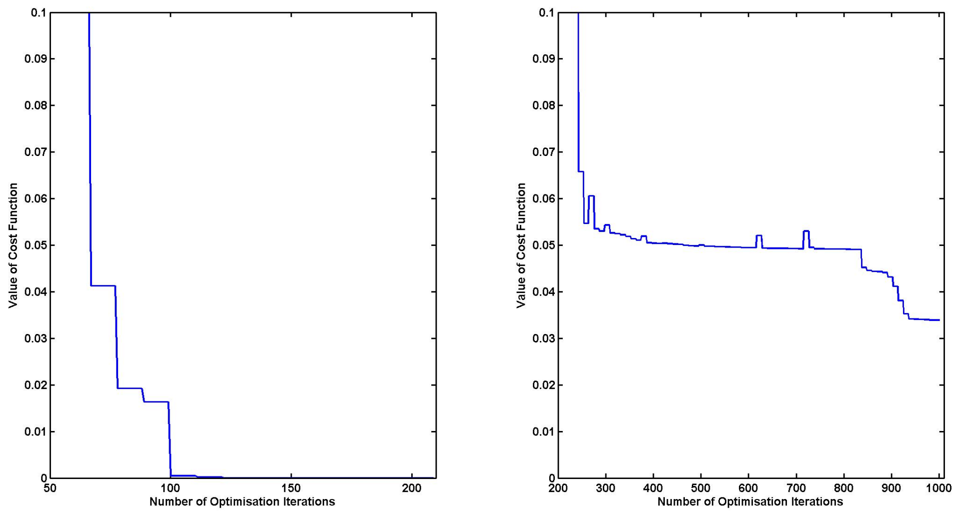

Figure 4.18 Plots showing the convergence of the objective function to a minimum value during the calibration of the SVJJ model to synthetic data via MATLAB lsqnonlin. The plot on the left gives evidence of convergence to a global minimum, whilst that on the right shows convergence only to a local minimum.

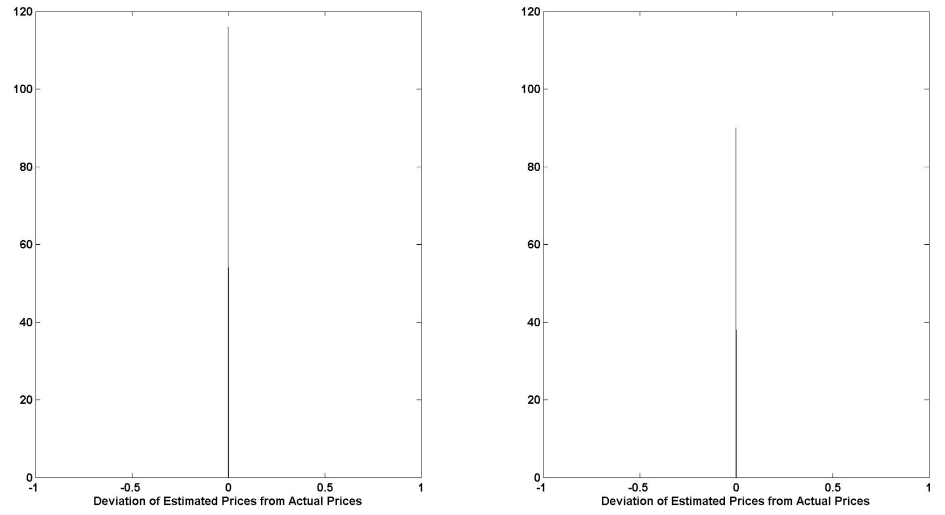

Figure 4.19 Histograms illustrating the deviation of the SVJJ model prices from the original, synthetic data after calibration via MATLAB lsqnonlin. We have fixed κ to 3.5 during the calibration. It is evident that doing so results in an improvement in the fit.

4.3.4 A Summary of Synthetic Data Calibration Results

From what we have seen, the GA calibration routine is the most robust method for all three models. A major advantage of this method is that it does not require initial parameter inputs, eliminating the need to decide on which initial inputs are appropriate and which are not. It is also fairly easy to write code for the algorithm in any programming language. This gives a lot of control to anyone wishing to implement the method, and makes it easy to customise the algorithm to the specific needs of a certain optimisation problem. Thus, although it is not the fastest method to implement, it is our preferred method. As a consequence, we use it in the next section to calibrate the three models to real world data.

The other two calibration methods underperformed relative to the GA method. Notably, the MATLAB lsqnonlin method proved to be quick to implement. It was, however, sensitive to its initial input values and did, in some instances, converge to a point other than the global minimum. Its combination, on the other hand, with the GA method proved to be very successful. This hints at a use for such local optimisation routines — they are very useful for honing the results of more complex global optimisation routines.

The ASA calibration scheme gave good results for the Heston model, but faired slightly worse when applied to the Bates and SVJJ models. It is also computationally cumbersome. Nonetheless, it is not very sensitive to initial parameter inputs and so it faired better than the lsqnonlin method. If more time were spent customising the algorithm to the specific problems at hand, it might give better results. Given the ease and simplicity of implementing the GA scheme, however, we still favour it over the ASA scheme.

4.4 Calibration Results Using Market Data

In this section, we calibrate our models to ALSI futures options data, as well as S&P 500 options data. For both sets of data, we use the genetic algorithm calibration method discussed above to obtain fits for our three models. Data for ALSI futures options can be obtained from the South African Futures Exchange (SAFEX) website [54] and data for S&P 500 options can be bought from Market Data Express (http://www.marketdataexpress.com).

4.4.1 Calibration to ALSI Options Data

South African ALSI options are based on the JSE Top 40 Index, referred to as the TOPI. Information for these contracts can be obtained from the SAFEX website (see SAFEX [54] and the JSE [55]). They are American style futures options and expire on the third Thursday of every expiration month. The options traded on the exchange are also margined, meaning that the option purchaser does not pay outright for the option at inception. Rather, he pays an initial margin at inception and then updates this based on the daily change in the mark-to-market value of the option. The call pricing formula used by SAFEX for mark-to-market purposes is as follows:

$$V^{\text{Call}} = FN\left(\frac{\log\left(\frac{F}{K}\right)+\frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}\right) - KN\left(\frac{\log\left(\frac{F}{K}\right)-\frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}\right),$$

where F is the value of the underlying futures contract and K is the strike price of the option. A similar formula holds for put options. This formula is similar to the Black pricing formula. Here, no risk-free rate is accounted for due to the daily margining requirements of the exchange. Moreover, it can be shown that the early exercise of calls and puts is not optimal (in spite of their structure), and so we can treat them as European options. This is convenient for our purposes, as we have only considered pricing formulas for European style contracts. See West [58] for a thorough treatment of this topic.

The exchange publishes daily implied volatility data as well as the mark-to-market prices of the ALSI options. We choose to calibrate to 76 options on the 11 May 2011, with strike prices ranging from 24000 to 36000 index points on an underlying of 28933 points. The maturity dates on the options range from 1 month to 10 months in quarterly intervals. Consequently, the market is fairly illiquid. We calibrate our models to this data using the GA routine with a population size of 1000 over 100 generations. Since we are dealing with futures options, we need not worry about dividend rates. The table below shows the results of our calibration.

Model Calibration to ALSI Futures Options Data

	Heston Parameters	Bates Parameters	SVJJ Parameters
κ	0.0710	0.0924	0.2467
θ	0.8089	0.4520	0.1925
σ_v	0.4273	0.3172	0.3293
ρ	-0.7504	-0.7825	-0.7682
V_0	0.0387	0.0367	0.0366
λ	-	0.0081	0.0082
μ_S	-	-4.9953	-4.6717
σ_S	-	0.0135	0.0000
ρ_J	-	-	-1.0000
μ_V	-	-	2.1746
Cost Function Value	1.6657×10^3	1.4541×10^3	1.0985×10^5
Calibration Time	2095 seconds	2972 seconds	4302 seconds

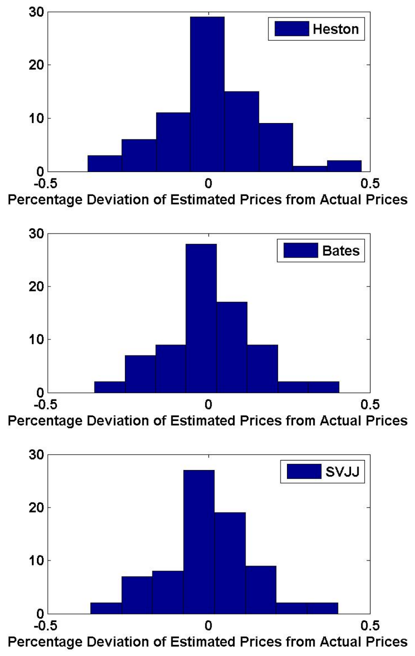

Figure 4.20 Histograms showing the fits of the three models to the ALSI options data. The plots depict the deviation of model prices using calibrated parameters from market prices, as a percentage of strike. We see that the fits of all three models are quite reasonable.

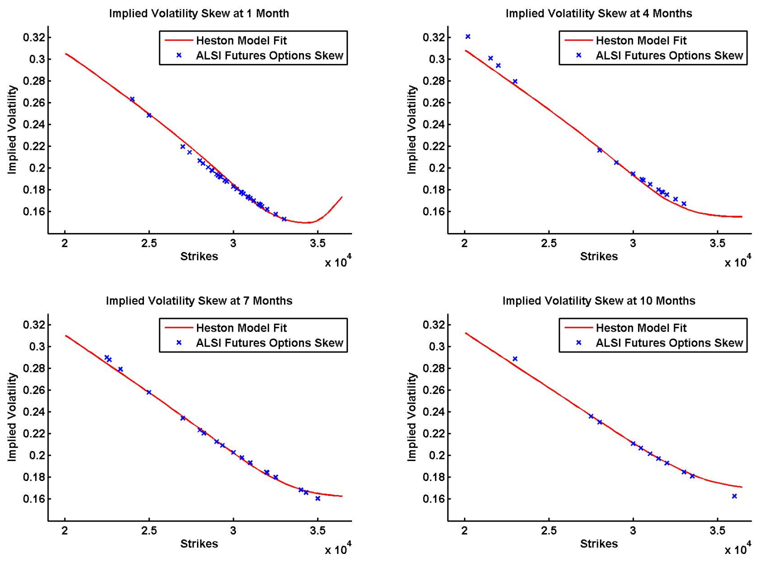

Figure 4.21 Plots showing the fit of the Heston model to ALSI implied volatility skews. The model provides a good fit for implied volatilities which are close to ATM. The fit for ITM and OTM options is not quite as good. The model also does not generate much skewness, particularly in the short term, and so probably provides the best fit to the data (given the shapes of the market skews).

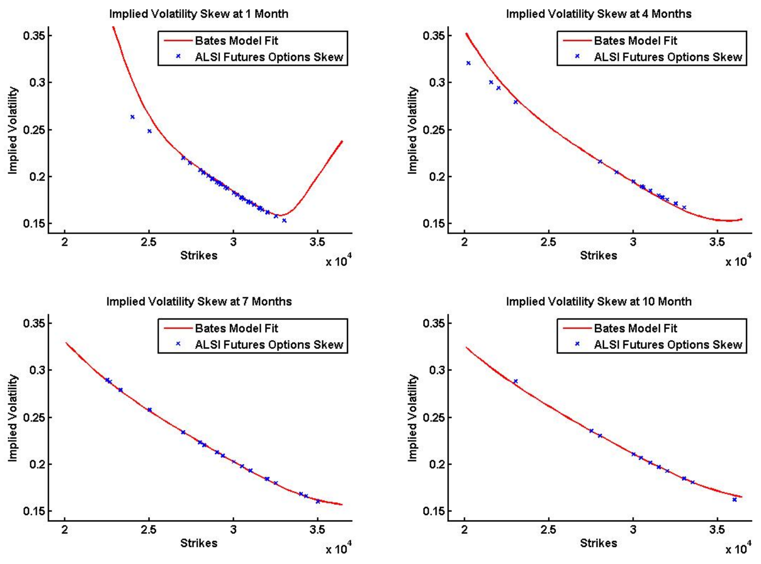

Figure 4.22 Plots showing the fit of the Bates model to ALSI implied volatility skews. In the long-term, the Bates model is able to capture the shape of the implied skew quite well. In the short term, the model generates too much skewness and steepness resulting in a poorer fit than that for the Heston model.

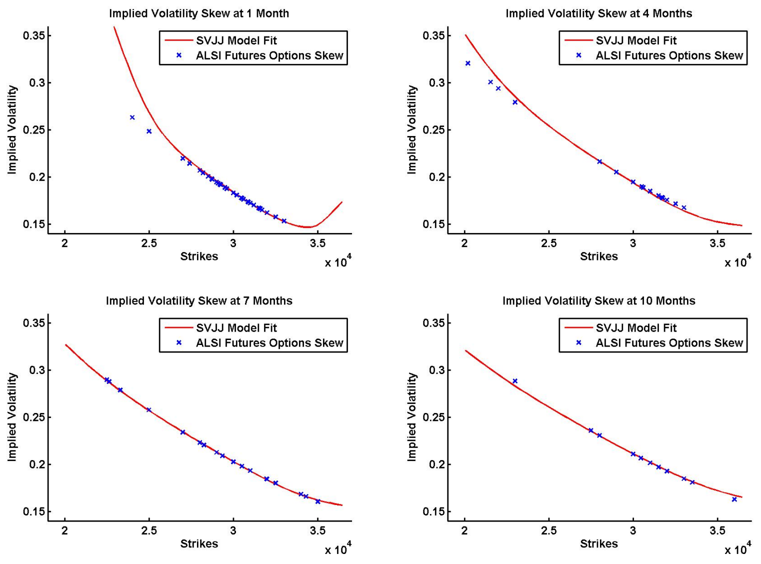

Figure 4.23 Plots showing the fit of the SVJJ model to ALSI implied volatility skews. The results for the SVJJ model are similar to those for the Bates model. It provides a good fit to the skew in the long term, but not such a good fit to that in the short term.

4.4.2 Calibration to S&P 500 Options Data

The S&P 500 index is an index of 500 stocks traded on the NYSE, the AMEX and the Nasdaq in the United States of America. The stocks selected for the index are not necessarily those from the 500 largest companies on these exchanges, but rather, they are selected from 500 companies which are deemed to be the most important companies in the most influential economic sectors in the USA. The index has come to be one of the most analysed and referenced indices in the world. Options traded on the S&P 500 are very liquid and, as a result, are often used by financial engineers for calibration and model testing purposes.

Information on S&P 500 options can be obtained from the CBOE website [17]. The options on the index are European in style and expire on the Saturday after the third Friday of a given expiration month. There can be as many as twelve "near-term" months in which the options expire and it is possible to find options going out to five years. Strike prices on the options can be as near as five index points apart for short dated options. Longer dated options tend to have fewer and further spaced strikes. The options settle for cash at their expiry.

Below, we calibrate the Heston, Bates and SVJJ models to last-trade option data on the index from the 11 May 2011. We use the genetic algorithm approach discussed above to do so and allow the algorithm to run for 100 generations, with 1000 individuals in each generation. The option data that we use comprise 148 options on an underlying of 1342.08 points. The strike prices considered range from 1200 to 1600 and the maturity dates extend from 10 days to 2.6 years. We also set the risk-free rate to a constant value of 0.5% and the dividend yield to 1.8%. The data are obtained from the Market Data Express website.

Model Calibration to S&P 500 Options Data

	Heston Parameters	Bates Parameters	SVJJ Parameters
κ	1.3887	6.4866	1.8370
θ	0.0780	0.0587	0.0263
σ_v	0.6126	2.0592	0.1594
ρ	-0.8208	-0.7342	-1.0000
V_0	0.0273	0.0236	0.0202
λ	-	0.0059	0.0488
μ_S	-	-1.8801	-1.1868
σ_S	-	0.8393	0.0010
ρ_J	-	-	0.0196
μ_V	-	-	0.0000
Cost Function Value	1.1987×10^6	3.987×10^3	8.7379×10^3
Calibration Time	5115 seconds	6768 seconds	7554 seconds

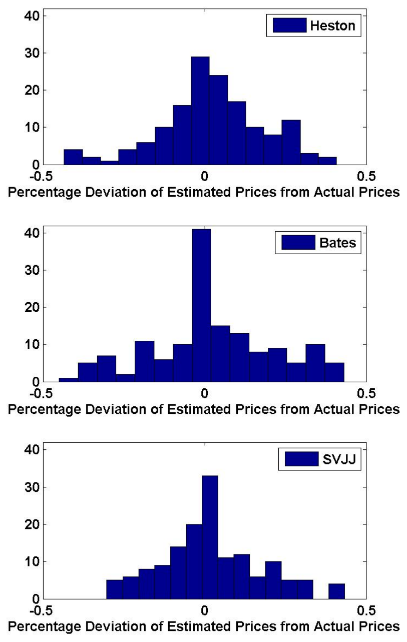

Figure 4.24 Histograms showing the fits of the three models to the S&P 500 options data. The plots depict the deviation of model prices using calibrated parameters from market prices, as a percentage of strike. We see that the fits of all three models are quite reasonable.

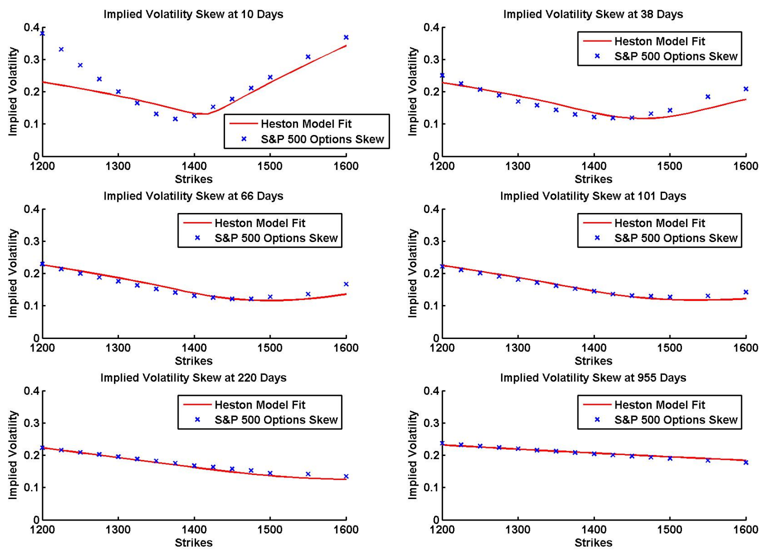

Figure 4.25 Plots showing the fit of the Heston model to S&P 500 implied volatility skews. We can see here that the Heston model provides a good fit to the long-dated skews. It cannot quite capture the skewness in the short-term however.

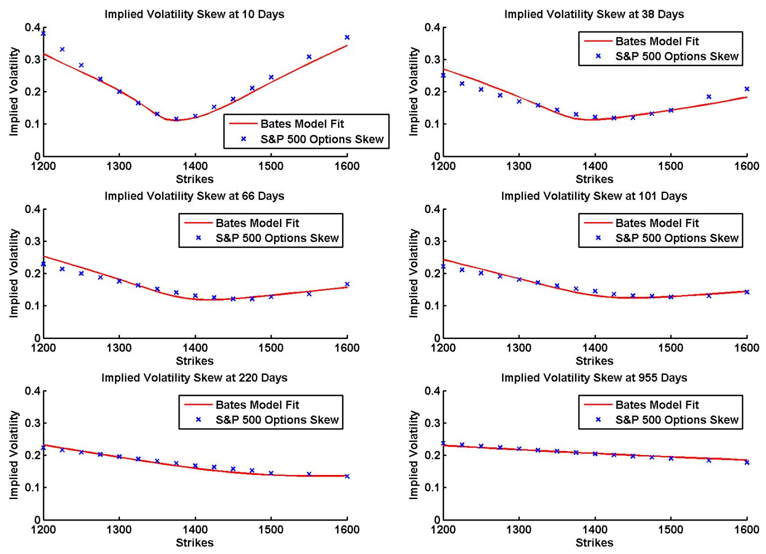

Figure 4.26 Plots showing the fit of the Bates model to S&P 500 implied volatility skews. The Bates model provides a good fit to both the short and long-dated skews. Unlike the Heston model, it is able to generate sufficient skewness in the short term and, out of the three models, probably provides the best fit to the skews.

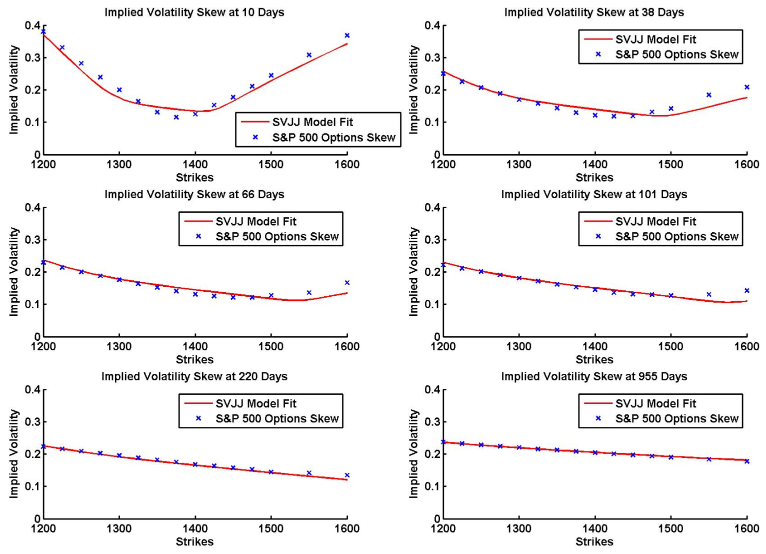

Figure 4.27 Plots showing the fit of the SVJJ model to S&P 500 implied volatility skews. The SVJJ model yields a good fit to both the short and long-dated skews. The model is more than capable of generating sufficient short term skewness.

4.4.3 A Summary of Market Data Calibration Results

The calibration results[3] for the ALSI options data suggest that the Heston and Bates models provide the best fits. The shape of the ALSI implied volatility skew is steep, but straight, at all maturities. Since the Heston model generates the least skewness and kurtosis out of the three models, it would seem to be the best model to fit to the option data. The implied volatility plots for the Bates and SVJJ models both indicate that the models generate too much skewness in the short term to provide a good fit to the ALSI skew. We thus favour the Heston model for calibration to ALSI option data (out of the three models). It must be stated, however, that the ALSI market is relatively illiquid which makes it difficult to calibrate models to its implied volatility surface.

By way of reference, two sources that have dealt with similar subject matter to us are those by West [58] and Kotzé and Joseph [37]. These two works are both concerned with fitting models to option data from options on the South African Top 40 index. Specifically, the paper by West deals with the calibration of the SABR model to data on this index, whilst that by Kotzé and Joseph presents the use of a quadratic deterministic model to fit the Top 40 implied volatility surface. This dissertation is in a similar vein to these two, as we have also presented methods for calibrating models to data on the South African market. More specifically, we attempt to fit stochastic volatility models to the whole implied volatility surface, which is an issue that is not explored in these works. We recommend that the reader refer to these sources as they provide a thorough investigation into this topic.

The calibration results for the S&P 500 index options are quite different to those for the ALSI options. We see that the volatility smile for the S&P 500 index is quite pronounced in the short term and flattens out in the long term. The Heston model is unable to satisfy these characteristics. It cannot generate sufficient skewness and kurtosis in the short term, while still providing a relatively flat skew in the long term. Instead, we require jumps in the stock price process and possibly in the volatility process too, in order to fit volatility surfaces with such characteristics. Out of the two jump models, the Bates model gives a slightly better fit to the S&P 500 index than the SVJJ model does. This agrees with the results of Gatheral [25], who performs a similar calibration and finds that a Bates-style model fits S&P 500 data better than the other two models do. He finds that the increased

[3]We ensure that the search range for each parameter in the GA calibration routine is sufficiently large. For example, for the SVJJ model, we search for κ in the interval $[0, 10]$, θ in $[0, 1]$, σ_v in $[0, 5]$, ρ in $[-1, 1]$, V_0 in $[0, 1]$, λ in $[0, 5]$, μ_S in $[-5.5]$, σ_S in $[0, 0.001]$ (since the parameter is quite sensitive and usually observed to be quite small in the literature), ρ_J in $[-1, 1]$ and μ_V in $[0, 5]$. Identical ranges were used for the other two models (excluding those for the parameters which are not relevant in these models).

number of parameters of the SVJJ model relative to a model with jumps in the stock price process only makes it harder to calibrate to market data and states this as the main reason for the poorer fit of the SVJJ model.

The results of the S&P 500 index fit are particularly relevant because they highlight the role that jump models with stochastic volatility can play in modeling implied volatility surfaces. Markets with such dynamics clearly require the use of models that can generate sufficient skewness and kurtosis to fit the short end of the implied volatility surface, while still being able to fit the long end of the surface. Models such as the Heston model are unable to do this, whereas including jumps in the stock price and possibly the volatility processes provides a solution to the dilemma. This conclusion is reached by numerous other authors (Bakshi et al. [3], Bates [5], Broadie et al. [10], Duffie et al. [22], Gatheral [25]) and justifies the use of higher dimensional models such as the Bates and SVJJ models.

4.4.4 A Comment on Calibration Speed Improvements with Parallel Computing Methods for the Genetic Algorithm

A slight divergence from the rest of the section, but nonetheless, a relevant consideration for the calibration of models to market data is the parallelisation of optimisation routines. The FFT pricing routine already provides a fast pricing scheme through which option price models can be calibrated to market data. This occurs as a result of its ability to compute option prices with a given maturity for a large range of strikes simultaneously. Other methods, which can only compute one option price at a time slow down the calibration routines dramatically. Nonetheless, the implementation of these schemes can be sped up further through the use of parallel computing methods. We have two options to do so. We can either implement the routines on multiple CPU cores or we can implement them on a GPU.

The implementation of the GA in parallel on the multiple cores of a CPU can be accomplished in MATLAB through the use of the *parfor* command. The only part of the GA that is computationally intensive is the computation of the fitness values of the population individuals in each generation of the algorithm. Inserting a *parfor* loop around this part of the algorithm could provide a speed-up to the calibration routine. Unfortunately, in our implementation of this method, we only achieved about a 2 times speed-up (on the six cores of the Intel i7 CPU). The reason for the rather poor improvement was a result of an increased computation time for each evaluation of the FFT routine in parallel compared to that in serial. This is most likely due to excessive communication overhead in passing variables to and from the cores of the CPU. A further research topic of interest would be to explore the successful implementation of this algorithm in parallel to get significant reductions in calibration times. Such research would improve the feasibility of implementing the GA calibration routine in practice.

The implementation of the FFT pricing routine on a GPU could also yield further speed improvements for the calibration routines that we have considered so far. Since GPU's are well suited to the computation of FFT's (due to their gaming origins), they would be very useful for such applications. Indeed, some of the worlds fastest computers today are now making use of GPU's to speed up many simulations that were previously computed on multiple CPU's. In our attempt to implement the GA calibration routine on a GPU, we were met with many restrictions due to the still underdeveloped ability of MATLAB to interface with a GPU. MATLAB is unable to implement routines such as the *parfor* routine on a GPU, making it difficult to simulate the GA calibration routine on the GPU. Instead, we simply attempted to evaluate each call of the FFT on the GPU. This, however, resulted in the need to transfer large matrices to and from the GPU, increasing the communication overhead and slowing down the routine. In the end, we saw no speed improvements as a result of implementing the calibration routine on the GPU. Further research in this area would be very useful and would yield ways of achieving large speed improvements for optimisation routine such as the GA.

Chapter 5

Hedging

The final theme of this dissertation is hedging. The calibration of models to market data, as well as option pricing methods for these models provide us with only half the tools for the successful trading of options. The methods that we employ to hedge the options once they have been traded are as important as using appropriate and properly calibrated models.

Traditionally, the Black-Scholes-Merton framework (Black and Scholes [7], Merton [42]) gave practitioners a simple and effective method (at least in theory) to fully hedge their option positions. Vanilla European calls and puts could simply be replicated by buying and selling stock, according to the delta of the option, to ensure that the payoff of the option and that of the hedging portfolio matched at maturity. The simplicity of hedging in the Black-Scholes-Merton world results since the only source of uncertainty in the model is due to the driving Brownian motion in the stock price process. This risk can be offset by trading in the underlying asset and as a result, every contingent claim can be hedged. According to the second Fundamental Theorem of Asset Pricing, this implies that the market is complete and the equivalent martingale measure in the model is unique (see Shreve [53]). Unfortunately, when dealing with stochastic volatility models, we do not have the luxury of working in complete markets and the equivalent martingale measure in the model is no longer unique. This complicates the hedging schemes that must be used to hedge options priced under these models. One approach to constructing hedging portfolios that provide an improved hedge is to consider traded vanilla options which are considered to be as liquid as the underlying. Hedging portfolios can then include vanilla options to offset volatility risk. Adding jumps to stochastic volatility models makes hedging even trickier and if jump magnitudes are unbounded, the jump risk in the model cannot be hedged away completely.

There are many articles and books that tackle the hedging problem in incomplete markets. Some of the more popular topics of investigation in these works include mean-variance hedging, mean-self-financing hedging and superhedging. Bingham and Kiesel [6] give a thorough overview of mean-variance hedging. This method essentially entails finding a self-financing strategy that, with minimal variance, gives an approximation to the payoff of a financial instrument (such as an option). Fouque et al. [24] illustrate a mean-self-financing option hedging strategy, involving only bonds and the underlying stock that, on average, does not require the input of additional funds, nor does it result in the outflow of funds. Cont and Tankov [20] give an overview of superhedging schemes, which entail finding self-financing strategies that always produce a greater payoff than that of the instrument that they are hedging.

In our brief investigation of hedging strategies for the Heston, Bates and SVJJ models, we draw from the paper by Kurpiel and Roncalli [38]. They compare three option hedging strategies — a delta hedging scheme, a delta-sigma hedging scheme and a delta-sigma-gamma hedging scheme — which use option price sensitivities to the underlying parameters (i.e. the stock price and volatility) to construct hedging portfolios. In doing so, they make use of vanilla options to construct hedges for the option under consideration. For our implementation of their methods, we consider a vanilla call option as the primary option. The parameters of this option are given in the simulation parts of this chapter. It might seem somewhat counterintuitive to use vanilla options to hedge a position in a vanilla option; however, we proceed in this way in order to illustrate the effectiveness of this method. In further research, it would also be possible to extend this analysis to the case of exotic options, which would be more realistic than the current setting. We will see that these methods are effective for the Heston model, but less so for the other two models due to the presence of jumps.

5.1 A Change of Measure in the Heston Model

In the preceding sections of this dissertation, we have concerned ourselves only with models under an appropriate risk-neutral measure. This was acceptable since we used the risk-neutral formulations of the models for the pricing of options as well as the calibration of the models to option data, both of which take place under the risk neutral measure (since calibration to option data essentially allows us to estimate the risk-neutral parameters of the models and so no transformation to the risk-neutral world is required). For the purpose of hedging however, it is useful to consider risk-neutral transformations.

The formulation of the Heston model can be expressed as follows under the real-world measure:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{V_t} S_t dZ_t^{(1)} \\ dV_t &= \kappa \left(\theta - V_t\right) dt + \sigma_v \sqrt{V_t} \left[\rho dZ_t^{(1)} + \sqrt{1-\rho^2} dZ_t^{(2)}\right], \end{aligned}$$

where $Z_t^{(1)}$ and $Z_t^{(2)}$ are independent Brownian motions. A change of measure in this model is as a result of Girsanov's Theorem (see Shreve [53]).

Theorem 4 (Girsanov in N Dimensions)**.** *Define a probability space $(\Omega, \mathcal{F}_t, \mathcal{F}, \mathbb{P})$, where $T > 0$ is a fixed time and $t \in [0,T]$, $\mathbb{P}$ is the real-world (objective) measure on that space and $\mathcal{F} = \mathcal{F}_T$. Upon this probability space, define an N-dimensional Brownian motion $Z_t = \left(Z_t^{(1)}, Z_t^{(2)}, \ldots, Z_t^{(N)}\right)$ where N is a positive integer. Let $\Theta_t = \left(\Theta_t^{(1)}, \Theta_t^{(2)}, \ldots, \Theta_t^{(N)}\right)$ be a N-dimensional measurable and adapted process. Define*

$$\zeta_t = \exp\left\{-\int_0^t \Theta_u \cdot dZ_u - \frac{1}{2}\int_0^t \sum_{i=1}^{N} \left(\Theta_u^{(i)}\right)^2 du\right\} \tag{5.1}$$

and

$$d\widetilde{Z}_t = dZ_t + \Theta_t dt, \tag{5.2}$$

assuming that Θ_t satisfies Novikov's condition:

$$\mathbb{E}\left[\exp\left(\frac{1}{2}\int_0^T \sum_{i=1}^{N} \left(\Theta_u^{(i)}\right)^2 du\right)\right] < \infty.$$

For convenience, we set $\zeta = \zeta_T$ so that $\mathbb{E}[\zeta] = 1$. Now, under the risk-neutral probability measure $\widetilde{\mathbb{P}}$, where

$$\widetilde{\mathbb{P}}(A) = \int_A \zeta(\omega)\, d\mathbb{P}(\omega), \ \text{for all } A \in \mathcal{F}, \tag{5.3}$$

the process $\widetilde{Z}_t = \left(\widetilde{Z}_t^{(1)}, \widetilde{Z}_t^{(2)}, \ldots, \widetilde{Z}_t^{(N)}\right)$ is a N-dimensional Brownian motion.

We refer to the works of Chernov and Ghysels [16] as well as Fouque et al. [24] for our treatment on the risk-neutral transformation in the Heston model. The aim of the risk-neutral transformation is to find a measure, equivalent to $\mathbb{P}$, under which the discounted stock price process is a martingale. In the case of the Heston model, such a measure is not unique, so we denote a chosen equivalent martingale measure by $\widetilde{\mathbb{P}}^{(\varphi)}$. Through the application of Girsanov's Theorem, we can construct the $\widetilde{\mathbb{P}}^{(\varphi)}$-Brownian motions, $\widetilde{Z}_t^{(1)}$ and $\widetilde{Z}_t^{(2)}$. The Radon-Nikodým derivative process can then be defined by:

$$\zeta_t = \exp\left\{-\frac{1}{2}\int_0^t \left[\left(\Theta_u^{(1)}\right)^2 + \left(\Theta_u^{(2)}\right)^2\right] du - \int_0^t \Theta_u^{(1)} dZ_u^{(1)} - \int_0^t \Theta_u^{(2)} dZ_u^{(2)}\right\}, \tag{5.4}$$

such that,

$$\begin{aligned} d\widetilde{Z}_t^{(1)} &= dZ_t^{(1)} + \Theta_t^{(1)} dt \qquad (5.5) \\ d\widetilde{Z}_t^{(2)} &= dZ_t^{(2)} + \Theta_t^{(2)} dt, \qquad (5.6) \end{aligned}$$

where

$$\Theta_t^{(1)} = \frac{\mu - r}{\sqrt{V_t}} \qquad (5.7)$$

is the market price of return risk and

$$\Theta_t^{(2)} = \varphi_t \qquad (5.8)$$

is the market price of volatility risk. Under $\widetilde{\mathbb{P}}^{(\varphi)}$, our two processes become

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{V_t} S_t \left[d\widetilde{Z}_t^{(1)} - \Theta_t^{(1)} dt \right] \\ &= S_t \left[\mu - \sqrt{V_t} \Theta_t^{(1)} \right] dt + \sqrt{V_t} S_t d\widetilde{Z}_t^{(1)} \\ &= r S_t dt + \sqrt{V_t} S_t d\widetilde{Z}_t^{(1)} \qquad (5.9) \end{aligned}$$

and

$$\begin{aligned} dV_t &= \kappa (\theta - V_t) dt + \sigma_v \sqrt{V_t} \left\{ \rho \left[d\widetilde{Z}_t^{(1)} - \Theta_t^{(1)} dt \right] + \sqrt{1-\rho^2} \left[d\widetilde{Z}_t^{(2)} - \Theta_t^{(2)} dt \right] \right\} \\ &= \left[\kappa (\theta - V_t) - \sigma_v \sqrt{V_t} \left(\rho \Theta_t^{(1)} + \sqrt{1-\rho^2} \Theta_t^{(2)} \right) \right] dt \\ &\quad + \sigma_v \sqrt{V_t} \left[\rho d\widetilde{Z}_t^{(1)} + \sqrt{1-\rho^2} d\widetilde{Z}_t^{(2)} \right] \\ &= \kappa \left(\theta^{(\varphi)} - V_t \right) dt + \sigma_v \sqrt{V_t} \left[\rho d\widetilde{Z}_t^{(1)} + \sqrt{1-\rho^2} d\widetilde{Z}_t^{(2)} \right], \qquad (5.10) \end{aligned}$$

where

$$\theta^{(\varphi)} = \theta - \frac{\sigma_v \rho (\mu - r)}{\kappa} - \frac{\sigma_v \sqrt{V_t} \sqrt{1-\rho^2} \varphi_t}{\kappa}. \qquad (5.11)$$

The market price of volatility risk stems from the driving Brownian motion in the volatility process. Since the equivalent martingale measure in the model depends on the market price of volatility risk, different calibrations of φ_t will be coupled to various risk-neutral transformations under the model. As a result, φ_t can be seen to parameterise the space of risk-neutral measures (Fouque et al. [24]).

The results above imply that a contingent claim in the Heston model market cannot be replicated by trading bonds and the underlying stock alone. This can be illustrated by considering the Martingale Representation Theorem (see Shreve [53]).

Theorem 5 (The Martingale Representation Theorem in N Dimensions)**.** *As for Girsanov's Theorem, consider a probability space $(\Omega, \mathcal{F}_t, \mathcal{F}, \mathbb{P})$ with a N-dimensional Brownian motion, Z_t defined on that space ($t \in [0,T]$). We suppose that the filtration $\mathcal{F}_t$ is generated by the Brownian motion Z_t. Next, defining on the probability space a $\mathbb{P}$-martingale, $\mathcal{M}_t$, with respect to this filtration, we can state that there exists a N-dimensional, adapted and square-integrable process $\vartheta_t = \left(\vartheta_t^{(1)}, \vartheta_t^{(2)}, \ldots, \vartheta_t^{(N)}\right)$ where*

$$\mathcal{M}_t \;=\; \mathcal{M}_0 + \int_0^t \vartheta_u \cdot dZ_u. \tag{5.12}$$

Considering the case under the Heston model, $\mathcal{F}_t$ is generated by the Brownian motions driving the stock price and volatility processes — $\widetilde{Z}_t^{(1)}$ and $\widetilde{Z}_t^{(2)}$. Moreover, the discounted risk-neutral pricing function $Y_t = e^{-rt}H_t$, where

$$H_t \;=\; \widetilde{\mathbb{E}}_t^{(\varphi)}\left[e^{-r(T-t)}H_T\right],$$

is a martingale with respect to $\widetilde{\mathbb{P}}^{(\varphi)}$, given the filtration up to time t (H_T is the time T payoff of a contingent claim). By the martingale representation theorem, we can state that

$$Y_t \;=\; Y_0 + \int_0^t \vartheta_u^{(1)} dZ_u^{(1)} + \int_0^t \vartheta_u^{(2)} dZ_u^{(2)} \tag{5.13}$$

for adapted and square-integrable processes $\vartheta_t^{(1)}$ and $\vartheta_t^{(2)}$. Now, we can use this to write a self-financing trading strategy for the contingent claim, H_T, of the form (see Fouque et al. [24]):

$$dH_t \;=\; \alpha_t dS_t + \beta_t r e^{rt} dt + \gamma_t dV_t$$

for some α_t, β_t, γ_t. Notably, the presence of $Z_t^{(2)}$ in the martingale representation of Y_t forces us to construct a self-financing strategy in terms of the volatility of the stock. This means that we cannot use bonds and the underlying stock alone to hedge the derivative, which is undesirable since volatility is not a traded asset. As a result, if we want to include only the stock and bonds in our hedging portfolio, we need to construct replicating portfolios that minimise our hedging losses (or that are mean-self-financing). We can also include traded options in our replicating portfolio so as to hedge the volatility risk (thus including vanilla options in the list of market traded assets). Such a strategy can be undesirable when we are faced with high transaction costs and illiquid options markets.

5.2 Hedging Strategies for the Heston Model

We turn our attention to the construction of hedging portfolios using bonds, the underlying stock and other options. Much of our inspiration comes from the paper by Kurpiel and Roncalli [38].

5.2.1 Delta Hedging in the Heston Model

Consider a portfolio, Π_t, consisting of a long position in a call option, $C(S_t, V_t, T, K)$, and a short position in $\delta_t^{(S)}$ units of the underlying stock S_t. As usual, $t \in [0, T]$, where T is the maturity of our option. We also assume that the stock price, S_t, follows the dynamics of the Heston model (already defined), with V_t denoting the volatility process. The infinitesimal change in the value of this portfolio is given by (using a simplified notation)

$$\begin{aligned} d\Pi &= dC - \delta^{(S)}dS \\ &= \left[\frac{\partial C}{\partial t} + \frac{1}{2}VS^2\frac{\partial^2 C}{\partial S^2} + \rho\sigma_v VS\frac{\partial^2 C}{\partial V\partial S} + \frac{1}{2}\sigma_v^2 V\frac{\partial^2 C}{\partial V^2}\right]dt \\ &\quad + \left[\frac{\partial C}{\partial S} - \delta^{(S)}\right]dS + \frac{\partial C}{\partial V}dV \end{aligned} \tag{5.14}$$

by Itô's lemma. To make the portfolio delta-neutral, we set

$$\frac{\partial C}{\partial S} - \delta^{(S)} = 0$$

and so

$$\delta^{(S)} = \frac{\partial C}{\partial S}. \tag{5.15}$$

The instantaneous variance of the portfolio value is given by

$$d\langle\Pi, \Pi\rangle_t = \left(\frac{\partial C}{\partial V}\right)^2 \sigma_v^2 V dt, \tag{5.16}$$

where $d\langle X, Y\rangle_t$ denotes the covariation of two stochastic processes X_t and Y_t. Clearly, delta-hedging does not eliminate all the risk in the portfolio — it only eliminates risk in the portfolio resulting from the randomness in the stock price process. Residual risk still remains as a result of the random component in the volatility process. To hedge this risk, we need to consider other options on the same underlying as traded assets and include them in the portfolio. This results in what Kurpiel and Roncalli [38] name the delta-sigma hedging procedure.

5.2.2 Delta-Sigma Hedging in the Heston Model

Again, we consider a portfolio, Π_t, which is comprised of a long position in a call option, $C(S_t, V_t, T, K)$, and a short position in $\delta^{(S)}$ units of the underlying stock S_t. In addition to this, however, we also include a short position in $\delta^{(H_1)}$ units of another option, $H_1(S_t, V_t, T_1, K_1)$ written on the same underlying as our original option. The infinitesimal change in the value of our portfolio is now given by

$$\begin{aligned} d\Pi &= dC - \delta^{(S)}dS - \delta^{(H_1)}dH_1 \\ &= \left[\frac{\partial C}{\partial t} + \frac{1}{2}VS^2\frac{\partial^2 C}{\partial S^2} + \rho\sigma_v VS\frac{\partial^2 C}{\partial V\partial S} + \frac{1}{2}\sigma_v^2 V\frac{\partial^2 C}{\partial V^2}\right]dt \\ &\quad - \delta^{(H_1)}\left[\frac{\partial H_1}{\partial t} + \frac{1}{2}VS^2\frac{\partial^2 H_1}{\partial S^2} + \rho\sigma_v VS\frac{\partial^2 H_1}{\partial V\partial S} + \frac{1}{2}\sigma_v^2 V\frac{\partial^2 H_1}{\partial V^2}\right]dt \\ &\quad + \left[\frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S} - \delta^{(S)}\right]dS + \left[\frac{\partial C}{\partial V} - \delta^{(H_1)}\frac{\partial H_1}{\partial V}\right]dV. \end{aligned} \tag{5.17}$$

To make this portfolio instantaneously risk-free, we set

$$\frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S} - \delta^{(S)} = 0$$

and

$$\frac{\partial C}{\partial V} - \delta^{(H_1)}\frac{\partial H_1}{\partial V} = 0.$$

This gives us the hedging ratios

$$\delta^{(H_1)} = \frac{{}^{\partial C}/_{\partial V}}{{}^{\partial H_1}/_{\partial V}} \tag{5.18}$$

$$\delta^{(S)} = \frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S}. \tag{5.19}$$

Unlike the case with delta-hedging, the instantaneous variance in this portfolio is now 0. The delta-sigma hedging scheme solves the hedging problem in the Heston model — adding a traded option to the hedging portfolio allows for a perfect hedge (if trading occurs continuously and there are no transaction costs). Note that $\delta^{(H_1)}$ is the ratio of the vegas of the two options, and $\delta^{(S)}$ is expressed in terms of the deltas of the two options. Our next step is to discretise this hedging scheme to allow for a more practical approach to the hedging problem. This introduces errors into the hedging scheme and so we include a gamma hedging component to improve the discrete hedge.

5.2.3 Delta-Sigma-Gamma Hedging in the Heston Model

In a similar way to the subsections above, consider a portfolio at time t given by

$$\Pi_t = C_t - \delta_t^{(S)}S_t - \delta_t^{(H_1)}H_{1,t} - \delta_t^{(H_2)}H_{2,t}, \tag{5.20}$$

where H_1 and H_2 are distinct exchange traded options dependent on the same underlying asset. If instead of re-balancing the portfolio continuously, we only re-balance it at equally spaced, discrete points in time, where the distance between successive points is Δt, then the change in the value of the portfolio over an interval $[t, t+\Delta t]$ is given by

$$\Pi_{t+\Delta t} - \Pi_t \;=\; \Delta\Pi_t.$$

Dropping the subscripts and applying the Taylor expansion about S, V and t we get

$$\begin{aligned}
\Delta\Pi_t \;=&\; \Delta C - \delta^{(S)}\Delta S - \delta^{(H_1)}\Delta H_1 - \delta^{(H_2)}\Delta H_2 \\
=&\; \left[\frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S} - \delta^{(H_2)}\frac{\partial H_2}{\partial S} - \delta^{(S)}\right]\Delta S + \left[\frac{\partial C}{\partial V} - \delta^{(H_1)}\frac{\partial H_1}{\partial V} - \delta^{(H_2)}\frac{\partial H_2}{\partial V}\right]\Delta V \\
&+ \left[\frac{\partial C}{\partial t} - \delta^{(H_1)}\frac{\partial H_1}{\partial t} - \delta^{(H_2)}\frac{\partial H_2}{\partial t}\right]\Delta t + \frac{1}{2}\left[\frac{\partial^2 C}{\partial S^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial S^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial S^2}\right](\Delta S)^2 \\
&+ \frac{1}{2}\left[\frac{\partial^2 C}{\partial V^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial V^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial V^2}\right](\Delta V)^2 \\
+&\; \frac{1}{2}\left[\frac{\partial^2 C}{\partial t^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial t^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial t^2}\right](\Delta t)^2 \\
&+ \ldots \text{ (Cross-terms and higher order terms).}
\end{aligned} \tag{5.21}$$

Now, to make the portfolio delta, gamma and vega neutral we set

$$\begin{aligned}
\left[\frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S} - \delta^{(H_2)}\frac{\partial H_2}{\partial S} - \delta^{(S)}\right] &= 0 \\
\left[\frac{\partial^2 C}{\partial S^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial S^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial S^2}\right] &= 0 \\
\left[\frac{\partial C}{\partial V} - \delta^{(H_1)}\frac{\partial H_1}{\partial V} - \delta^{(H_2)}\frac{\partial H_2}{\partial V}\right] &= 0
\end{aligned}$$

respectively. Solving yields the following hedging ratios:

$$\delta^{(H_1)} \;=\; \frac{\frac{\partial H_2}{\partial V}\frac{\partial^2 C}{\partial S^2} - \frac{\partial C}{\partial V}\frac{\partial^2 H_2}{\partial S^2}}{\frac{\partial H_2}{\partial V}\frac{\partial^2 H_1}{\partial S^2} - \frac{\partial H_1}{\partial V}\frac{\partial^2 H_2}{\partial S^2}} \;=\; \frac{\mathcal{V}^{(H_2)}\Gamma^{(C)} - \mathcal{V}^{(C)}\Gamma^{(H_2)}}{\mathcal{V}^{(H_2)}\Gamma^{(H_1)} - \mathcal{V}^{(H_1)}\Gamma^{(H_2)}} \tag{5.22}$$

$$\delta^{(H_2)} \;=\; \frac{\frac{\partial^2 C}{\partial S^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial S^2}}{\frac{\partial^2 H_2}{\partial S^2}} \;=\; \frac{\Gamma^{(C)} - \delta^{(H_1)}\Gamma^{(H_1)}}{\Gamma^{(H_2)}} \tag{5.23}$$

$$\delta^{(S)} \;=\; \frac{\partial C}{\partial S} - \delta^{(H_1)}\frac{\partial H_1}{\partial S} - \delta^{(H_2)}\frac{\partial H_2}{\partial S} \;=\; \mathit{\Delta}^{(C)} - \delta^{(H_1)}\mathit{\Delta}^{(H_1)} - \delta^{(H_2)}\mathit{\Delta}^{(H_2)}, \tag{5.24}$$

where $\mathit{\Delta}$, Γ and $\mathcal{V}$ denote the delta, gamma and vega of the respective options. Note, particularly the notational difference between Δ and $\mathit{\Delta}$. The former is used to describe the change in the variable it prefixes. The latter is used to describe the delta of an option.

The variance of $\Delta\Pi_t$ can be shown, approximately, to be (see Kurpiel and Roncalli [38])

$$\begin{aligned}
\mathbb{V}\left[\Delta\Pi_t \,|\mathcal{F}_t\right] &\approx \frac{1}{4}\left[\frac{\partial^2 C}{\partial V^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial V^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial V^2}\right]^2 \mathbb{V}\left[(\Delta V)^2 \,|\mathcal{F}_t\right] \\
&\approx \frac{1}{4}\left[\frac{\partial^2 C}{\partial V^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial V^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial V^2}\right]^2 \left(\sigma_v\sqrt{V}\right)^4 \mathbb{V}\left[(\Delta t)Y^2 \,|\mathcal{F}_t\right] \\
&\quad \text{(where } Y \sim N(0,1)\text{)} \\
&= \frac{1}{2}\left[\frac{\partial^2 C}{\partial V^2} - \delta^{(H_1)}\frac{\partial^2 H_1}{\partial V^2} - \delta^{(H_2)}\frac{\partial^2 H_2}{\partial V^2}\right]^2 \left(\sigma_v\sqrt{V}\right)^4 (\Delta t)^2 . \qquad (5.25)
\end{aligned}$$

This value should be close to 0 for near-to at-the-money options, since the vega of an option is only sensitive to the change in the volatility for deep in-the-money and out-of-the-money options.

5.2.4 Simulations of Hedging Methods in the Heston Model

We now turn our attention to the simulation of the delta, delta-sigma and the delta-sigma-gamma hedging methods. In simulating the hedging schemes, we use Euler Monte Carlo methods to produce the stock paths and the FFT pricing scheme to produce option prices. Our original option, which we are attempting to hedge effectively, is a vanilla at-the-money call with three months to expiry. The other two options used in the delta-sigma and delta-sigma-gamma schemes are also at-the-money vanilla calls with six months and one year to maturity respectively.

The parameters that we select to generate our stock price sample paths are:

$$\begin{aligned}
\mu &= 0.08 \\
r &= 0.05 \\
\kappa &= 1 \\
\theta &= 0.0382 \\
\sigma_v &= 0.2 \\
\rho &= -0.3 \\
V_0 &= 0.04.
\end{aligned}$$

We also set φ_t, our market price of volatility risk, to 0. This essentially means that no volatility risk premium is demanded by investors. Such a choice of φ_t is justified in Kurpiel and Roncalli [38]. Consequently, we have that $\theta^{(\varphi)} = 0.04$ under $\mathbb{P}^{(\varphi)}$.

The simulation of these three hedging strategies requires us to calculate certain option price sensitivities — or "Greeks" — numerically. Specifically, we need to calculate option price deltas, vegas and gammas, where the option delta and vega are the sensitivity of the option price to stock price and volatility changes respectively. The gamma for an option is the second order sensitivity of the option price to stock price changes — or the sensitivity of the option delta to stock price changes. We simulate values for these at each time point by making use of finite difference coefficients. Thus, for an arbitrary vanilla call (denoted by $C(S_t, V_t, T, K)$), we calculate values for the "Greeks" $\mathit{\Delta}$, $\mathcal{V}$ and Γ at time t as follows:

$$\begin{aligned}\mathit{\Delta} &= \frac{C(S_t+\Delta S_t, V_t, T, K) - C(S_t, V_t, T, K)}{\Delta S_t}\\ \mathcal{V} &= \frac{C(S_t, V_t+\Delta V_t, T, K) - C(S_t, V_t, T, K)}{\Delta V_t}\\ \Gamma &= \frac{C(S_t+\Delta S_t, V_t, T, K) - 2C(S_t, V_t, T, K) + C(S_t-\Delta S_t, V_t, T, K)}{(\Delta S_t)^2}.\end{aligned}$$

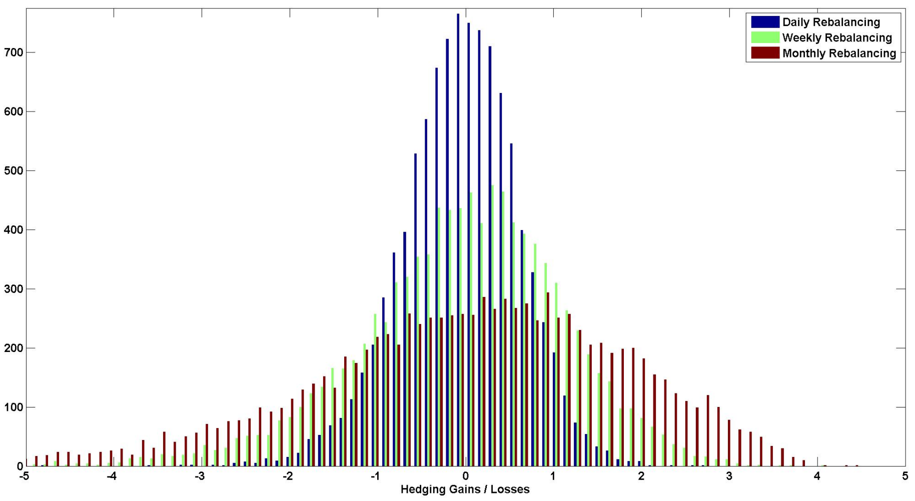

Figure 5.1 The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the delta hedging scheme. Euler Monte Carlo methods, with 10 000 stock paths, were used to produce this plot.

Figure 5.1 displays the performance of the delta hedging scheme in the Heston model. As can be seen from the plot, the performance of the scheme improves as the frequency of portfolio rebalancing increases. In spite of this, however, the scheme quite inadequate in hedging the volatility risk implicit in the vanilla call. This is to be expected as delta hedging methods only offset the risk resulting from the randomness in the stock price process. If we want to improve our hedging schemes in a stochastic volatility environment, we need to include traded assets in our hedging portfolio that will eliminate the volatility risk in the model.

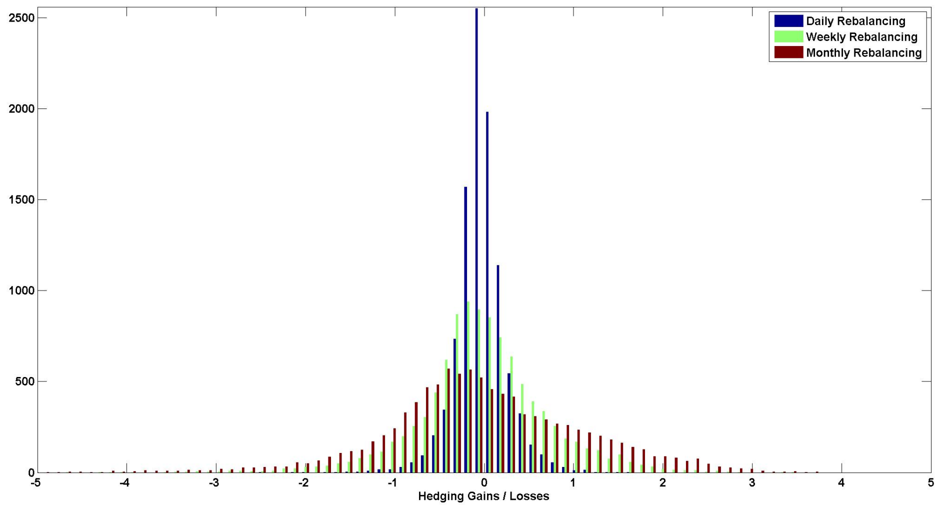

Figure 5.2 The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the delta-sigma hedging scheme. Euler Monte Carlo methods, with 10 000 stock paths, were used to produce this plot.

Consequently, we turn our attention to the delta-sigma hedging scheme, the performance of which is shown in Figure 5.2. This entails introducing a market traded option into the hedging portfolio. Immediately, we can see that our hedging performance has markedly improved. Even when rebalancing at monthly intervals the losses and gains on our hedging scheme are all bounded between -3 and 3 for the simulation parameters used. Including the second option, written on the same underlying, in the portfolio allows us to offset the risk arising from the random component in the volatility process. Our hedging performance has improved to the extent that our only hedging error arises from being forced to rebalance the portfolio at discrete points in time.

Introducing a third option into our portfolio provides us with a delta, vega and gamma hedge for the option. The gamma hedging component of this scheme greatly reduces the hedging errors in the method that arise from discrete portfolio rebalancing. Figure 5.3 shows the success of this scheme in improving our hedging performance. Using this method, and

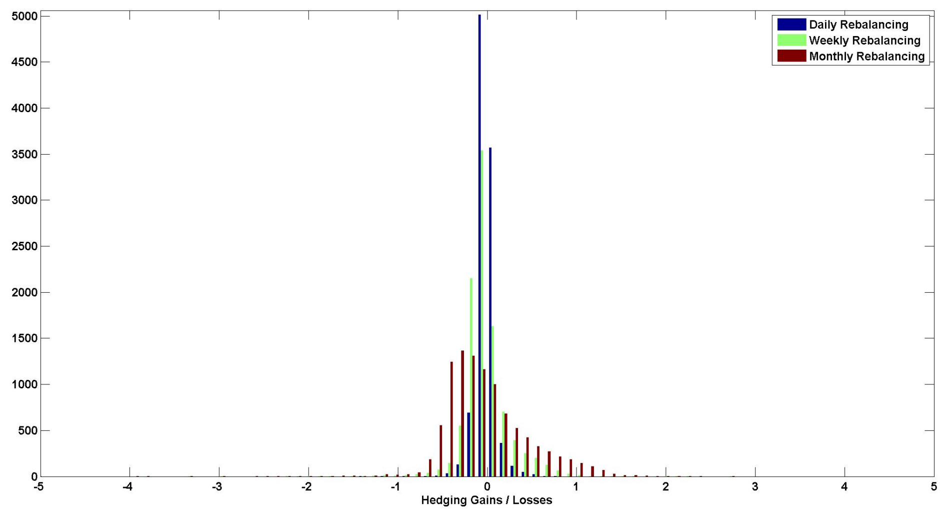

Figure 5.3 The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the delta-sigma-gamma hedging scheme. Euler Monte Carlo methods, with 10 000 stock paths, were used to produce this plot.

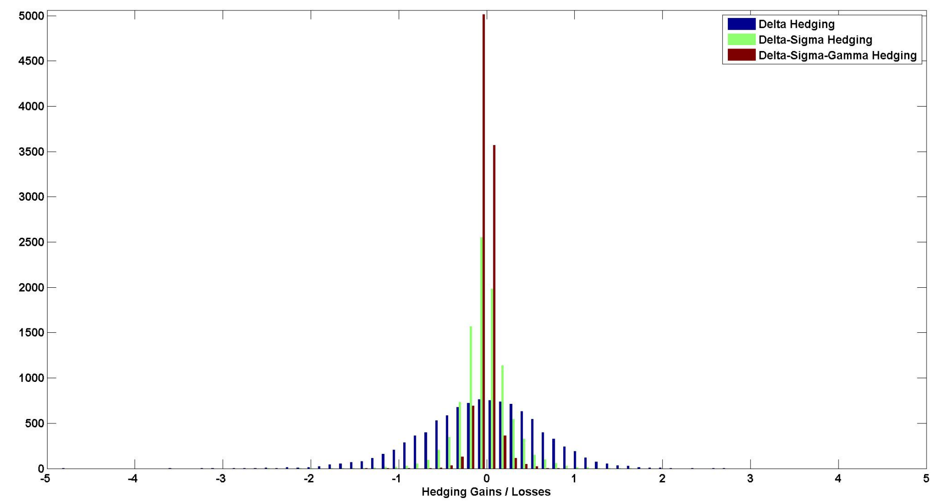

Figure 5.4 In this plot, we compare the three hedging schemes (with daily portfolio rebalancing) for the Heston model. The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the three hedging schemes. Euler Monte Carlo methods, with 10 000 stock paths, were used to produce this plot.

rebalancing our portfolio at daily intervals essentially allows us to limit our hedging errors to between -0.5 and 0.5.

Comparing the different hedging schemes under daily rebalancing, we can see the extent to which the delta-sigma-gamma method outperforms the others. By treating vanilla options as market traded assets and using them for the purpose of hedging the volatility risk in the Heston model, as well as to reduce the discrete rebalancing error, we get a significantly improved hedging methodology.

5.3 Hedging Strategies for the Bates and SVJJ Models

A natural extension of the previous section is to consider the hedging of options priced under the Bates and SVJJ models. Thus, we now turn our attention to analysing the effectiveness of the delta-sigma-gamma hedging scheme when jump processes are introduced to the stock price and variance processes. We expect the performance of this scheme to worsen with the introduction of jumps, since there are no assets in the replicating portfolio to hedge the jump risk. Unlike in the previous sections on hedging under Heston dynamics, we do not introduce the theory for risk-neutral transformations under the dynamics of stochastic volatility jump-diffusion models here. Rather, we briefly consider this topic in Appendix D and ask the interested reader to refer there for this purpose.

5.3.1 Hedging Simulations for the Bates and SVJJ Models

For simplicity, we set the market price of volatility risk to zero and assume no compensation for jump risk. This means that we assume the same parameters for the jump components in the two models under both the risk-neutral and the real-world measures. We also set $\mu = r$. This gives us the simplest possible specification for simulating hedging strategies under the two models. The other parameters for the models are as follows:

Bates Model Parameters:		SVJJ Model Parameters:	
κ	$= 1$	κ	$= 3.5$
θ	$= 0.04$	θ	$= 0.008$
σ_v	$= 0.2$	σ_v	$= 0.2$
ρ	$= 0$	ρ	$= 0$
V_0	$= 0.04$	V_0	$= 0.008$
λ_J	$= 1.5$	λ_J	$= 1.5$
μ_J	$= 0$	μ_S	$= -0.04$
σ_S	$= 0.15$	σ_S	$= 0.0001$
		μ_V	$= 0$
		ρ_J	$= 0.04$

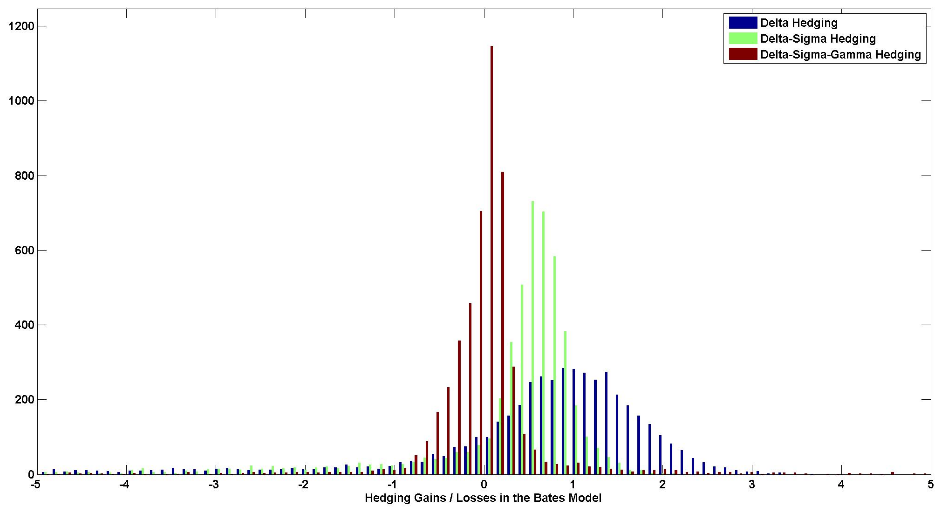

Figure 5.5 Comparison of the three hedging schemes (with daily portfolio rebalancing) for the Bates model. The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the three hedging schemes. Euler Monte Carlo methods, with 5000 stock paths, were used to produce this plot.

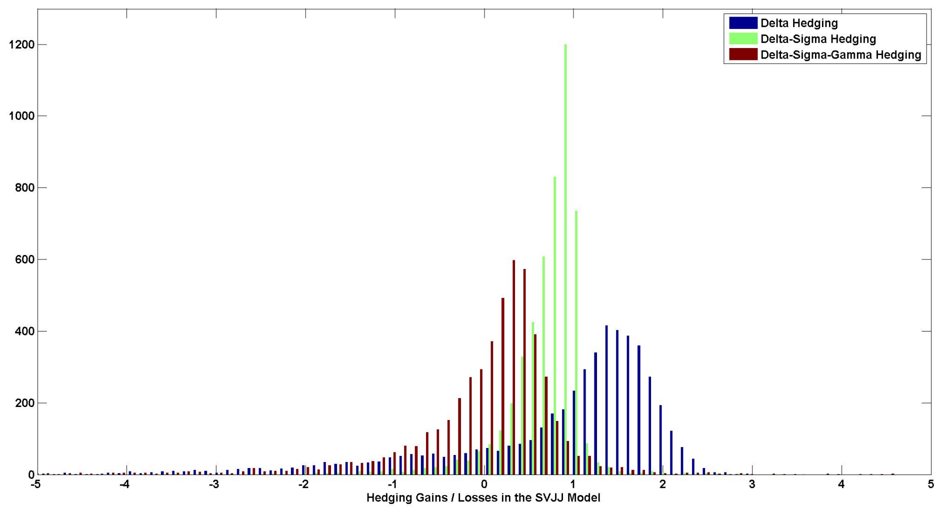

Figure 5.6 Comparison of the three hedging schemes (with daily portfolio rebalancing) for the SVJJ model. The plot gives the distribution of absolute gains (on the positive part of the horizontal axis) and losses (on the negative part of the horizontal axis) from the three hedging schemes. Euler Monte Carlo methods, with 5000 stock paths, were used to produce this plot.

We can see from Figures 5.5 and 5.6 that the hedging schemes under-perform in these two models, relative to their performance in the Heston model (see Figure 5.4). This is due to the inclusion of jumps in the stock price and volatility processes of the models. Even under the delta-sigma-gamma hedging scheme, there are no assets in the replicating portfolio to eliminate the jump risk. In fact, we would need an infinite number of hedging instruments in the replicating portfolio to fully offset the jump risk. The magnitudes of the parameters for the jump processes obviously have a direct influence on the hedging performance in the models. The larger the sizes of the jumps and the more frequently they occur, the worse these methods will perform.

5.3.2 A Comment on Hedging Strategies when Jumps are Involved

In the Black-Scholes-Merton world, a dynamic delta hedging scheme is sufficient to perfectly hedge a vanilla option contract. Moving into a stochastic volatility environment, as in the Heston model, we find ourselves facing two sources of risk. The first of these is the same source of risk in the Black-Scholes model, that being the driving Brownian motion in the stock price process. The second source of randomness comes from the driving Brownian motion in the volatility process. A delta hedging scheme, as well as a vega hedging scheme are now required to immunize an option portfolio against random stock price and volatility movements. This adds some complexity to our hedging scheme, but an almost perfect hedge against random fluctuations is nonetheless possible by adding a liquidly traded option based on the same underlying to our hedging portfolio. Moving to a world where jumps occur in the price and/or the volatility of a stock, we are no longer able to perfectly hedge all random fluctuations (not even in theory). Instead, we now have to attempt to minimise our hedging error.

In the case where a finite number jumps occur in the stock price (and/or volatility) at random times and having random jump sizes, we would need an infinite number of hedging instruments to eliminate the jump risk completely. This is because we would need a different hedging instrument for each possible jump size. Such a strategy would be impossible to construct (Gatheral [25]). Instead, we can select a finite number of suitable jump amplitudes to hedge against in an attempt to minimise our hedging error due to jumps. This can be done in addition to a delta and vega hedging scheme to minimise our overall hedging loss.

He et al. [27] provide an insightful investigation into the hedging of jump diffusion models. In their paper, they consider a jump-diffusion model without the stochastic volatility component and show the hedging performance when various numbers of jump sizes are chosen to hedge against. They report that a hedging portfolio consisting of the underlying stock (to enforce delta-neutrality), the risk free asset and ten appropriately chosen options yields almost no hedging error. The number of hedging options that can be incorporated into the hedging portfolio is largely governed by the number of liquidly tradable contracts available in the market. Such a procedure could easily be extended to the hedging problem in the Bates and SVJJ models.

In summary, we have seen that by including options as hedging instruments in our replication portfolio, we can hedge against the effects of stochastic volatility. The performance of these schemes is also reasonable if we apply them to situations where jumps occur in the model processes, as long as the jumps are fairly infrequent and are generally not very large. To provide a better hedging portfolio for models that permit jumps, we need to consider more hedging options to minimise the jump risk.

Chapter 6

Conclusion

The purpose of this dissertation has been to provide a consistent framework for approaching the themes of pricing, calibration and hedging with respect to the Heston, Bates and SVJJ models. From a South African perspective, we have written this report with the view of better modelling and risk management of ALSI futures options in mind. In the first chapter, we saw the parameter effects for the different models on the implied volatility surface and the distribution of stock price returns. We also compared these to empirical observations of the JSE Top 40 and S&P 500 indices. Notably, we observed that the presence of jumps in the Bates and SVJJ models enables them to produce larger price movements than the Heston model is able to produce. This means that the two jump models are also more capable of producing stock price returns distributions with fat tails and which exhibit skewness. Such characteristics agree with the price movements and returns distributions of the two indices.

Thereafter, we considered pricing methods for the models. Specifically, we examined the application of the fast Fourier transform pricing method and Monte Carlo pricing methods to the three models. The FFT method provides a fast and accurate means of pricing vanilla options. The Monte Carlo methods, on the other hand, are much slower to implement. Their strength arises, however, in their ability to provide pricing solutions for more complex financial instruments, as well as their application to the simulation of hedging strategies.

The third chapter gave a comprehensive investigation into different calibration to option price data techniques for the models as well as the application of these to actual market calibration. It drew on the FFT pricing framework explored in the second chapter to devise such schemes. We found here that a genetic algorithm routine, in conjunction with the MATLAB lsqnonlin optimiser, provided the best means of fitting the models to synthetically generated data. These results were in comparison to those obtained by implementing calibration routines which made use of the adaptive simulated annealing framework as well as the lsqnonlin optimiser on its own. Performing calibration in this way allowed us to test the effectiveness of the different calibration techniques before considering calibration to market data. The calibration of the three models to ALSI and S&P 500 options data was performed by making use of the GA optimisation routine. We found that the ASLI implied volatility skews were steep, but quite linear, and that the dynamics of the Heston model matched these dynamics better than the other two models did. Calibration to the S&P 500 options data, however, revealed that the jump models provide a better description of market dynamics. More specifically, for the data analysed, the Bates model gave the best fit to this data and gave the best performance in the calibration routine. The SVJJ model seemed to be slightly over parameterised, making it more difficult to use. We thus found that the Heston and Bates models were the most robust of the three and that the GA calibration scheme provided the best means by which to fit the models to market data. An important point to note is that we did not investigate the temporal stability of parameters and recommend this as further research.

The final chapter of the dissertation examined hedging techniques for the models by calculating option price sensitivities and constructing hedging portfolios which made use of these. For the Heston model, a delta-sigma-gamma hedging scheme provided the best performance. It eliminated all the return and volatility risk from the model. It also yielded the best performance for discretely rebalanced portfolios. Furthermore, the application of this method to the jump models gave reasonable results. The lack of hedging instruments for the jump risk in the models meant that the method was not as effective as it was for the Heston model however. An examination of current literature revealed that constructing perfect hedging schemes for options priced under the dynamics of jump models — with the jump magnitudes being defined by some continuous distribution — is impossible. Instead, hedging portfolios need to incorporate numerous traded vanilla options to minimise hedging errors.

Further work can always be undertaken in studying these three models. Notably, the use of such models for pricing exotic options and the implementation of the techniques explored in this project on faster computing infrastructures would form very relevant research topics. The work in this dissertation has, however, focused on the main topics in the study of stochastic volatility and jump processes. We hope that this will provide practitioners with a robust framework for implementing these models.

Appendix A

Risk-Neutral Dynamics for Jump Diffusion Models

In this chapter, we supply an intuitive explanation for the form of the stock price process for a jump-diffusion model under the risk-neutral measure.

The form of the stock price process of a jump diffusion model (specifically, the stock price process of the Bates and the SVJJ models) is

$$dS_t \;=\; \bar{\mu} S_t dt + \sqrt{V_t} S_t d\widetilde{W}_t + J S_t d\widetilde{N}_t$$

under the risk-neutral measure, where $\bar{\mu}$ is the drift term, $\widetilde{W}$ is a Brownian motion and $\widetilde{N}$ is a Poisson process (with intensity λ). Note that we omit the enumeration of the Brownian motion (seen in the main part of the text) for the sake of brevity. Now, as is the usual case, we wish for the discounted version of this process to be a martingale under the risk-neutral measure. An application of Itô's formula for general semimartingales yields

$$\begin{aligned} d\left(e^{-rt}S_t\right) &= (-r+\bar{\mu})\left(e^{-rt}S_t\right)dt + \sqrt{V_t}\left(e^{-rt}S_t\right)d\widetilde{W}_t + J\left(e^{-rt}S_t\right)d\widetilde{N}_t. \\ &= (-r+\bar{\mu})\left(e^{-rt}S_t\right)dt + \sqrt{V_t}\left(e^{-rt}S_t\right)d\widetilde{W}_t + J\left(e^{-rt}S_t\right)d\left(\widetilde{N}_t - \lambda t\right) \\ &\quad + \lambda J\left(e^{-rt}S_t\right)dt \\ &= (-r+\bar{\mu}+\lambda J)\left(e^{-rt}S_t\right)dt + \sqrt{V_t}\left(e^{-rt}S_t\right)d\widetilde{W}_t + J\left(e^{-rt}S_t\right)d\left(\widetilde{N}_t - \lambda t\right). \end{aligned}$$

Now, from a purely heuristic point of view, this process is a martingale under the risk-neutral measure if

$$\widetilde{\mathbb{E}}\left[d\left(e^{-rt}S_t\right)\right] \;=\; 0,$$

where $\widetilde{\mathbb{E}}\left[\cdot\right]$ denotes the expectation operator under the risk-neutral measure. It is a common result that both the Brownian motion, $\widetilde{W}_t$, and the compensated Poisson process, $\widetilde{N}_t - \lambda t$, are martingales, null at zero, under the risk-neutral measure (by Girsanov's Theorem), and so we require

$$\begin{aligned} 0 &= \widetilde{\mathbb{E}}\left[-r + \bar{\mu} + \lambda J\right] \\ &= -r + \bar{\mu} + \lambda \mu_J, \end{aligned}$$

with

$$\mu_J = \exp\left\{\mu_S + \frac{\sigma_S^2}{2}\right\} - 1,$$

where μ_S and σ_S^2 are the distributional parameters for the log-normally distributed jump sizes (more specifically, for the distribution of $1 + J$). Consequently,

$$\bar{\mu} = r - \lambda \mu_J$$

and so,

$$dS_t = (r - \lambda \mu_J) S_t dt + \sqrt{V_t} S_t d\widetilde{W}_t + J S_t d\widetilde{N}_t.$$

This is the result that we see for the stock-price processes of both the Bates and SVJJ models.

Appendix B

Model Characteristic Functions

B.1 The Heston Characteristic Function

We follow the works of Gatheral [25] and Zhu [59] in our treatment of the Heston characteristic function. As we have seen before, the log-stock price and volatility processes of the Heston model can be expressed as,

$$\begin{aligned} d\log S_t &= rdt - \frac{1}{2}V_t dt + \sqrt{V_t}\left[\rho d\widetilde{Z}_t^{(2)} + \sqrt{1-\rho^2}d\widetilde{Z}_t^{(1)}\right] \\ dV_t &= \kappa\left(\theta - V_t\right)dt + \sigma_v\sqrt{V_t}d\widetilde{Z}_t^{(2)}. \end{aligned}$$

The characteristic function for the log-stock price is then given by,

$$\begin{aligned} \phi_{\log(S_T)}(u) &= \widetilde{\mathbb{E}}\left[\exp\left\{iu\log\left(S_T\right)\right\}\right] \\ &= \widetilde{\mathbb{E}}\left[\exp\left\{iu\left(\log\left(S_0\right) + rT - \frac{1}{2}\int_0^T V_t dt + \rho\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)}\right.\right.\right. \\ &\quad \left.\left.\left. + \sqrt{1-\rho^2}\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(1)}\right)\right\}\right] \\ &= \exp\left\{iu\left(\log\left(S_0\right) + rT\right)\right\}\widetilde{\mathbb{E}}\left[\exp\left\{-\frac{iu}{2}\int_0^T V_t dt\right.\right. \\ &\quad \left.\left. + iu\rho\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)} + \frac{(iu)^2}{2}\left(1-\rho^2\right)\int_0^T V_t dt\right\}\right]. \end{aligned}$$

Now, integrating dV_t, we get

$$V_T = V_0 + \kappa\theta T - \kappa\int_0^T V_t dt + \sigma_v\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)},$$

and so,

$$\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)} = \frac{1}{\sigma_v}\left[V_T - V_0 - \kappa\theta T + \kappa\int_0^T V_t dt\right].$$

Inserting this into our expression for the characteristic function, we get

$$\begin{aligned}
\phi_{\log(S_T)}(u) &= \exp\left\{iu\left(\log\left(S_0\right)+rT\right)\right\}\widetilde{\mathbb{E}}\left[\exp\left\{-\frac{iu}{2}\int_0^T V_t dt\right.\right.\\
&\quad + \frac{iu\rho}{\sigma_v}\left[V_T - V_0 - \kappa\theta T + \kappa\int_0^T V_t dt\right]\\
&\quad \left.\left. + \frac{(iu)^2}{2}\left(1-\rho^2\right)\int_0^T V_t dt\right\}\right]\\
&= \exp\left\{iu\left(\log\left(S_0\right)+rT\right) - s_2\left(V_0+\kappa\theta T\right)\right\}\\
&\quad \times \widetilde{\mathbb{E}}\left[\exp\left\{-s_1\int_0^T V_t dt + s_2 V_T\right\}\right],
\end{aligned}$$

where

$$\begin{aligned}
s_1 &= -iu\left(\frac{\rho\kappa}{\sigma_v} - \frac{1}{2} + \frac{iu}{2}\left(1-\rho^2\right)\right)\\
s_2 &= \frac{iu\rho}{\sigma_v}.
\end{aligned}$$

Through an application of the Feynman-Kac Theorem, Zhu [59] shows that

$$\widetilde{\mathbb{E}}\left[\exp\left\{-s_1\int_0^T V_t dt + s_2 V_T\right\}\right] = \exp\left\{H_T^{(1)}(u)V_0 + H_T^{(2)}(u)\right\},$$

so that

$$\phi_{\log(S_T)}(u) = \exp\left\{iu\left(\log\left(S_0\right)+rT\right) - s_2\left(V_0+\kappa\theta T\right) + H_T^{(1)}(u)V_0 + H_T^{(2)}(u)\right\},$$

where

$$\begin{aligned}
H_T^{(1)}(u) &= \frac{s_2 d\left(1+e^{-dT}\right) - \left(1-e^{-dT}\right)\left(-\frac{iu\rho\kappa}{\sigma_v} + iu - (iu)^2\left(1-\rho^2\right)\right)}{\left(1-ge^{-dT}\right)(\beta+d)}\\
H_T^{(2)}(u) &= \frac{2\kappa\theta}{\sigma_v^2}\log\left(\frac{2d}{\left(1-ge^{-dT}\right)(\beta+d)}e^{\frac{1}{2}(\kappa-d)T}\right)\\
g &= \frac{r_{\text{neg}}}{r_{\text{pos}}}\\
r_{\text{pos/neg}} &= \frac{\beta \pm d}{2\gamma}\\
d &= \sqrt{\beta^2 - 4\alpha\gamma}\\
\alpha &= \frac{\left(-u^2 - iu\right)}{2}\\
\beta &= \kappa - \rho\sigma_v iu\\
\gamma &= \frac{\sigma_v^2}{2}.
\end{aligned}$$

Now, it is fairly simple to show that

$$\begin{aligned}
\left(H_T^{(1)}(u) - s_2\right) V_0 &= r_{\text{neg}} \left[\frac{1-e^{-dT}}{1-ge^{-dT}}\right] V_0 \;=:\; D\left(u,T\right) V_0 \\
H_T^{(2)}(u) - s_2\kappa\theta T &= \kappa\theta \left[r_{\text{neg}}T - \frac{2}{\sigma_v^2}\log\left(\frac{1-ge^{-dT}}{1-g}\right)\right] \;=:\; C\left(u,T\right)\theta,
\end{aligned}$$

which leads us to the commonly used form of the Heston characteristic function:

$$\phi_{\log(S_T)}(u) = \exp\left\{C\left(u,T\right)\theta + D\left(u,T\right)V_0 + iu\left(\log\left(S_0\right) + rT\right)\right\}. \tag{B.1}$$

B.2 The Bates Characteristic Function

Drawing from Zhu [59], we derive the characteristic function for the log-stock price in the Bates model. The log-stock price and volatility processes of the Bates model can be expressed as,

$$\begin{aligned}
d\log S_t &= rdt - \lambda\mu_J dt - \frac{1}{2}V_t dt + \sqrt{V_t}\left[\rho d\widetilde{Z}_t^{(2)} + \sqrt{1-\rho^2}d\widetilde{Z}_t^{(1)}\right] + \log\left(1+J\right)dN_t \\
dV_t &= \kappa\left(\theta - V_t\right)dt + \sigma_v\sqrt{V_t}d\widetilde{Z}_t^{(2)},
\end{aligned}$$

where $\log\left(1+J\right)$ has a Normal $\left(\log\left(1+\mu_J\right) - \frac{\sigma_S^2}{2}, \sigma_S^2\right)$ distribution. Now, the characteristic function of the log-stock price is given by

$$\begin{aligned}
\phi_{\log(S_T)}(u) &= \widetilde{\mathbb{E}}\left[\exp\left\{iu\log\left(S_T\right)\right\}\right] \\
&= \widetilde{\mathbb{E}}\left[\exp\left\{iu\left(\log\left(S_0\right) + rT - \lambda\mu_J T - \frac{1}{2}\int_0^T V_t dt + \rho\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)}\right.\right.\right. \\
&\quad + \left.\left.\left.\sqrt{1-\rho^2}\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(1)} + \log\left(1+J\right)\int_0^T dN_t\right)\right\}\right] \\
&= \exp\left\{iu\left(\log\left(S_0\right) + rT\right)\right\}\widetilde{\mathbb{E}}\left[\exp\left\{-\frac{iu}{2}\int_0^T V_t dt\right.\right. \\
&\quad + \left. iu\rho\int_0^T \sqrt{V_t}d\widetilde{Z}_t^{(2)} + \frac{(iu)^2}{2}\left(1-\rho^2\right)\int_0^T V_t dt\right\} \\
&\quad \times \left.\exp\left\{-iu\lambda\mu_J T + \lambda T\left(e^{iu\log(1+J)} - 1\right)\right\}\right] \\
&= \phi_{\log(S_T)}^{(\mathrm{H})}(u) \times \widetilde{\mathbb{E}}\left[\exp\left\{-iu\lambda\mu_J T + \lambda T\left(e^{iu\log(1+J)} - 1\right)\right\}\right],
\end{aligned}$$

where $\phi_{\log(S_T)}^{(\mathrm{H})}(u)$ is the Heston characteristic function given by (B.1). Now,

$$\begin{aligned}
\widetilde{\mathbb{E}}\left[\exp\left\{\lambda T\left(e^{iu\log(1+J)} - 1\right)\right\}\right] &= \exp\left\{\lambda T\left(e^{iu\log(1+\mu_J) - \frac{1}{2}iu\sigma_S^2 + \frac{1}{2}(iu)^2\sigma_S^2} - 1\right)\right\} \\
&= \exp\left\{\lambda T\left(\left(1+\mu_J\right)^{iu} e^{\sigma_S^2\frac{iu}{2}(iu-1)} - 1\right)\right\}.
\end{aligned}$$

Thus,

$$\begin{aligned}\phi_{\log(S_T)}(u) &= \phi^{(\mathrm{H})}_{\log(S_T)}(u) \times \exp\left\{-iu\lambda\mu_J T + \lambda T\left(\left(1+\mu_J\right)^{iu} e^{\sigma_S^2 \frac{iu}{2}(iu-1)} - 1\right)\right\} \\ &=: \phi^{(\mathrm{H})}_{\log(S_T)}(u) \times \exp\left\{P\left(u\right)\lambda T\right\}. \qquad \text{(B.2)}\end{aligned}$$

B.3 The SVJJ Characteristic Function

The characteristic function of the log-stock price in the SVJJ model can be evaluated in a similar way. Again, it is formed by multiplying the characteristic function of the log-stock price in the Heston model with the characteristic function of the jump processes in the log-stock price and volatility processes. Works by Duffie et. al. [22], Gatheral [25] and Zhu [59] give the derivation of this characteristic function.

Appendix C

ASAMIN Installation Instructions

The following instructions are adapted from those given by Moins [44]. The installation procedure for Windows is as follows:

1. Ensure that there is a C compiler installed on the PC that will be implementing the ASA routine.

2. Download the ASA packages from http://www.ingber.com/#ASA.

3. Download the ASAMIN packages from http://ssakata.sdf.org/software/ and place them in their own directory.

4. Place the ASA files asa.c, asa.h and asa_user.h in the same directory in which the ASAMIN packages were placed.

5. Open the MATLAB console and change the "current folder" to the directory containing the ASAMIN and relevant ASA files.

6. In the MATLAB command window, type:

   ```
   mex asamin.c asa.c -DUSER_ACCEPTANCE_TEST#TRUE -DUSER_ASA_OUT#TRUE
   -DDBL_MIN#2.2250738585072014e-308.
   ```

   MATLAB will then create a MEX file, allowing the user to interface with the C-language ASA code via MATLAB.

Importantly, the pathname for the directory containing the ASAMIN, relevant ASA and MEX files must be incorporated into every script that calls the asamin function. This can be done through the use of the *addpath* command in MATLAB.

Appendix D

Measure Changes for Jump Diffusion Models

In this chapter, we introduce the theory for measure changes for processes comprising a compound Poisson process and a Brownian motion. Later, we apply this theory to the cases of the Bates and the SVJJ models. We make a special note here that much of the content in this section is an adaption of the works by Broadie et al. [10] and Shreve [53].

D.1 A Change of Measure for a Compound Poisson Process as well as a Brownian Motion

The stock price and volatility processes in the Bates and SVJJ models are comprised of some combination of a drift process, a diffusion process (in terms of a Brownian motion) and a jump process (in terms of a compound Poisson process). A change of measure under these models, to an equivalent martingale measure, must therefore be constructed by changing the measure for both the Brownian motion and the Poisson process. We start by defining a compound Poisson process.

Definition 6 (Compound Poisson Process)**.** Let N_t be a Poisson process with jump intensity λ, defined on a probability space $(\Omega, \mathcal{F}_t, \mathcal{F}, \mathbb{P})$, where $\mathbb{P}$ is the real-world measure on that space, $\mathcal{F}_t$ is our filtration process, $t \in [0, T]$ and $T > 0$. Suppose that $X_1, X_2, \ldots$ is a series of independently and identically distributed random variables that are also independent of N_t. A compound Poisson process is then defined by:

$$Y_t = \sum_{i=1}^{N_t} X_i,$$

where the X_i represent the magnitudes of the jumps in the compound Poisson process.

Now, considering the definition above, suppose that the X_i have a common density $g(x)$. The Radon-Nikodým derivative process

$$\zeta_t = e^{(\lambda-\widetilde{\lambda})t}\prod_{i=1}^{N_t}\frac{\widetilde{\lambda}\widetilde{g}(X_i)}{\lambda g(X_i)}$$

facilitates a change of measure from $\mathbb{P}$ to the risk-neutral measure $\widetilde{\mathbb{P}}$, where now, $\widetilde{Y}_t$ is a $\widetilde{\mathbb{P}}$-compound Poisson process with intensity $\widetilde{\lambda}$ and jump size distribution $\widetilde{g}(x)$. It is possible to show that ζ_t is a martingale and, more specifically, that $\mathbb{E}[\zeta_T]=1$ for all values of $t>0$ (see Shreve [53]). Consequently, we have that

$$\widetilde{\mathbb{P}}(A) = \int_A \zeta_T d\mathbb{P} \text{ for all } A\in\mathcal{F}.$$

This completes the measure change for the compound Poisson process. Importantly, we have seen that a measure change for a compound Poisson processes changes the jump intensity and the jump distribution of the process. We now define a Brownian motion on the same probability space and consider the change of measure for both the compound Poisson process and the Brownian motion.

Consider the same probability space as before, upon which is defined a Brownian motion, Z_t, and a compound Poisson process, Y_t. We assume that the two processes are independent and both adapted to the filtration $\mathcal{F}_t$. Furthermore, take $\widetilde{\lambda}$ and λ to be positive numbers, let $\widetilde{g}(x)$ be a density function that agrees with $g(x)$ in assigning zero probability to certain intervals and define Θ_t to be an adapted process. Define

$$\begin{aligned}
\xi_t^{(1)} &= \exp\left\{-\int_0^t \Theta_v dW_v - \frac{1}{2}\int_0^t (\Theta_v)^2 dv\right\}\\
\xi_t^{(2)} &= e^{(\lambda-\widetilde{\lambda})t}\prod_{i=1}^{N_t}\frac{\widetilde{\lambda}\widetilde{g}(X_i)}{\lambda g(X_i)}\\
\xi_t &= \xi_t^{(1)}\xi_t^{(2)}.
\end{aligned}\tag{D.1}$$

Again, we can show that ξ_t is a martingale and that $\mathbb{E}[\xi_T]=1$ (see Shreve [53]). Now, ξ_t is the Radon-Nikodým derivative process that allows us to change measures from the real-world measure to the risk-neutral measure since

$$\widetilde{\mathbb{P}}(A) = \int_A \xi_T d\mathbb{P} \text{ for all } A\in\mathcal{F}.$$

This leads us to assert the following:

Definition 7 (Measure Change for a Compound Poisson Process and a Brownian Motion)**.** Under the measure $\widetilde{\mathbb{P}}$, $\widetilde{Y}_t$ is a compound Poisson process with jump intensity $\widetilde{\lambda}$ and i.i.d.

jump sizes according to the risk-neutral distribution $\widetilde{g}(x)$. Furthermore, $\widetilde{W_t}$ is a $\widetilde{\mathbb{P}}$-Brownian motion such that

$$d\widetilde{W_t} = dW_t + \Theta_t dt,$$

with $\widetilde{Y_t}$ and $\widetilde{W_t}$ independent.

This now gives us the basic structure for performing measure changes for jump-diffusion models. The framework above is easily extended to the multidimensional case, allowing us to perform measure changes in the Bates and SVJJ models.

D.2 A Change of Measure in the Bates Model

Under the real-world probability measure, $\mathbb{P}$, we can define the following dynamics for the Bates model:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{V_t} S_t dZ_t^{(1)} + S_t d\left(Q_t - \lambda \mu_J t\right) \\ dV_t &= \kappa\left(\theta - V_t\right) dt + \sigma_v \sqrt{V_t}\left[\rho dZ_t^{(1)} + \sqrt{1-\rho^2} dZ_t^{(2)}\right], \end{aligned}$$

where $Z_t^{(1)}$ and $Z_t^{(2)}$ are independent Brownian motions, Q_t is a compound Poisson process with jump intensity λ and jump sizes specified by a log-normal $\left(\mu_S, \sigma_S^2\right)$ distribution and

$$\mu_J = \exp\left\{\mu_S + \frac{\sigma_S^2}{2}\right\} - 1.$$

The Radon-Nikodým derivative process is given by:

$$\xi_t = \xi_t^{(1)} \xi_t^{(2)},$$

where,

$$\begin{aligned} \xi_t^{(1)} &= \exp\left\{-\frac{1}{2}\int_0^t \left[\left(\Theta_u^{(1)}\right)^2 + \left(\Theta_u^{(2)}\right)^2\right] du - \int_0^t \Theta_u^{(1)} dZ_u^{(1)} - \int_0^t \Theta_u^{(2)} dZ_u^{(2)}\right\} \\ \xi_t^{(2)} &= e^{\left(\lambda - \widetilde{\lambda}\right)t} \prod_{i=1}^{N_t} \frac{\widetilde{\lambda}\widetilde{g}\left(X_i\right)}{\lambda g\left(X_i\right)}, \end{aligned}$$

such that

$$\begin{aligned} d\widetilde{Z}_t^{(1)} &= dZ_t^{(1)} + \Theta_t^{(1)} dt \\ d\widetilde{Z}_t^{(2)} &= dZ_t^{(2)} + \Theta_t^{(2)} dt. \end{aligned}$$

Now, λ and $\widetilde{\lambda}$ are the jump intensities of the compound Poisson process under $\mathbb{P}$ and $\widetilde{\mathbb{P}}$ respectively. Moreover, $g\left(x\right)$ and $\widetilde{g}\left(x\right)$ specify log-normal jump-size distributions under $\mathbb{P}$

and $\widetilde{\mathbb{P}}$ respectively. As in the Heston model, $\Theta_t^{(1)}$ and $\Theta_t^{(2)}$ are the market prices of return and volatility risk. As usual, ξ_t allows us to change measure from the real-world measure to the risk-neutral measure. Defining

$$\begin{aligned}\Theta_t^{(1)} &= \frac{\mu - r - \left(\lambda\mu_J - \widetilde{\lambda}\widetilde{\mu}_J\right)}{\sqrt{V_t}} \\ \Theta_t^{(2)} &= \gamma_t\end{aligned}$$

allows us to change the parameters of the Bates model processes in order to specify the model under the risk-neutral measure. Thus, under $\widetilde{\mathbb{P}}$,

$$\begin{aligned}dS_t &= rS_t dt + \sqrt{V_t} S_t d\widetilde{Z}_t^{(1)} + S_t d\left(\widetilde{Q}_t - \widetilde{\lambda}\widetilde{\mu}_J t\right) \\ dV_t &= \kappa\left(\widetilde{\theta} - V_t\right)dt + \sigma_v\sqrt{V_t}\left[\rho d\widetilde{Z}_t^{(1)} + \sqrt{1-\rho^2}d\widetilde{Z}_t^{(2)}\right],\end{aligned}$$

where $\widetilde{Z}_t^{(1)}$ and $\widetilde{Z}_t^{(2)}$ are independent $\widetilde{\mathbb{P}}$-Brownian motions, $\widetilde{Q}_t$ is a $\widetilde{\mathbb{P}}$-compound Poisson process with intensity $\widetilde{\lambda}$ and log-normal $\left(\widetilde{\mu}_S, \widetilde{\sigma}_S^2\right)$ distributed jumps and

$$\widetilde{\mu}_J = \exp\left\{\widetilde{\mu}_S + \frac{\widetilde{\sigma}_S^2}{2}\right\} - 1.$$

As was the case in the Heston model,

$$\widetilde{\theta} = \theta - \frac{\sigma_v\sqrt{V_t}\rho\Theta_t^{(1)}}{\kappa} - \frac{\sigma_v\sqrt{V_t}\sqrt{1-\rho^2}\Theta_t^{(2)}}{\kappa}.$$

Note, again, that this transformation is not unique. As in the Heston model, there are multiple martingale measures equivalent to $\mathbb{P}$.

D.3 A Change of Measure in the SVJJ Model

The change of measure in the SVJJ model is almost identical to that in the Bates model. Specifying the SVJJ processes under the real-world measure $\mathbb{P}$, we have:

$$\begin{aligned}dS_t &= \mu S_t dt + \sqrt{V_t} S_t dZ_t^{(1)} + S_t d\left(Q_t^{(s)} - \lambda\mu_J t\right) \\ dV_t &= \kappa\left(\theta - V_t\right)dt + \sigma_v\sqrt{V_t}\left[\rho dZ_t^{(1)} + \sqrt{1-\rho^2}dZ_t^{(2)}\right] + dQ_t^{(v)},\end{aligned}$$

where $Q_t^{(s)}$ and $Q_t^{(v)}$ are two compound Poisson processes and

$$\mu_J = \frac{\exp\left\{\mu_S + \frac{\sigma_S^2}{2}\right\}}{1 - \rho_J\mu_V} - 1.$$

If we define N_t to be a Poisson process with intensity λ, then

$$\begin{aligned} Q_t^{(s)} &= \sum_{i=1}^{N_t} J_i \\ Q_t^{(v)} &= \sum_{i=1}^{N_t} X_i, \end{aligned}$$

where X_i has an Exponential (μ_V) distribution and J_i has a log-normal $\left(\mu_S + \rho_J X_i, \sigma_S^2\right)$ distribution. Again, the Radon-Nikodým derivative process is given by:

$$\xi_t = \xi_t^{(1)} \xi_t^{(2)},$$

where $\xi_t^{(1)}$ is the same as that in the Bates model and Broadie et al. [10] show that

$$\xi_t^{(2)} = \prod_{i=1}^{N_t} \frac{\widetilde{\lambda}\widetilde{\pi}\left(W_i\right)}{\lambda\pi\left(W_i\right)} e^{\left(\int_W \lambda\pi(W) - \widetilde{\lambda}\widetilde{\pi}(W) dW\right)t},$$

where $\pi\left(W\right)$ and $\widetilde{\pi}\left(W\right)$ represent the joint distribution of the jump magnitudes in the two compound Poisson processes in the model, under $\mathbb{P}$ and $\widetilde{\mathbb{P}}$ respectively. Finally, under $\widetilde{\mathbb{P}}$,

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{V_t} S_t d\widetilde{Z}_t^{(1)} + S_t d\left(\widetilde{Q}_t^{(s)} - \widetilde{\lambda}\widetilde{\mu}_J t\right) \\ dV_t &= \kappa\left(\widetilde{\theta} - V_t\right) dt + \sigma_v \sqrt{V_t}\left[\rho d\widetilde{Z}_t^{(1)} + \sqrt{1-\rho^2} d\widetilde{Z}_t^{(2)}\right] + d\widetilde{Q}_t^{(v)}, \end{aligned}$$

where $\widetilde{\theta}$ is defined as before. Furthermore, $\widetilde{Q}_t^{(s)}$ has a log-normal $\left(\widetilde{\mu}_S + \widetilde{\rho}_J X_t, \widetilde{\sigma}_S^2\right)$ distribution and $\widetilde{Q}_t^{(v)}$ has an Exponential $(\widetilde{\mu}_V)$ distribution under the risk-neutral measure and we define

$$\widetilde{\mu}_J = \frac{\exp\left\{\widetilde{\mu}_S + \frac{\widetilde{\sigma}_S^2}{2}\right\}}{1 - \widetilde{\rho}_J \widetilde{\mu}_V} - 1.$$

Again, this transformation is not unique.

Appendix E

Selected MATLAB Code

E.1 Monte-Carlo Methods

Euler-Maruyama Monte Carlo for the Heston Model

The following MATLAB code is a routine for pricing vanilla European call options under the dynamics of the Heston model via the Euler-Maruyama Monte Carlo method. The inputs for the function `HestonEuler` are the Heston model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`) as well as the risk-free rate of interest (`r`), the maturity of the option (`T`), the value of the underlying at the inception of the option (`S0`), the strike price of the option (`K`), as well as the number of simulation paths (`PATHS`) and time-steps (`n`) to be used in estimating the option price. The function then outputs an option price (`Price`) and the simulation time of the routine (`Time`).

```
function [Price, Time] = HestonEuler(KAPPA,THETA,SIGMAv,RHO,V0,r,T,S0,K,...
    PATHS,n)

Timer = clock; %Start the algorithm timer
dt = T/n; %Interval length

V = V0*ones(PATHS,1); %Initialise variance vector
S = S0*ones(PATHS,1); %Initialise price vector

for i = 2:n+1;
   Zv = randn(PATHS,1);
   Zs = RHO*Zv+sqrt(1-RHO^2).*randn(PATHS,1);
   S = S.*exp(r*dt-0.5*max(V,0)*dt+sqrt(max(V,0)).*Zs*sqrt(dt));
   V = V+KAPPA.*(THETA-V).*dt+SIGMAv.*sqrt(max(V,0)).*Zv*sqrt(dt);
end

Price = mean(max(S-K,0)*exp(-r*T)); %Call option price
Time = etime(clock,Timer); %Stop the algorithm timer
```

Euler-Maruyama Monte Carlo for the Bates Model

The following MATLAB code is a routine for pricing vanilla European call options under the dynamics of the Bates model via the Euler-Maruyama Monte Carlo method. The inputs for the function `BatesEuler` are the Bates model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`, `MUJ`, `SIGMAS`, `LAMBDA`) as well as the risk-free rate of interest (`r`), the maturity of the option (`T`), the value of the underlying at the inception of the option (`S0`), the strike price of the option (`K`), as well as the number of simulation paths (`PATHS`) and time-steps (`n`) to be used in estimating the option price. The function then outputs an option price (`Price`) and the simulation time of the routine (`Time`).

```
function [Price, Time] = BatesEuler(KAPPA,THETA,SIGMAv,RHO,V0,MUJ,...
    SIGMAS,LAMBDA,r,T,S0,K,PATHS,n)

Timer = clock; %Start the algorithm timer
dt = T/n; %Interval length

V = V0*ones(PATHS,1); %Initialise variance vector
S = S0*ones(PATHS,1); %Initialise price vector

%Simulate the diffusion part of the stock price process:
for i = 2:n+1;
    Zv = randn(PATHS,1);
    Zs = RHO*Zv+sqrt(1-RHO^2).*randn(PATHS,1);
    S = S + S*(r - LAMBDA*MUJ)*dt + S.*sqrt(max(V,0)).*Zs*sqrt(dt);
    V = V+KAPPA.*(THETA-V).*dt+SIGMAv.*sqrt(max(V,0)).*Zv*sqrt(dt);
end

%Simulate the jump part of the stock price process:
PoissonRVs = poissrnd(LAMBDA*T,PATHS,1); %Randomly generate the
           %number of jumps occurring along each simulated stock path
NT = zeros(PATHS,1); %Initialise cumulative jumpsize vector
```

```
for j = 1:PATHS
    NT(j) = prod(lognrnd(log(1+MUJ)-0.5*SIGMAS^2,SIGMAS,1,PoissonRVs(j)));
        %Generate random jump sizes
end
S = S + S.*(NT-1); %Add jump and diffusion parts together

Price = mean(max(S-K,0)*exp(-r*T)); %Call option price
Time = etime(clock,Timer); %Stop the algorithm timer
```

Euler-Maruyama Monte Carlo for the SVJJ Model

The following MATLAB code is a routine for pricing vanilla European call options under the dynamics of the SVJJ model via the Euler-Maruyama Monte Carlo method. The inputs for the function `SVJJEuler` are the SVJJ model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`, `LAMBDA`, `MUS`, `SIGMAS`, `MUV`, `RHOJ`) as well as the risk-free rate of interest (`r`), the maturity of the option (`T`), the value of the underlying at the inception of the option (`S0`), the strike price of the option (`K`), as well as the number of simulation paths (`PATHS`) and time-steps (`n`) to be used in estimating the option price. The function then outputs an option price (`Price`) and the simulation time of the routine (`Time`).

```
function [Price, Time] = SVJJEuler(KAPPA,THETA,SIGMAv,RHO,V0,LAMBDA,MUS,...
    SIGMAS,RHOJ,MUV,r,T,S0,K,PATHS,n)

Timer = clock; %Start the algorithm timer

MUJ = ( ( exp(MUS + 0.5*(SIGMAS^2)) ) / (1 - RHOJ*MUV) ) - 1;

V = V0*ones(PATHS,1); %Initialise variance vector
S = S0*ones(PATHS,1); %Initialise price vector

for i = 1:PATHS %Simulate along each path

    %Simulate Poisson jumps:
    JumpTimes = PoissonProcess(LAMBDA,T); %Jump times
    Times = [JumpTimes T]; %Jump times and maturity
    TimeDiffs = [Times(1) diff(Times)]; %Times between jumps
    NumJumps = length(JumpTimes); %Number of jumps
    JumpSizesV = [exprnd(MUV,1,NumJumps) 0]; %Jumps sizes in
```

```
                                             %volatility process
    JumpSizesS = [lognrnd(MUS + RHOJ * JumpSizesV(1:end-1),SIGMAS) 1];
        %Jumps sizes in stock price process

    TimeSteps = ceil( (TimeDiffs/T) * n);
        %Divide n (time steps) among jump times
    dt = TimeDiffs ./ TimeSteps; %Time step length

    % Demonstration:
    % If T = 2, Times = [0.23 0.76 1.34 1.9 2], n = 100
    % TimeSteps = [12 27 29 28 5]
    %           <__________________________________________>
    %             |   |         |          |         |   |
    % Times =     0  0.23      0.76       1.34      1.9  2
    % TimeSteps =  12      27        30         28      6    sum = 103

    %Separate jump and diffusion parts
    for j = 1:NumJumps+1
        for k = 1:TimeSteps(j) %Simulate diffusion part between jumps
                Zv = randn;
                Zs = RHO*Zv+sqrt(1-RHO^2).*randn;
                S(i) = S(i) + S(i)*(r - LAMBDA*MUJ)*dt(j) + S(i).*...
                    sqrt(max(V(i),0)).*Zs*sqrt(dt(j));
                V(i) = V(i)+KAPPA.*(THETA-V(i)).*dt(j) + SIGMAv.*....
                    sqrt(max(V(i),0)).*Zv*sqrt(dt(j));
        end
        S(i) = S(i) * JumpSizesS(j); %Include jump in stock price
        V(i) = V(i) + JumpSizesV(j); %Include jump in variance
    end
end

Price = mean(max(S-K,0)*exp(-r*T)); %Call option price
Time = etime(clock,Timer); %Stop the algorithm timer

function times = PoissonProcess(LAMBDA,t)
randt = exprnd(1/LAMBDA); %Generate exponential random variable
tTot = randt;
```

```
count = 0;
times = []; %Jump times
while tTot <= t
    count = count + 1;
    times(count) = tTot; %Jump times
    randt = exprnd(1/LAMBDA); %Generate exponential random variable
    tTot = tTot + randt;
end
```

Exact Simulation Monte Carlo for the Heston Model

The following MATLAB code is a routine for pricing vanilla European call options under the dynamics of the Heston model via the exact simulation scheme. The inputs for the function `HestonExactSimulation` are the Heston model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`) as well as the risk-free rate of interest (`r`), the maturity of the option (`T`), the value of the underlying at the inception of the option (`S0`), the strike price of the option (`K`), as well as the number of simulation paths (`PATHS`) and time-steps (`n`) to be used in estimating the option price. The function then outputs an option price (`Price`) and the simulation time of the routine (`Time`).

```
function [Price Time] = HestonExactSimulation(KAPPA,THETA,SIGMAv,RHO,V0,...
    r,T,S0,K,PATHS,n)

Timer = clock; %Start the algorithm timer
dt = T/n;
t = 0:dt:T;
N = 800;
h = 0.5;

%Stock price simulation:
Vold = V0*ones(PATHS,1); %Initialise variance vector
S = S0*ones(PATHS,1); %Initialise price vector
IVds = zeros(PATHS,1); %Initialise integral sample vector
ops = optimset('TolX',1e-6);

for col = 2:n+1
    Vnew = NoncentralChiSquared(SIGMAv,KAPPA,THETA,t(col),t(col-1),Vold,...
        PATHS); %Sample variance values from non-central chi-square distribution
    RandCDF = rand(PATHS,1);
    for row = 1:PATHS
        [IVds(row) fval exitflag output] = fzero(@(x)FourierInversion(h,...
            Vold(row),Vnew(row),dt,SIGMAv,KAPPA,THETA,N,x) - ....
            RandCDF(row),0.1,ops); %Sample values for integral of V_t
        IVds = max(IVds,0);
    end
    IsqrtVdW = (1 / SIGMAv) * (Vnew - Vold - KAPPA * THETA * dt + ...
        KAPPA * IVds); %Solve for values of integral of sqrt{V_t}
    mu = log(S) + r * dt - 0.5 * IVds + RHO * IsqrtVdW;
    sig = (1 - RHO^2) * IVds;
    S = exp(mu + randn(PATHS,1) .* sqrt(sig)); %Generate stock prices from
                                               %LogN(mu,sig) distribution
    Vold = Vnew;
end

Price = mean(max(S-K,0))*exp(-r*T); %Call option price
Time = etime(clock,Timer); %Stop the algorithm timer

function X = NoncentralChiSquared(SIGMAv,KAPPA,THETA,t,s,Vs,n)
V = 4 * KAPPA * THETA / (SIGMAv^2);
D = ( 4 * KAPPA * exp(-KAPPA*(t-s)) * Vs ) / ( (SIGMAv^2) * ...
    (1 - exp(-KAPPA*(t-s))) );
C = ( ( (SIGMAv^2) * (1 - exp(-KAPPA*(t-s))) ) / (4 * KAPPA) );
X = C * ncx2rnd(V,D,n,1);

function FIn = FourierInversion(h,Vs,Vt,dt,SIGMAv,KAPPA,THETA,N,x)
start = 0.01;
grid = h * (start:N);
integrand = ( sin(grid * x) ./ grid ) .* ...
    real( CharFun(grid,Vs,Vt,dt,SIGMAv,KAPPA,THETA) );
FIn = (2 / pi) * trapz(grid,integrand);

function ChFn = CharFun(a,Vs,Vt,dt,SIGMAv,KAPPA,THETA)
d = 4 * THETA * KAPPA / (SIGMAv^2);
GAMMA =@(a) sqrt( (KAPPA^2) - 2 * (SIGMAv^2) * 1i * a );
A = ( GAMMA(a) .* exp( -0.5 * (GAMMA(a)-KAPPA) * (dt) ) * ...
    (1 - exp(-KAPPA*(dt))) ) ./ ( KAPPA * (1 - exp(-GAMMA(a) * (dt))) );
B1 = (Vs + Vt) / (SIGMAv^2);
B2 = KAPPA * ( 1 + exp(-KAPPA * (dt)) ) / ( 1 - exp(-KAPPA * (dt)) );
B3 = GAMMA(a) .* ( 1 + exp(-GAMMA(a) * (dt)) ) ./ ...
    ( 1 - exp(-GAMMA(a) * (dt)) );
nu = 0.5 * d - 1;
Z1 = sqrt(Vs .* Vt) * 4 .* GAMMA(a) .* exp(-0.5 * GAMMA(a) * (dt)) ...
     ./ ( (SIGMAv^2) * (1 - exp(-GAMMA(a) * (dt))) );
Z2 = sqrt(Vs .* Vt) * 4 * KAPPA * exp(-0.5 * KAPPA * (dt)) ...
    / ( (SIGMAv^2) * (1 - exp(-KAPPA * (dt))) );
C1 = besseli(nu,Z1);
C2 = besseli(nu,Z2);
ChFn = A .* exp(B1 .* (B2 - B3)) .* C1 ./ C2;
```

Exact Simulation Monte Carlo for the Bates and SVJJ Models. The exact simulation schemes for the Bates and SVJJ models are simple extensions of that for the Heston model. They can be constructed from the exact simulation scheme for the Heston model by following the same framework laid out in the Euler-Maruyama methods for the Bates and SVJJ models. We do not include the MATLAB code here for the sake of brevity.

E.2 Fast Fourier Transform Pricing Methods

FFT for the Heston Model

The following MATLAB routine computes the price of a European vanilla call option under the dynamics of the Heston model by means of the fast Fourier transform method of Carr and Madan [13]. The function, `HestFFT` takes the Heston model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`) as inputs, as well as the risk-free rate of return (`r`), the dividend yield on the stock (`q`), the maturity of the option (`T`), the spot price of the underlying (`S0`) and the strike price of the option (`K`). It outputs a single option price for the call option as well as the strike price on the FFT strike grid that is closest to `K`. It is simple to extend the code to output option prices for a range of strikes — a particular advantage of the FFT pricing framework, especially for the purpose of model calibration.

```
function [hestfft strike] = HestonFFT(KAPPA,THETA,SIGMAv,RHO,V0,r,q,T,S0,K)

alpha = 0.75;
N = 2^12;
```

```
a = 600; %Upper limit of integration
eta = a/N; %Grid spacing for integration
lambda = (2*pi) / (N*eta); %Width of intervals between successive strikes
b = N*lambda / 2;

if S0 >= K %For ITM and ATM options

    u = (0:(N-1)) * eta; %Integration grid
    v = u - (alpha + 1) * 1i;

    %Characteristic Function Variables:
    ALPHA = -0.5*(v.^2 + 1i*v);
    BETA = KAPPA - RHO*SIGMAv*1i*v;
    GAMMA = (SIGMAv^2)/2;
    d = sqrt(BETA.^2 - 4*ALPHA*GAMMA);
    rpos = (BETA + d)/(SIGMAv^2);
    rneg = (BETA - d)/(SIGMAv^2);
    g = rneg./rpos;
    D = rneg .* ((1 - exp(-d*T)) ./ (1 - g.*exp(-d*T)));
    C = KAPPA * (rneg*T - (2/(SIGMAv^2)) * log((1 - g.*exp(-d*T)) ...
       ./ (1 - g)));

    %Characteristic Function and Fourier Transform
    CharFun = exp(C*THETA + D*V0 + 1i*v*(log(S0) + (r-q)*T));
    FourierTrans = (exp(-r*T) * CharFun) ./ ((alpha + 1i*u) ...
        .* (alpha + 1i*u + 1));
    SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
        %Include Simpson's weightings
    FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;

    %FFT Routine
    FFT = real(fft(FFT)); %Call MATLAB FFT routine

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    hestfft = (exp(-strikes*alpha)/pi) .* FFT; %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    hestfft = (1-(position-floor(position))) * hestfft(floor(position))...
        + (position-floor(position)) * hestfft(floor(position)+1);
        %Interpolated FFT call price

elseif S0 < K %For OTM options

    u = (0:(N-1)) * eta; %Integration grid
    v1 = u - 1i*alpha;
    v2 = u + 1i*alpha;
    w1 = u - 1i*alpha - 1i;
    w2 = u + 1i*alpha - 1i;

    %Characteristic Function 1 Variables:
    ALPHA1 = -0.5*(w1.^2 + 1i*w1);
    BETA1 = KAPPA - RHO*SIGMAv*1i*w1;
    GAMMA1 = (SIGMAv^2)/2;
    d1 = sqrt(BETA1.^2 - 4*ALPHA1*GAMMA1);
    rpos1 = (BETA1 + d1)/(SIGMAv^2);
    rneg1 = (BETA1 - d1)/(SIGMAv^2);
    g1 = rneg1./rpos1;
    D1 = rneg1 .* ((1 - exp(-d1*T)) ./ (1 - g1.*exp(-d1*T)));
    C1 = KAPPA * (rneg1*T - (2/(SIGMAv^2)) * log((1 - g1.*exp(-d1*T)) ...
        ./ (1 - g1)));

    CharFun1 = exp(C1*THETA + D1*V0 + 1i*w1*(log(S0) + (r-q)*T));

    %Characteristic Function 2 Variables:
    ALPHA2 = -0.5*(w2.^2 + 1i*w2);
    BETA2 = KAPPA - RHO*SIGMAv*1i*w2;
    GAMMA2 = (SIGMAv^2)/2;
    d2 = sqrt(BETA2.^2 - 4*ALPHA2*GAMMA2);
    rpos2 = (BETA2 + d2)/(SIGMAv^2);
    rneg2 = (BETA2 - d2)/(SIGMAv^2);
    g2 = rneg2./rpos2;
    D2 = rneg2 .* ((1 - exp(-d2*T)) ./ (1 - g2.*exp(-d2*T)));
    C2 = KAPPA * (rneg2*T - (2/(SIGMAv^2)) * log((1 - g2.*exp(-d2*T)) ...
        ./ (1 - g2)));
    CharFun2 = exp(C2*THETA + D2*V0 + 1i*w2*(log(S0) + (r-q)*T));

    %Characteristic Function and Fourier Transform
    zeta1 = exp(-r*T) * ((1./(1 + 1i*v1)) - exp(r*T)./(1i*v1) ...
        - CharFun1./(v1.^2 - 1i*v1));
    zeta2 = exp(-r*T) * ((1./(1 + 1i*v2)) - exp(r*T)./(1i*v2) ...
        - CharFun2./(v2.^2 - 1i*v2));
    FourierTrans = (zeta1 - zeta2) / 2;
    SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
        %Include Simpson's weightings
    FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;

    %FFT Routine
    FFT = real(fft(FFT));

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    hestfft = (1 ./ (pi*sinh(alpha*strikes))) .* FFT;
        %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    hestfft = (1-(position-floor(position))) * hestfft(floor(position)) ...
        + (position-floor(position)) * hestfft(floor(position)+1);
        %Interpolated FFT call price
end

strikes = -b + lambda*(0:N-1);
strike = exp(strikes(round(position))); %Strike price on strike grid
                                        %closest to required strike
end
```

FFT for the Bates Model

The following MATLAB routine computes the price of a European vanilla call option under the dynamics of the Bates model by means of the fast Fourier transform method of Carr and Madan [13]. The function, `BatesFFT` takes the Bates model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`, `MUJ`, `SIGMAS`, `LAMBDA`) as inputs, as well as the risk-free rate of return (`r`), the dividend yield on the stock (`q`), the maturity of the option (`T`), the spot price of the underlying (`S0`) and the strike price of the option (`K`). It outputs a single option price for the call option as well as the strike price on the FFT strike grid that is closest to `K`. It is simple to extend the code to output option prices for a range of strikes.

```
function [batesfft strike] = BatesFFT(KAPPA,THETA,SIGMAv,RHO,V0,LAMBDA,...
    MUJ,SIGMAS,r,q,T,S0,K)

alpha = 0.75;
N = 2^12;
a = 600; %Upper limit of integration
eta = a/N; %Grid spacing for integration
lambda = (2*pi) / (N*eta); %Width of intervals between successive strikes
b = N*lambda / 2;

if S0 >= K %For ITM and ATM options

    u = (0:(N-1)) * eta; %Integration grid
    v = u - (alpha + 1) * 1i;

    %Characteristic Function Variables:
    ALPHA = -0.5*(v.^2 + 1i*v);
    BETA = KAPPA - RHO*SIGMAv*1i*v;
    GAMMA = (SIGMAv^2)/2;
    d = sqrt(BETA.^2 - 4*ALPHA*GAMMA);
    rpos = (BETA + d)/(SIGMAv^2);
    rneg = (BETA - d)/(SIGMAv^2);
    g = rneg./rpos;
    D = rneg .* ((1 - exp(-d*T)) ./ (1 - g.*exp(-d*T)));
    C = KAPPA * (rneg*T - (2/(SIGMAv^2)) * log((1 - g.*exp(-d*T))...
        ./ (1 - g)));

    %Characteristic Function and Fourier Transform
    CharFun = exp(C*THETA + D*V0 + 1i*v*(log(S0) + (r-q)*T));
        %Diffusion characteristic function
    JumpCFn = exp(-LAMBDA*MUJ*1i*v*T + LAMBDA*T * ((1 + MUJ).^(1i.*v)...
        .* exp(SIGMAS^2*(1i.*v/2).*(1i.*v - 1)) - 1));
        %Jump characteristic function
    FourierTrans = (exp(-r*T) * CharFun .* JumpCFn) ./ ((alpha + 1i*u)...
        .* (alpha + 1i*u + 1));
    SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
        %Include Simpson's weightings
    FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;

    %FFT Routine
    FFT = real(fft(FFT)); %Call MATLAB FFT routine

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    batesfft = (exp(-strikes*alpha)/pi) .* FFT; %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    batesfft = (1-(position-floor(position))) * ...
        batesfft(floor(position)) + (position-floor(position)) ...
        * batesfft(floor(position)+1);
        %Interpolated FFT call price

elseif S0 < K %For OTM options

    u = (0:(N-1)) * eta; %Integration grid
    v1 = u - 1i*alpha;
    v2 = u + 1i*alpha;
    w1 = u - 1i*alpha - 1i;
    w2 = u + 1i*alpha - 1i;

    %Characteristic Function 1 Variables
    ALPHA1 = -0.5*(w1.^2 + 1i*w1);
    BETA1 = KAPPA - RHO*SIGMAv*1i*w1;
    GAMMA1 = (SIGMAv^2)/2;
    d1 = sqrt(BETA1.^2 - 4*ALPHA1*GAMMA1);
    rpos1 = (BETA1 + d1)/(SIGMAv^2);
    rneg1 = (BETA1 - d1)/(SIGMAv^2);
    g1 = rneg1./rpos1;
    D1 = rneg1 .* ((1 - exp(-d1*T)) ./ ...
        (1 - g1.*exp(-d1*T)));
```

```
C1 = KAPPA * (rneg1*T - (2/(SIGMAv^2)) * ...
    log((1 - g1.*exp(-d1*T)) ./ (1 - g1)));

JumpCFn1 = exp(-LAMBDA*MUJ*1i*w1*T + LAMBDA*T * ((1 + MUJ).^(1i*w1)...
    .* exp(SIGMAS^2*(1i.*w1/2).*(1i.*w1 - 1)) - 1));
    %Jump characteristic function
CharFun1 = exp(C1*THETA + D1*V0 + 1i*w1*(log(S0) + (r-q)*T)) ...
    .* JumpCFn1;

%Characteristic Function 2 Variables
ALPHA2 = -0.5*(w2.^2 + 1i*w2);
BETA2 = KAPPA - RHO*SIGMAv*1i*w2;
GAMMA2 = (SIGMAv^2)/2;
d2 = sqrt(BETA2.^2 - 4*ALPHA2*GAMMA2);
rpos2 = (BETA2 + d2)/(SIGMAv^2);
rneg2 = (BETA2 - d2)/(SIGMAv^2);
g2 = rneg2./rpos2;
D2 = rneg2 .* ((1 - exp(-d2*T)) ./ ...
    (1 - g2.*exp(-d2*T)));
C2 = KAPPA * (rneg2*T - (2/(SIGMAv^2)) * ...
    log((1 - g2.*exp(-d2*T)) ./ (1 - g2)));

JumpCFn2 = exp(-LAMBDA*MUJ*1i*w2*T + LAMBDA*T * ((1 + MUJ).^(1i*w2)...
    .* exp(SIGMAS^2*(1i.*w2/2).*(1i.*w2 - 1)) - 1));
    %Jump characteristic function
CharFun2 = exp(C2*THETA + D2*V0 + 1i*w2*(log(S0) + (r-q)*T)) ...
    .* JumpCFn2;

%Characteristic Function and Fourier Transform
zeta1 = exp(-r*T) * ((1./(1 + 1i*v1)) - exp(r*T)./(1i*v1) ...
    - CharFun1./(v1.^2 - 1i*v1));
zeta2 = exp(-r*T) * ((1./(1 + 1i*v2)) - exp(r*T)./(1i*v2) ...
    - CharFun2./(v2.^2 - 1i*v2));
FourierTrans = (zeta1 - zeta2) / 2;
SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
    %Include Simpson's weightings
FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;
```

```
    %FFT Routine
    FFT = real(fft(FFT)); %Call MATLAB FFT routine

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    batesfft = (1 ./ (pi*sinh(alpha*strikes))) .* FFT;
        %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    batesfft = (1-(position-floor(position))) * ...
        batesfft(floor(position)) + (position-floor(position)) ...
        * batesfft(floor(position)+1);
        %Interpolated FFT call price
end

strikes = -b + lambda*(0:N-1);
strike = exp(strikes(round(position))); %Strike price on strike grid
                                        %closest to required strike
```

FFT for the SVJJ Model

The following MATLAB routine computes the price of a European vanilla call option under the dynamics of the SVJJ model by means of the fast Fourier transform method of Carr and Madan [13]. The function, `SVJJFFT` takes the SVJJ model parameters (`KAPPA`, `THETA`, `SIGMAv`, `RHO`, `V0`, `LAMBDAJ`, `MUS`, `SIGMAS`, `MUV`, `RHOJ`) as inputs, as well as the risk-free rate of return (`r`), the dividend yield on the stock (`q`), the maturity of the option (`T`), the spot price of the underlying (`S0`) and the strike price of the option (`K`). It outputs a single option price for the call option as well as the strike price on the FFT strike grid that is closest to `K`. It is simple to extend the code to output option prices for a range of strikes.

```
function [svjjfft strike] = SVJJFFT(KAPPA,THETA,SIGMAv,RHO,V0,LAMBDAJ,...
    MUS,SIGMAS,RHOJ,MUV,r,q,T,S0,K)

alpha = 0.75;
N = 2^12;
a = 600; %Upper limit of integration
eta = a/N; %Grid spacing for integration
lambda = (2*pi) / (N*eta); %Width of intervals btw successive strikes
```

```
b = N*lambda / 2;

if S0 >= K %For ITM and ATM options

    u = (0:(N-1)) * eta; %Integration grid
    v = u - (alpha + 1) * 1i;

    %Characteristic Function Variables
    %Diffusion Variables
    ALPHA = -0.5*(v.^2 + 1i*v);
    BETA = KAPPA - RHO*SIGMAv*1i*v;
    GAMMA = (SIGMAv^2)/2;
    d = sqrt(BETA.^2 - 4*ALPHA*GAMMA);
    rpos = (BETA + d)/(SIGMAv^2);
    rneg = (BETA - d)/(SIGMAv^2);
    g = rneg./rpos;
    D = rneg .* ((1 - exp(-d*T)) ./ (1 - g.*exp(-d*T)));
    C = KAPPA * (rneg*T - (2/(SIGMAv^2)) * log((1 - g.*exp(-d*T)) ...
        ./ (1 - g)));

    %Jump Variables
    MUJ = exp(MUS + 0.5*SIGMAS^2) / (1 - RHOJ*MUV) - 1;
    c = 1 - RHOJ*MUV*1i*v;
    nu = ( (BETA + d) ./ ((BETA + d).*c - 2*MUV*ALPHA) ) * T + ...
        ( (4*MUV*ALPHA) ./ ((d.*c).^2 - (2*MUV*ALPHA - BETA.*c) ...
        .^2) ) .* log( 1 - ( ((d-BETA).*c + 2*MUV*ALPHA) ./ ...
        (2*d.*c) ).*(1 - exp(-d*T)) );
    P = -T*(1 + MUJ*1i*v) + exp( MUS*1i*v + 0.5*(SIGMAS^2)*(1i*v).^2 ).*nu;

    %Characteristic Function and Fourier Transform
    CharFun = exp(C*THETA + D*V0 + P*LAMBDAJ + 1i*v*(log(S0) + (r-q)*T));
    FourierTrans = (exp(-r*T) * CharFun) ./ ((alpha + 1i*u) ...
        .* (alpha + 1i*u + 1));
    SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
        %Include Simpson's weightings
    FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;
    %FFT Routine
    FFT = real(fft(FFT)); %Call MATLAB FFT routine

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    svjjfft = (exp(-strikes*alpha)/pi) .* FFT; %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    svjjfft = (1-(position-floor(position))) * ...
        svjjfft(floor(position)) + (position-floor(position)) ...
        * svjjfft(floor(position)+1);
        %Interpolated FFT call price

elseif S0 < K %For OTM options

    u = (0:(N-1)) * eta; %Integration grid
    v1 = u - 1i*alpha;
    v2 = u + 1i*alpha;
    w1 = u - 1i*alpha - 1i;
    w2 = u + 1i*alpha - 1i;

    %Characteristic Function 1 Variables
    %Diffusion Variables
    ALPHA1 = -0.5*(w1.^2 + 1i*w1);
    BETA1 = KAPPA - RHO*SIGMAv*1i*w1;
    GAMMA1 = (SIGMAv^2)/2;
    d1 = sqrt(BETA1.^2 - 4*ALPHA1*GAMMA1);
    rpos1 = (BETA1 + d1)/(SIGMAv^2);
    rneg1 = (BETA1 - d1)/(SIGMAv^2);
    g1 = rneg1./rpos1;
    D1 = rneg1 .* ((1 - exp(-d1*T)) ./ (1 - g1.*exp(-d1*T)));
    C1 = KAPPA * (rneg1*T - (2/(SIGMAv^2)) * log((1 - g1.*exp(-d1*T)) ...
        ./ (1 - g1)));

    %Jump Variables
    MUJ1 = exp(MUS + 0.5*SIGMAS^2) / (1 - RHOJ*MUV) - 1;
    c1 = 1 - RHOJ*MUV*1i*w1;
    nu1 = ( (BETA1 + d1) ./ ((BETA1 + d1).*c1 - 2*MUV*ALPHA1) ) * T + ...
        ( (4*MUV*ALPHA1) ./ ((d1.*c1).^2 - (2*MUV*ALPHA1 - BETA1.*c1) ...
        .^2) ) .* log( 1 - ( ((d1-BETA1).*c1 + 2*MUV*ALPHA1) ./ ...
        (2*d1.*c1) ) .* (1 - exp(-d1*T)) );
    P1 = -T*(1 + MUJ1*1i*w1) + exp( MUS*1i*w1 + ...
        0.5*(SIGMAS^2)*(1i*w1).^2 ).*nu1;

    CharFun1 = exp(C1*THETA + D1*V0 + P1*LAMBDAJ + ...
        1i*w1*(log(S0) + (r-q)*T));

    %Characteristic Function 2 Variables
    ALPHA2 = -0.5*(w2.^2 + 1i*w2);
    BETA2 = KAPPA - RHO*SIGMAv*1i*w2;
    GAMMA2 = (SIGMAv^2)/2;
    d2 = sqrt(BETA2.^2 - 4*ALPHA2*GAMMA2);
    rpos2 = (BETA2 + d2)/(SIGMAv^2);
    rneg2 = (BETA2 - d2)/(SIGMAv^2);
    g2 = rneg2./rpos2;
    D2 = rneg2 .* ((1 - exp(-d2*T)) ./ (1 - g2.*exp(-d2*T)));
    C2 = KAPPA * (rneg2*T - (2/(SIGMAv^2)) * log((1 - g2.*exp(-d2*T)) ...
        ./ (1 - g2)));

    %Jump Variables
    MUJ2 = exp(MUS + 0.5*SIGMAS^2) / (1 - RHOJ*MUV) - 1;
    c2 = 1 - RHOJ*MUV*1i*w2;
    nu2 = ( (BETA2 + d2) ./ ((BETA2 + d2).*c2 - 2*MUV*ALPHA2) ) * T + ...
        ( (4*MUV*ALPHA2) ./ ((d2.*c2).^2 - (2*MUV*ALPHA2 - BETA2.*c2) ...
        .^2) ) .* log( 1 - ( ((d2-BETA2).*c2 + 2*MUV*ALPHA2) ./ ...
        (2*d2.*c2) ) .* (1 - exp(-d2*T)) );
    P2 = -T*(1 + MUJ2*1i*w2) + exp( MUS*1i*w2 + ...
        0.5*(SIGMAS^2)*(1i*w2).^2 ).*nu2;

    CharFun2 = exp(C2*THETA + D2*V0 + P2*LAMBDAJ + ...
        1i*w2*(log(S0) + (r-q)*T));

    %Characteristic Function and Fourier Transform
    zeta1 = exp(-r*T) * ((1./(1 + 1i*v1)) - exp(r*T)./(1i*v1) ...
        - CharFun1./(v1.^2 - 1i*v1));
    zeta2 = exp(-r*T) * ((1./(1 + 1i*v2)) - exp(r*T)./(1i*v2) ...
        - CharFun2./(v2.^2 - 1i*v2));
    FourierTrans = (zeta1 - zeta2) / 2;
    SWeightings = (1/3) * (3 + (-1).^(1:N) - [1 zeros(1,N-1)]);
        %Include Simpson's weightings
    FFT = exp(1i*b*u) .* FourierTrans * eta .* SWeightings;

    %FFT Routine
    FFT = real(fft(FFT)); %Call MATLAB FFT routine

    %Call Price Calculation
    strikes = -b + lambda*(0:N-1); %Log-strike price grid
    svjjfft = (1 ./ (pi*sinh(alpha*strikes))) .* FFT;
       %Include dampening factor
    position = (log(K) + b) / lambda + 1; %Strike position on grid
    svjjfft = (1-(position-floor(position))) * ...
        svjjfft(floor(position)) + (position-floor(position)) ...
        * svjjfft(floor(position)+1);
        %Interpolated FFT call price
end

strikes = -b + lambda*(0:N-1);
strike = exp(strikes(round(position))); %Strike price on strike grid
                                        %closest to required strike
```

E.3 The Genetic Algorithm

The following is our implementation of the genetic algorithm. The routine requires the input of an objective function to be optimised, the number of unknowns in the optimisation problem, the parameter bounds for the objective function as well as a variable declaring whether a minimisation or maximisation is being undertaken. It outputs the optimised function parameters as well as the running time of the algorithm. It also produces statistics pertaining to the fitness of individuals across generations.

```
 function [Output Time MeanFitness SumFitness MaxFitness MinFitness] = ...
    GeneticAlgorithm(ObjectiveFunction,NoOfUnknowns,...
                     UpperBounds,LowerBounds,minmax)
```

```
Tstart = clock; %Start algorithm timer

% *** ALGORITHM INPUTS ***

% Objective Function - The function upon which the GA will act
% NoOfUnknowns - Number of model parameters
% UpperBounds - Upper bounds for the unknowns / model parameters
% LowerBounds - Lower bounds for the unknowns / model parameters
% minmax - Minimise of maximise the function. Set to 'MIN' or 'MAX'.

% *** ALGORITHM SETTINGS ***

disp('Configuring Settings.')
PopSize = 1000; %Number of individuals in the population.
SubStrLength = 100; %Binary string length for each unknown.
TotStrLength = NoOfUnknowns * SubStrLength;
MaxNoOfGens = 100; %Maximum number of generations.
CrossOverProb = 0.9; %Probability associated with crossover.
MutationProb = (1 / TotStrLength) * 20; %Prob associated with mutation.
Elitism = 'on'; %Turn elitism 'on' or 'off'.
TournSelectionProb = 0.75; %Prob of selecting fittest individual in
                           %tournament selection
CONST = 1000; %Set constant used to turn minimisation into maximisation.

% *** INITIAL POPULATION - GENERATION 1 ***

disp('Creating Initial Population.')
Population = round(rand(PopSize,TotStrLength)); %Population matrix as
    %a bit string. There are as many rows as population size and cols
    %as total string length.
IntPop = zeros(PopSize,NoOfUnknowns); %Integer values for
                                      %population strings.
for bit = 1:NoOfUnknowns
    IntPop(1:end,bit) = sum( Population(1:end,(bit-1)...
        *SubStrLength+1:(bit)*SubStrLength)...
        *2.^( ones(PopSize,1)*(SubStrLength-1:-1:0) ),2 );
        %Converts binary digits into integer values.
end
BoundDiffs = UpperBounds - LowerBounds;
    %Difference between upper and lower bounds.
RealPop =  ones(PopSize,1) * (BoundDiffs / (2^(SubStrLength) - 1))...
    .* IntPop + ones(PopSize,1) * LowerBounds;
    %Calculate the real value for each unknown.

% *** EVALUATE FITNESS OF INDIVIDUALS ***

disp('Evaluating Initial Fitness.')

Fitness = zeros(PopSize,1);
if strcmp('MAX',minmax) %Fitness function for each individual.
    for individ = 1:PopSize
        Fitness(individ) = max( ObjectiveFunction ...
            (RealPop(individ,:)) , 0 );
    end
elseif strcmp('MIN',minmax)
    for individ = 1:PopSize
        Fitness(individ) = max( CONST - ObjectiveFunction ...
            (RealPop(individ,:)), 0);
    end
else
    disp('minmax incorrectly specified.')
end

MeanFitness = [mean(Fitness) zeros(1,MaxNoOfGens-1)];
    %Average fitness of population.
MaxFitness = [max(Fitness) zeros(1,MaxNoOfGens-1)];
    %Maximum fitness value.
MinFitness = [min(Fitness) zeros(1,MaxNoOfGens-1)];
    %Minimum fitness value.
SumFitness = [sum(Fitness) zeros(1,MaxNoOfGens-1)];
    %Total fitness of population.

% *** RUNNING TIMER ***
```

```
disp('****************************')
disp(['Generation ',num2str(1)])
disp('-------------------')
disp(['Running Time: ',num2str( etime(clock,Tstart) )])
disp(['Time Remaining: ',...
    num2str( (etime(clock,Tstart)) * (MaxNoOfGens-1) )])
disp('****************************')

% *** GENETIC ALGORITHM ROUTINE ***

disp('Running the Algorithm.')
for gen = 2:MaxNoOfGens

    NewPopulation = zeros(PopSize,TotStrLength);
    NewNewPopulation = zeros(PopSize,TotStrLength);

    % *** SELECTION ***

    NoOfFittest = 0.2 * PopSize;
    NoInTournament = 0.7 * PopSize;
    NoNew = 0.1 * PopSize;

    %Select Fittest Individuals
    [OrderedFitness Index] = sort(Fitness,'descend');
    for fit = 1:NoOfFittest
        NewPopulation(fit,:) = Population(Index(fit),:);
    end

    %Tournament Selection
    TournIndividuals = 4;

    for TournRound = 1:NoInTournament
        TournRnd = floor(rand(1,TournIndividuals)*PopSize) + 1;
        [OrdrderedFitness Index] = sort(Fitness(TournRnd),'descend');
        for fighter = 1:TournIndividuals
            if rand <= TournSelectionProb
```

```
                Winner = fighter;
                break
            elseif fighter == TournIndividuals
                Winner = fighter;
            end
        end
        NewPopulation(TournRound+NoOfFittest,:) = ...
            Population(TournRnd(Index(Winner)),:);
end

%New Individuals
NewPopulation(NoOfFittest+NoInTournament+1:end,1:end) = ...
    round(rand(NoNew,TotStrLength));

RandNos = rand(1,PopSize);
[SortRandNos RandomArrangement] = sort(RandNos);
NewPopulation = NewPopulation(RandomArrangement,1:end);

% *** CROSSOVER ***

NoOfCrossovers = 5;
for individ = 1:PopSize/2
    NewIndivid1 = NewPopulation(2*(individ-1) + 1,1:end);
    NewIndivid2 = NewPopulation(2*individ,1:end);
    if rand <= CrossOverProb %Crossover with given probability.
        CrossSites = floor((TotStrLength - 1) * ...
            rand(1,NoOfCrossovers)) + 1; %Select a point to crossover.
        CrossSites = [0 sort(CrossSites) TotStrLength];
        NewNewIndivid1 = [];
        NewNewIndivid2 = [];
        %Perform crossover
        for cross = 2:length(CrossSites);
            if mod(cross,2) == 0
                A = NewIndivid1;
                B = NewIndivid2;
            else
                A = NewIndivid2;
                B = NewIndivid1;
            end
            NewNewIndivid1 = [NewNewIndivid1 A(CrossSites(cross-1)...
                +1:CrossSites(cross))];
            NewNewIndivid2 = [NewNewIndivid2 B(CrossSites(cross-1)...
                +1:CrossSites(cross))];
        end
        NewNewPopulation(2*(individ-1) + 1,1:end) = NewNewIndivid1;
            %Create new population.
        NewNewPopulation(2*individ,1:end) = NewNewIndivid2;
            %Create new population.
    else %No crossover.
        NewNewPopulation(2*(individ-1) + 1,1:end) = NewIndivid1;
            %Create new population.
        NewNewPopulation(2*individ,1:end) = NewIndivid2;
            %Create new population.
    end
end

NewPopulation = NewNewPopulation;

% *** MUTATION ***

MutationMatrix = rand(PopSize,TotStrLength) <= MutationProb;
    %Matrix displaying bits to be flipped.
NewPopulation = NewPopulation + MutationMatrix;
NewPopulation = (NewPopulation) .* (NewPopulation < 2);
    %Mutated population.

% *** EVALUATE REAL VALUES OF NEW POPULATION ***

NewIntPop = zeros(PopSize,NoOfUnknowns);
    %Integer values for population strings.
for bit = 1:NoOfUnknowns
    NewIntPop(1:end,bit) = sum( NewPopulation(1:end,(bit-1)...
        *SubStrLength+1:(bit)*SubStrLength)...
        .*2.^( ones(PopSize,1)*(SubStrLength-1:-1:0) ),2 );
```

```
            %Converts binary digits into integer values.
    end
    NewRealPop =  ones(PopSize,1) * (BoundDiffs ...
        / (2^(SubStrLength) - 1)) .* NewIntPop ...
        + ones(PopSize,1) * LowerBounds;
        %Calculate the real value for each unknown.

    % *** EVALUATE FITNESS OF NEW POPULATION ***

    NewFitness = zeros(PopSize,1);
    if strcmp('MAX',minmax) %Fitness function for each individual.
        for individ = 1:PopSize
            NewFitness(individ) = max( ObjectiveFunction...
                (NewRealPop(individ,:)) , 0 );
        end
    elseif strcmp('MIN',minmax)
        for individ = 1:PopSize
            NewFitness(individ) = max( CONST - ...
                ObjectiveFunction(NewRealPop(individ,:)), 0);
        end
    end

    % *** ELITISM ***

    if strcmp(Elitism,'on');
        %Test if old elite individual is fitter than new elite individual.
        if MaxFitness(gen-1) > max(NewFitness)
            RandIndivid = floor(rand*(PopSize - 1)) + 1;
                %Choose random individual to replace.
            EliteIndivid = sum( (MaxFitness(gen-1) == Fitness)*...
                ones(1,TotStrLength) .* Population ); %Find elite individual.
            NewPopulation(RandIndivid,1:end) = EliteIndivid;
                %Keep elite individual string.
            NewFitness(RandIndivid) = MaxFitness(gen-1);
                %Keep elite individual fitness.
            NewRealPop(RandIndivid,1:end) = sum( (MaxFitness(gen-1) ...
                == Fitness)*ones(1,NoOfUnknowns) .* RealPop);
                %Keep the values of the unknowns attributed
                %to the elite individual.
        end
    end

    % *** REPLACE OLD POPULATION ***

    Population = NewPopulation;
    Fitness = NewFitness;
    RealPop = NewRealPop;

    MeanFitness(gen) = mean(Fitness); %Average fitness of population.
    MaxFitness(gen) = max(Fitness); %Maximum fitness value.
    MinFitness(gen) = min(Fitness); %Minimum fitness value.
    SumFitness(gen) = sum(Fitness); %Total fitness of population.

    % *** RUNNING TIMER ***

    disp('****************************')
    disp(['Generation ',num2str(gen)])
    disp('-------------------')
    disp(['Running Time: ',num2str( etime(clock,Tstart) )])
    disp(['Time Remaining: ',...
        num2str( (etime(clock,Tstart)/(gen)) * (MaxNoOfGens-(gen)) )])
    disp('****************************')

end

[OrderedFitness Index] = sort(Fitness,'descend');
NoOfParOutputs = 5; %Number of fittest individuals selected
Output = RealPop(Index(1:NoOfParOutputs),1:end);
    %Values of unknowns that optimise the objective function.

Tend = clock; %Stop algorithm timer.
Time = etime(Tend,Tstart); %Calculate time taken to run algorithm.
```

Bibliography

[1] Attari, M., *Option Pricing Using Fourier Transforms: A Numerically Efficient Simplification*, Charles River Associates Inc. Working Paper (2004).

[2] Bäck, T., Hammel, U., Schwefel, H-P., *Evolutionary Computation: Comments on the History and Current State*, IEEE Transactions on Evolutionary Computation **1** (1997), no. 1, 3–17.

[3] Bakshi, G., Cao, C., Chen, Z., *Empirical Performance of Alternative Option Pricing Models*, The Journal of Finance **52** (1997), no. 5, 2003–2049.

[4] Barndorff-Nielsen, O., Shepard, N., *Non-Gaussian OU Based Models and Some of Their Uses in Financial Economics*, OFRC Working Papers Series, Oxford Financial Research Centre (2000).

[5] Bates, D., *Jumps and Stochastic Volatility: Exchange Rates Processes Implicit in Deutsche Mark Options*, The Review of Financial Studies **9** (1996), no. 1, 69–107.

[6] Bingham, N.H., Kiesel, R., *Risk-Neutral Valuation: Pricing and Hedging of Financial Derivatives*, 2 ed., Springer-Verlag, 2004.

[7] Black, F., Scholes, M., *The Pricing of Options and Corporate Liabilities*, The Journal of Political Economy **81** (1973), 631–659.

[8] Borak, S., Detlefsen , K., Härdle, W., *FFT Based Option Pricing*, Discussion Paper: Center for Applied Statistics and Economics (2005).

[9] Brigo, D., Dalessandro, A., Neugebauer, M., Triki, F., *A Stochastic Processes Toolkit for Risk Management*, The Journal of Risk Management for Financial Institutions (2007).

[10] Broadie, M., Chernov, M., Johannes, M., *Model Specification and Risk Premia: Evidence from Futures Options*, The Journal of Finance **62** (2007), no. 3, 1453–1490.

[11] Broadie, M., Kaya, Ö., *Exact Simulation of Stochastic Volatility and Other Affine Jump Diffusion Processes*, Operations Research **54** (2006), no. 2, 217–231.

[12] Carr, P., Geman, H., Madan, D., Yor, M., *Stochastic Volatility for Lévy Processes*, The Journal of Mathematical Finance **13** (2003), no. 3, 345–382.

[13] Carr, P., Madan, D., *Option Valuation Using the Fast Fourier Transform*, The Journal of Computational Finance **2** (1999), no. 4, 61–73.

[14] ______, *A Note on Sufficient Conditions for No Arbitrage*, Finance Research Letters **2** (2005), no. 3, 125–130.

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

[16] Chernov, M., Ghysels, E., *A Study Towards a Unified Approach to the Joint Estimation of Objective and Risk Neutral Measures for the Purpose of Options Valuation*, The Journal of Financial Economics **56** (2000), 407–458.

[17] Chicago Board Options Exchange, *S&P 500 Index Options Specifications on the CBOE Website*, Accessed at http://www.cboe.com.

[18] Coley, D., *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, 2010.

[19] Cont, R., *Recovering Volatility from Option Prices by Evolutionary Optimisation*, Research Paper Series (2005).

[20] Cont, R., Tankov, P., *Financial Modelling with Jump Processes*, Chapman & Hall/CRC Financial Mathematics Series, The United States of America, 2004.

[21] Cox, J., Ingersoll, J., Ross, S., *A Theory of the Term Structure of Interest Rates*, Econometrica **53** (1985), no. 2, 385–408.

[22] Duffie, D., Pan J., Singleton K., *Transform Analysis and Asset Pricing for Affine Jump Diffusions*, Econometrica **68** (2000), 1343–1376.

[23] Fang, F., Oosterlee, C.W., *A Novel Method for European Options Based on Fourier-Cosine Series Expansions*, SIAM Journal on Scientific Computing **31** (2008), no. 2, 826–848.

[24] Fouque, J.P., Papanicolaou, G., Sircar, K.R., *Derivatives in Financial Markets with Stochastic Volatility*, Cambridge University Press, United Kingdom, 2001.

[25] Gatheral, J., *The Volatility Surface: A Practitioners Guide*, John Wiley and Sons, Inc., The United States of America, 2006.

[26] Hagan, P., Kumar, D., Lesniewski, A., Woodward, D., *Managing Smile Risk*, Wilmott Magazine (2002), 84–108.

[27] He, C., Kennedy, J., Coleman, T., Forsyth, P., Li, Y., Vetzal, K., *Calibration and Hedging Under Jump Diffusion*, Review of Derivatives Research **9** (2006), no. 1, 1–35.

[28] Heston, S., *A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options*, The Review of Financial Studies **6** (1993), no. 2, 327–343.

[29] Hong, G., *Forward Smile and Derivative Pricing*, (2004).

[30] Ingber, L., *Lester Ingber's Archive*, Accessed at http://www.ingber.com.

[31] ______, *Adaptive Simulated Annealing (Readme File)*, 1993-2010, Accessed at http://www.ingber.com.

[32] ______, *Adaptive Simulated Annealing (ASA): Lessons Learned*, Control and Cybernetics (1995).

[33] Ingber, L., Rosen, B., *Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison*, Mathematical and Computer Modelling **16** (1992), no. 11, 87–100.

[34] Ingber, L., Wilson, J., *Statistical Mechanics of Financial Markets: Exponential Modifications to Black-Scholes*, Mathematical Computer Modelling **31** (2000), no. 8, 167–192.

[35] Kilin, F., *Accelerating the Calibration of Stochastic Volatility Models*, CPQF Working Paper, Frankfurt School of Finance and Management (2007).

[36] Kloeden, P., Platen, E, *Numerical Solution of Stochastic Differential Equations*, Springer-Verlag, 1992.

[37] Kotzé, A., Joseph, A., *Constructing a South African Implied Volatility Surface from Exchange Traded Data*, Tech. report, November 2009.

[38] Kurpiel, A., Roncalli, T., *Option Hedging with Stochastic Volatility*, FERC Working Paper, City University Business School (1998).

[39] Lee, R., *Option Pricing by Transform Methods: Extensions, Unification, and Error Control*, The Journal of Computational Finance **7** (2004), no. 3, 51–86.

[40] Lord, R., Koekkoek, R., van Dijk, D, *A Comparison of Biased Simulation Schemes for Stochastic Volatility Models*, Working paper: Tinbergen Institute (2006).

[41] Medvedev, A., Scaillet, O., *A Simple Calibration Procedure of Stochastic Volatility Models with Jumps by Short Term Asymptotics*, Discussion Paper HEC, Genève and FAME, Université de Genève (2004).

[42] Merton, R., *Theory of Rational Option Pricing*, The Bell Journal of Economics and Management Science **4** (1973), no. 1, 141–183.

[43] Mikhailov, S., Nögel, U., *Heston's Stochastic Volatility Model Implementation, Calibration and Some Extensions*, Wilmott Magazine (2003), 74–79.

[44] Moins, S., *Implementation of a Simulated Annealing Algorithm for MATLAB*, Tech. report, Linköping Institute of Technology, August 2002.

[45] Moodley, N., *The Heston Model: A Practical Approach*, Honours Thesis: The University of the Witwatersrand (2005).

[46] Pakel, C., Shephard, N., Sheppard, K., *Nuisance Parameters, Composite Likelihoods, and a Panel of GARCH Models*, Statistica Sinica **21** (2011), 307–329.

[47] Pillay, E., *FFT Based Option Pricing under Mean Reversion, Jumps and Stochastic Volatility*, Masters Thesis: The University of KwaZulu-Natal (2009).

[48] Poklewski-Koziell, W., *Convergence of Monte Carlo Methods for the Heston Stochastic Volatility Model*, Honours Thesis: The University of the Witwatersrand (2009).

[49] Putschögl, W., *On Calibrating Stochastic Volatility Models with Time-Dependent Parameters*, ArXiv e-prints (2010).

[50] Sakata, S., *Shinichi Sakata's Webpage*, http://ssakata.sdf.org/.

[51] Schmelzle, M., *Option Pricing Formulae using Fourier Transform: Theory and Application*, April 2010, Accessed at http://pfadintegral.com.

[52] Schoutens, W., Simons, E., Tistaert, J., *A Perfect Calibration! Now What?*, Wilmott Magazine (2004).

[53] Shreve, S., *Stochastic Calculus for Finance 2 - Continuous Time Models*, Springer Finance, 2008.

[54] South African Futures Exchange, *ALSI Futures Option Data on the SAFEX Website*, Accessed at http://www.safex.co.za.

[55] The JSE Limited, *JSE Equity Options Explained*, Tech. report, November 2009, Accessed at http://www.jse.co.za.

[56] The Mathworks, *Help Files on the Mathworks Website - lsqnonlin*, Accessed at http://www.mathworks.com.

[57] Walker, J., *Fast Fourier Transforms*, CRC Press, Inc., The United States of America, 1991.

[58] West, G., *Calibration of the SABR Model in Illiquid Markets*, Applied Mathematical Finance **12** (2005), no. 4, 371–385.

[59] Zhu, J., *Applications of Fourier Transform to Smile Modelling: Theory and Implementation*, Springer Finance, 2010.